33rd Annual European Symposium on Algorithms

ESA 2025, September 15–17, 2025, Warsaw, Poland

Edited by

Anne Benoit

Haim Kaplan

Sebastian Wild

Grzegorz Herman



LIPIcs – Vol. 351 – ESA 2025 www.dagstuhl.de/lipics

Editors

Anne Benoit
ENS Lyon, France
Anne.Benoit@ens-lyon.fr

Haim Kaplan
Tel Aviv University, Israel
haimk@post.tau.ac.il

Sebastian Wild
University of Marburg, Germany
University of Liverpool, UK
wild@informatik.uni-marburg.de

Grzegorz Herman
Jagiellonian University, Kraków, Poland
grzegorz.herman@uj.edu.pl

ACM Classification 2012
Applied computing → Bioinformatics; Applied computing → Computational biology; Applied computing → Sequencing and genotyping technologies; Computing methodologies → Motion path planning; Computing methodologies → Search methodologies; Information systems → Point lookups; Mathematics of computing → Approximation Algorithms; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Combinatoric problems; Mathematics of computing → Combinatorics; Mathematics of computing → Combinatorics on words; Mathematics of computing → Discrete optimization; Mathematics of computing → Geometric topology; Mathematics of computing → Graph algorithms; Mathematics of computing → Graph coloring; Mathematics of computing → Graphs and surfaces; Mathematics of computing → Graph theory; Mathematics of computing → Hypergraphs; Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Permutations and combinations; Mathematics of computing → Probability and statistics; Mathematics of computing → Random graphs; Mathematics of computing → Stochastic processes; Mathematics of computing → Submodular optimization and polymatroids; Security and privacy; Theory of computation → Approximation algorithms analysis; Theory of computation → Bloom filters and hashing; Theory of computation → Complexity classes; Theory of computation → Computational complexity and cryptography; Theory of computation → Computational geometry; Theory of computation → Data compression; Theory of computation → Data structures design and analysis; Theory of Computation → Design and analysis of algorithms; Theory of computation → Distributed algorithms; Theory of computation → Dynamic graph algorithms; Theory of computation → Facility location and clustering; Theory of computation → Finite Model Theory; Theory of computation → Fixed parameter tractability; Theory of computation → Generating random combinatorial structures; Theory of computation → Graph algorithms analysis; Theory of computation → Integer programming; Theory of computation → Massively parallel algorithms; Theory of computation → Network flows; Theory of computation → Online algorithms; Theory of computation → Packing and covering problems; Theory of computation → Parallel algorithms; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Pattern matching; Theory of computation → Problems, reductions and completeness; Theory of computation → Pseudorandomness and derandomization; Theory of computation → Quantum complexity theory; Theory of computation → Quantum computation theory; Theory of computation → Quantum information theory; Theory of computation → Quantum query complexity; Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Shared memory algorithms; Theory of computation → Shortest paths; Theory of computation → Sorting and searching; Theory of computation → Sparsification and spanners; Theory of computation → Streaming models; Theory of computation → Streaming, sublinear and near linear time algorithms

ISBN 978-3-95977-395-9

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-395-9.

Publication date
September, 2025

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists all publications of this volume in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

License
This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/legalcode.
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.



Digital Object Identifier: 10.4230/LIPIcs.ESA.2025.0

LIPIcs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université Paris Cité, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Holger Hermanns (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Daniel Král' (Leipzig University, DE and Max Planck Institute for Mathematics in the Sciences, Leipzig, DE)
- Sławomir Lasota (University of Warsaw, PL)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN – Chair)
- Chih-Hao Luke Ong (Nanyang Technological University, SG)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Pierre Senellart (ENS, Université PSL, Paris, France)
- Alexandra Silva (Cornell University, Ithaca, US)

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman 0:xv

Program Committees
..... 0:xvii

External Reviewers
..... 0:xix

Invited Talks

Graph Decompositions and Length-Constrained Expanders
Bernhard Haeupler 1:1–1:2

Securing Dynamic Data: A Primer on Differentially Private Data Structures
Monika Henzinger and Roodabeh Safavi 2:1–2:14

Regular Papers

Generalized Graph Packing Problems Parameterized by Treewidth
Barış Can Esmer and Dániel Marx 3:1–3:15

Optimal Quantum Algorithm for Estimating Fidelity to a Pure State
Wang Fang and Qisheng Wang 4:1–4:12

External-Memory Priority Queues with Optimal Insertions
Gerth Stølting Brodal, Michael T. Goodrich, John Iacono, Jared Lo, Ulrich Meyer, Victor Pagan, Nodari Sitchinava, and Rolf Svenning 5:1–5:14

On the Complexity of Knapsack Under Explorable Uncertainty: Hardness and Algorithms
Jens Schlöter 6:1–6:15

Graph Modification of Bounded Size to Minor-Closed Classes as Fast as Vertex Deletion
Laure Morelle, Ignasi Sau, and Dimitrios M. Thilikos 7:1–7:18

Fast Computation of k-Runs, Parameterized Squares, and Other Generalised Squares
Yuto Nakashima, Jakub Radoszewski, and Tomasz Waleń 8:1–8:18

MorphisHash: Improving Space Efficiency of ShockHash for Minimal Perfect Hashing
Stefan Hermann 9:1–9:16

Connected Partitions via Connected Dominating Sets
Aikaterini Niklanovits, Kirill Simonov, Shaily Verma, and Ziena Zeif 10:1–10:14

Courcelle's Theorem for Lipschitz Continuity
Tatsuya Gima, Soh Kumabe, and Yuichi Yoshida 11:1–11:14

33rd Annual European Symposium on Algorithms (ESA 2025).
Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Subtrajectory Clustering and Coverage Maximization in Cubic Time, or Better
Jacobus Conradi and Anne Driemel 12:1–12:18

Tight Bounds for Some Classical Problems Parameterized by Cutwidth
Narek Bojikian, Vera Chekan, and Stefan Kratsch 13:1–13:17

Testing Sumsets Is Hard
Xi Chen, Shivam Nadimpalli, Tim Randolph, Rocco A. Servedio, and Or Zamir .. 14:1–14:16

Linear Layouts Revisited: Stacks, Queues, and Exact Algorithms
Thomas Depian, Simon D. Fink, Robert Ganian, and Vaishali Surianarayanan ... 15:1–15:18

On Algorithmic Applications of $\mathcal{F}$-Branchwidth
Benjamin Bergougnoux, Thekla Hamm, Lars Jaffke, and Paloma T. Lima 16:1–16:17

Weighted Matching in a Poly-Streaming Model
Ahammed Ullah, S M Ferdous, and Alex Pothen 17:1–17:18

Efficiency of Learned Indexes on Genome Spectra
Md. Hasin Abrar, Paul Medvedev, and Giorgio Vinciguerra 18:1–18:18

On Finding ℓ-Th Smallest Perfect Matchings
Nicolas El Maalouly, Sebastian Haslebacher, Adrian Taubner, and Lasse Wulf 19:1–19:15

Beeping Deterministic CONGEST Algorithms in Graphs
Pawel Garncarek, Dariusz R. Kowalski, Shay Kutten, and Miguel A. Mosteiro 20:1–20:17

A Unified FPT Framework for Crossing Number Problems
Éric Colin de Verdière and Petr Hliněný 21:1–21:18

Constructing Long Paths in Graph Streams
Christian Konrad and Chhaya Trehan 22:1–22:19

Computing Largest Subsets of Points Whose Convex Hulls Have Bounded Area and Diameter
Gianmarco Picarella, Marc van Kreveld, Frank Staals, and Sjoerd de Vries 23:1–23:16

Fréchet Distance in Unweighted Planar Graphs
Ivor van der Hoog, Thijs van der Horst, Eva Rotenberg, and Lasse Wulf 24:1–24:16

Instance-Optimal Imprecise Convex Hull
Sarita de Berg, Ivor van der Hoog, Eva Rotenberg, Daniel Rutschmann, and Sampson Wong 25:1–25:15

Faster Dynamic 2-Edge Connectivity in Directed Graphs
Loukas Georgiadis, Konstantinos Giannis, and Giuseppe F. Italiano 26:1–26:16

Online Makespan Scheduling Under Scenarios
Ekin Ergen 27:1–27:16

Sliding Squares in Parallel
Hugo A. Akitaya, Sándor P. Fekete, Peter Kramer, Saba Molaei, Christian Rieck, Frederick Stock, and Tobias Wallner 28:1–28:17

The Tape Reconfiguration Problem and Its Consequences for Dominating Set Reconfiguration
Nicolas Bousquet, Quentin Deschamps, Arnaud Mary, Amer E. Mouawad, and Théo Pierron 29:1–29:15

Reconstructing Random Graphs from Distance Queries
Michael Krivelevich and Maksim Zhukovskii 30:1–30:17

An Optimal Algorithm for Shortest Paths in Unweighted Disk Graphs
Bruce W. Brewer and Haitao Wang 31:1–31:8

Linear-Time Multilevel Graph Partitioning via Edge Sparsification
Lars Gottesbüren, Nikolai Maas, Dominik Rosch, Peter Sanders, and Daniel Seemaier 32:1–32:20

Better Indexing for Rectangular Pattern Matching
Paweł Gawrychowski and Adam Górkiewicz 33:1–33:7

On the Approximability of Train Routing and the Min-Max Disjoint Paths Problem
Umang Bhaskar, Katharina Eickhoff, Lennart Kauther, Jannik Matuschke, Britta Peis, and Laura Vargas Koch 34:1–34:15

The Geodesic Fréchet Distance Between Two Curves Bounding a Simple Polygon
Thijs van der Horst, Marc van Kreveld, Tim Ophelders, and Bettina Speckmann . 35:1–35:16

Efficient Contractions of Dynamic Graphs – With Applications
Monika Henzinger, Evangelos Kosinas, Robin Münk, and Harald Räcke 36:1–36:14

Hardness of Computation of Quantum Invariants on 3-Manifolds with Restricted Topology
Henrique Ennes and Clément Maria 37:1–37:16

Tight Guarantees for Cut-Relative Survivable Network Design via a Decomposition Technique
Nikhil Kumar, J. J. Nan, and Chaitanya Swamy 38:1–38:18

Going Beyond Surfaces in Diameter Approximation
Michał Włodarczyk 39:1–39:19

Maximum List r-Colorable Induced Subgraphs in kP_3-Free Graphs
Esther Galby, Paloma T. Lima, Andrea Munaro, and Amir Nikabadi 40:1–40:13

Min-Max Correlation Clustering via Neighborhood Similarity
Nairen Cao, Steven Roche, and Hsin-Hao Su 41:1–41:18

(Multivariate) k-SUM as Barrier to Succinct Computation
Geri Gokaj, Marvin Künnemann, Sabine Storandt, and Carina Truschel 42:1–42:19

Edge Clique Partition and Cover Beyond Independence
Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov 43:1–43:16

Near-Optimal Vertex Fault-Tolerant Labels for Steiner Connectivity
Koustav Bhanja and Asaf Petruschka 44:1–44:13

Bounded Weighted Edit Distance: Dynamic Algorithms and Matching Lower Bounds
Itai Boneh, Egor Gorbachev, and Tomasz Kociumaka 45:1–45:16

Max-Distance Sparsification for Diversification and Clustering
Soh Kumabe 46:1–46:14

Fast and Lightweight Distributed Suffix Array Construction
Manuel Haag, Florian Kurpicz, Peter Sanders, and Matthias Schimek 47:1–47:18

The Support of Bin Packing Is Exponential
Klaus Jansen, Lis Pirotton, and Malte Tutas 48:1–48:16

Efficient Top-Down Updates in AVL Trees
Vincent Jugé 49:1–49:15

Online Hitting Sets for Disks of Bounded Radii
Minati De, Satyam Singh, and Csaba D. Tóth 50:1–50:16

Quantum Approximate k-Minimum Finding
Minbo Gao, Zhengfeng Ji, and Qisheng Wang 51:1–51:15

Beating Competitive Ratio 4 for Graphic Matroid Secretary
Kiarash Banihashem, MohammadTaghi Hajiaghayi, Dariusz R. Kowalski, Piotr Krysta, Danny Mittal, and Jan Olkowski 52:1–52:16

When Is String Reconstruction Using de Bruijn Graphs Hard?
Ben Bals, Sebastiaan van Krieken, Solon P. Pissis, Leen Stougie, and Hilde Verbeek 53:1–53:16

Non-Boolean OMv: One More Reason to Believe Lower Bounds for Dynamic Problems
Bingbing Hu and Adam Polak 54:1–54:16

Safe Sequences via Dominators in DAGs for Path-Covering Problems
Francisco Sena, Romeo Rizzi, and Alexandru I. Tomescu 55:1–55:17

PLS-Completeness of String Permutations
Dominik Scheder and Johannes Tantow 56:1–56:14

Improved Hardness-Of-Approximation for Token-Swapping
Sam Hiken and Nicole Wein 57:1–57:16

Streaming Diameter of High-Dimensional Points
Magnús M. Halldórsson, Nicolaos Matsakis, and Pavel Veselý 58:1–58:10

Improved Dominance Filtering for Unions and Minkowski Sums of Pareto Sets
Konstantinos Karathanasis, Spyros Kontogiannis, and Christos Zaroliagis 59:1–59:17

Fast and Memory-Efficient BWT Construction of Repetitive Texts Using Lyndon Grammars
Jannik Olbrich 60:1–60:19

ε-Net Algorithm Implementation on Hyperbolic Surfaces
Vincent Despré, Camille Lanuel, Marc Pouget, and Monique Teillaud 61:1–61:18

Polynomial-Time Constant-Approximation for Fair Sum-Of-Radii Clustering
Sina Bagheri Nezhad, Sayan Bandyapadhyay, and Tianzhi Chen 62:1–62:16

Connected $\boldsymbol{k}$-Median with Disjoint and Non-Disjoint Clusters
Jan Eube, Kelin Luo, Dorian Reineccius, Heiko Röglin, and Melanie Schmidt 63:1–63:14

A Dynamic Piecewise-Linear Geometric Index with Worst-Case Guarantees
Emil Toftegaard Gæde, Ivor van der Hoog, Eva Rotenberg, and Tord Stordalen ... 64:1–64:18

From Theory to Practice: Engineering Approximation Algorithms for Dynamic Orientation
Ernestine Grossmann, Henrik Reinstädtler, Eva Rotenberg, Christian Schulz, Ivor van der Hoog, and Juliette Vlieghe 65:1–65:18

Bicriteria Approximation for k-Edge-Connectivity
Zeev Nutov and Reut Cohen 66:1–66:17

Compact Representation of Semilinear and Terrain-Like Graphs
Jean Cardinal and Yelena Yuditsky 67:1–67:19

Faster Algorithm for Second (s,t)-Mincut and Breaking Quadratic Barrier for Dual Edge Sensitivity for (s,t)-Mincut
Surender Baswana, Koustav Bhanja, and Anupam Roy 68:1–68:19

Bandwidth vs BFS Width in Matrix Reordering, Graph Reconstruction, and Graph Drawing
David Eppstein, Michael T. Goodrich, and Songyu (Alfred) Liu 69:1–69:17

Improved Parallel Derandomization via Finite Automata with Applications
Jeff Giliberti and David G. Harris 70:1–70:17

Simpler Universally Optimal Dijkstra
Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann 71:1–71:9

Color Distance Oracles and Snippets: Separation Between Exact and Approximate Solutions
Noam Horowicz and Tsvi Kopelowitz 72:1–72:17

Classical Algorithms for Constant Approximation of the Ground State Energy of Local Hamiltonians
François Le Gall 73:1–73:19

Core-Sparse Monge Matrix Multiplication: Improved Algorithm and Applications
Paweł Gawrychowski, Egor Gorbachev, and Tomasz Kociumaka 74:1–74:17

An Improved Bound for Plane Covering Paths
Hugo A. Akitaya, Greg Aloupis, Ahmad Biniaz, Prosenjit Bose, Jean-Lou De Carufel, Cyril Gavoille, John Iacono, Linda Kleist, Michiel Smid, Diane Souvaine, and Leonidas Theocharous 75:1–75:10

A 3.3904-Competitive Online Algorithm for List Update with Uniform Costs
Mateusz Basiak, Marcin Bienkowski, Martin Böhm, Marek Chrobak, Łukasz Jeż, Jiří Sgall, and Agnieszka Tatarczuk 76:1–76:15

Tolerant Testers for Subgraph-Freeness
Reut Levi and Jonathan Meiri 77:1–77:15

Testing Depth First Search Numbering
Artur Czumaj, Christian Sohler, and Stefan Walzer 78:1–78:16

Semi-Streaming Algorithms for Hypergraph Matching
Henrik Reinstädtler, S M Ferdous, Alex Pothen, Bora Uçar, and Christian Schulz 79:1–79:19

Online Metric TSP
Christian Bertram 80:1–80:9

An $O(n \log n)$ Algorithm for Single-Source Shortest Paths in Disk Graphs
Mark de Berg and Sergio Cabello 81:1–81:15

Buffered Partially-Persistent External-Memory Search Trees
Gerth Stølting Brodal, Casper Moldrup Rysgaard, and Rolf Svenning 82:1–82:18

Fault-Tolerant Matroid Bases
Matthias Bentert, Fedor V. Fomin, Petr A. Golovach, and Laure Morelle 83:1–83:14

Property Testing of Curve Similarity
Peyman Afshani, Maike Buchin, Anne Driemel, Marena Richter, and Sampson Wong 84:1–84:16

A Simple yet Exact Analysis of the MultiQueue
Stefan Walzer and Marvin Williams 85:1–85:14

New Algorithms for Pigeonhole Equal Subset Sum
Ce Jin, Ryan Williams, and Stan Zhang 86:1–86:12

Multicut Problems in Almost-Planar Graphs: the Dependency of Complexity on the Demand Pattern
Florian Hörsch and Dániel Marx 87:1–87:13

Parameterized Approximability for Modular Linear Equations
Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, and Magnus Wahlström 88:1–88:15

A Simple Algorithm for Trimmed Multipoint Evaluation
Nick Fischer, Melvin Kallmayer, and Leo Wennmann 89:1–89:11

Separating Two Points with Obstacles in the Plane: Improved Upper and Lower Bounds
Jack Spalding-Jamieson and Anurag Murty Naredla 90:1–90:18

Near-Optimal Differentially Private Graph Algorithms via the Multidimensional AboveThreshold Mechanism
Laxman Dhulipala, Monika Henzinger, George Z. Li, Quanquan C. Liu, A. R. Sricharan, and Leqi Zhu 91:1–91:20

Incremental Maximization for a Broad Class of Objectives
Yann Disser and David Weckbecker 92:1–92:13

Recognizing and Realizing Temporal Reachability Graphs
Thomas Erlebach, Othon Michail, and Nils Morawietz 93:1–93:18

Faster Algorithm for Bounded Tree Edit Distance in the Low-Distance Regime
Tomasz Kociumaka and Ali Shahali 94:1–94:16

The Planted Orthogonal Vectors Problem
David Kühnemann, Adam Polak, and Alon Rosen 95:1–95:17

Counting Small Induced Subgraphs: Scorpions Are Easy but Not Trivial
Radu Curticapean, Simon Döring, and Daniel Neuen 96:1–96:16

A Fast and Simple Algorithm for the Resource Constrained Shortest Path Problem
Saman Ahmadi, Andrea Raith, and Mahdi Jalili 97:1–97:15

Fine-Grained Classification of Detecting Dominating Patterns
Jonathan Dransfeld, Marvin Künnemann, and Mirza Redzic 98:1–98:15

Engineering Minimal k-Perfect Hash Functions
Stefan Hermann, Sebastian Kirmayer, Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer 99:1–99:18

Cut-Query Algorithms with Few Rounds
Yotam Kenneth-Mordoch and Robert Krauthgamer 100:1–100:14

Improved Algorithms for Quantum MaxCut via Partially Entangled Matchings
Anuj Apte, Eunou Lee, Kunal Marwaha, Ojas Parekh, and James Sud 101:1–101:14

A Combinatorial Proof of Universal Optimality for Computing a Planar Convex Hull
Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann 102:1–102:13

On the Satisfiability of Random 3-SAT Formulas with k-Wise Independent Clauses
Ioannis Caragiannis, Nick Gravin, and Zhile Jiang 103:1–103:17

Optimal Antimatroid Sorting
Benjamin Aram Berendsohn 104:1–104:14

Parameterized Algorithms for Computing Pareto Sets
Joshua Marc Könen, Heiko Röglin, and Tarek Stuck 105:1–105:15

On Estimating the Quantum ℓ_α Distance
Yupan Liu and Qisheng Wang 106:1–106:19

Length-Constrained Directed Expander Decomposition and Length-Constrained Vertex-Capacitated Flow Shortcuts
Bernhard Haeupler, Yaowei Long, Thatchaphol Saranurak, and Shengzhe Wang 107:1–107:17

Deterministic Approximation Algorithm for Graph Burning
Matej Lieskovský 108:1–108:7

Combined Search and Encoding for Seeds, with an Application to Minimal Perfect Hashing
Hans-Peter Lehmann, Peter Sanders, Stefan Walzer, and Jonatan Ziegler 109:1–109:18

Faster Exponential Algorithms for Cut Problems via Geometric Data Structures
László Kozma and Junqi Tan 110:1–110:12

Hardness of Median and Center in the Ulam Metric
Nick Fischer, Elazar Goldenberg, Mursalin Habib, and Karthik C. S. 111:1–111:17

A Faster Parametric Search for the Integral Quickest Transshipment Problem
Mariia Anapolska, Dario van den Boom, Christina Büsing, and Timo Gersing 112:1–112:15

Bootstrapping Dynamic APSP via Sparsification
Rasmus Kyng, Simon Meierhans, and Gernot Zöcklein 113:1–113:15

A Deterministic Partition Tree and Applications
Haitao Wang 114:1–114:17

Smoothed Analysis of Online Metric Problems
Christian Coester and Jack Umenberger 115:1–115:14

Fast Gaussian Elimination for Low Treewidth Matrices
Martin Fürer, Carlos Hoppen, and Vilmar Trevisan 116:1–116:15

Preface

This volume contains the extended abstracts selected for presentation at ESA 2025, the 33rd European Symposium on Algorithms. The event was organized by the University of Warsaw, Poland, as a part of ALGO 2025, on September 15–17, 2025.

The scope of ESA includes original, high-quality, theoretical and applied research on algorithms and data structures. Since 2002, it has had two tracks: the Design and Analysis Track (Track A), for papers with a strong emphasis on the theoretical analysis of algorithms, and the Engineering and Applications Track (Track B), for papers reporting on the results of extensive experimental evaluations and/or providing original contributions to the engineering of algorithms for practical applications. In 2022, a Simplicity Track (Track S) was added, inviting submissions that prove or explain results, possibly already known, in a particularly clear, simple or elegant way.

In response to the call for papers for ESA 2025, 372 papers were submitted, 261 for Track A, 54 for Track B, and 57 for Track S (these numbers exclude 5 submissions that were later withdrawn). Paper selection was based on originality, technical quality, exposition quality, and relevance. Each paper received at least three reviews (apart from two exceptional cases).

Due to a substantial increase (+11% compared to last year) in the number of submissions and the strong quality of submitted results, the program committees selected 115 papers for inclusion in the program (also +11%): 83 from Track A, 16 from Track B, and 16 from Track S, yielding an overall acceptance rate of about 31%. The presentations of the accepted papers, together with two invited talks by Bernhard Haeupler (INSAIT, University of Sofia "St Kliment Ohridski" and ETH Zurich) and Monika Henzinger (Institute of Science and Technology Austria) promise to make up an exciting program.

The European Association for Theoretical Computer Science (EATCS) sponsored best paper awards. For Track A, the best paper award was given to Hans-Peter Lehmann, Peter Sanders, Stefan Walzer and Jonatan Ziegler for the paper *Combined Search and Encoding for Seeds, with an Application to Minimal Perfect Hashing*. For Track B, the best paper award was given to Ahammed Ullah, SM Ferdous and Alex Pothen for the paper *Weighted Matching in a Poly-Streaming Model*. For Track S, the best paper award was given to Ivor van der Hoog, Eva Rotenberg and Daniel Rutschmann for the paper *Simpler Universally Optimal Dijkstra*.

As a typical convention in the field is to list authors in alphabetical order, we randomized the order of papers in these proceedings to avoid biases.

We wish to thank all the authors who submitted papers for consideration, the invited speakers, the members of the program committees for their hard work, and the (exactly!) 600 external reviewers who assisted the program committees in the evaluation process. Special thanks go to the organizing committee, who helped us with the organization of the conference.

Information on past ESA symposia, including locations and proceedings, is maintained at `https://algo-conference.org/`.

33rd Annual European Symposium on Algorithms (ESA 2025).
Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committees

Track A (Design and Analysis) Program Committee

- Hee-Kap Ahn, Pohang University of Science and Technology (POSTECH), Korea
- Shyan Akmal, Institute for Computer Science, Artificial Intelligence and Technology (INSAIT) Sofia, Bulgaria
- Boris Aronov, NYU
- Alkida Balliu, Gran Sasso Science Institute (GSSI), L'Aquila, Italy
- Hideo Bannai, Institute of Science Tokyo
- Soheil Behnezhad, Northeastern University
- Omri Ben-Eliezer, Technion
- Ioana O. Bercea, KTH
- Ahmad Biniaz, University of Windsor, Canada
- Eric Blais, University of Waterloo
- Édouard Bonnet, CNRS, LIP, ENS Lyon
- Karl Bringmann, Saarland University, MPI
- Kevin Buchin, Technical University Dortmund
- Amit Chakrabarti, Dartmouth
- Chandra Chekuri, University of Illinois, Urbana-Champaign
- Eden Chlamtac, Ben Gurion University
- Yann Disser, Technische Universität Darmstadt
- Dani Dorfman, MPI
- Shaddin Dughmi, University of Southern California
- Marek Elias, Bocconi University in Milan
- Esther Ezra, Bar Ilan University
- Paolo Ferragina, University of Pisa
- Aris Filos-Ratsikas, University of Edinburgh
- Pierre Fraigniaud, CNRS and Université Paris Cité
- Pinar Heggernes, University of Bergen
- David Ellis Hershkowitz, Brown University
- Zhiyi Huang, The University of Hong Kong
- Arun Jambulapati, University of Michigan
- Rajesh Jayaram, Google
- Haim Kaplan, Tel Aviv University (Chair)
- Thomas Kesselheim, University of Bonn
- Linda Kleist, Universtität Potsdam
- Jakub Łącki, Google
- Stefano Leonardi, Sapienza University of Rome
- Roie Levin, Rutgers University
- Daniel Lokshtanov, University of California Santa Barbara
- Pasin Manurangsi, Google
- Dániel Marx, CISPA Helmholtz Center for Information Security
- Benjamin Moseley, Carnegie Mellon University
- Sagnik Mukhopadhyay, University of Birmingham
- Cameron Musco, University of Massachusetts, Amherst
- Aaron Potechin, University of Chicago

33rd Annual European Symposium on Algorithms (ESA 2025).
Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Manish Purohit, Google
- Chris Schwiegelshohn, Aarhus University
- Cliff Stein, Columbia University
- Ioan Todinca, Univ. Orléans
- Jie Xue, NYU Shanghai
- Sorrachai Yingchareonthawornchai, ETH Zurich

Track B (Engineering and Applications) Program Committee

- Anne Benoit, ENS Lyon (Chair)
- Petra Berenbrink, University of Hamburg, Germany
- Florina Ciorba, University of Basel, Switzerland
- Pål Grønås Drange, University of Bergen, Norway
- SM Ferdous, Pacific Northwest National Lab, USA
- Li Han, ECNU, China
- Valentin Honoré, ENSIIE, France
- Kamer Kaya, Sabancı University, Turkey
- Alba Cristina Magalhaes Alves de Melo, University of Brasilia, Brazil
- Henning Meyerhenke, Karlsruhe Institute of Technology, Germany
- Valia Mitsou, IRIF, France
- Manuel Penschuck, Goethe University Frankfurt, Germany
- Cynthia Phillips, Sandia National Laboratories, USA
- Krzysztof Rządca, University of Warsaw and Google, Poland
- Blair D. Sullivan, University of Utah, USA
- Hongyang Sun, University of Kansas, USA
- Sivan Toledo, Tel-Aviv University, Israel
- Jesper Larsson Träff, TU Vienna, Austria
- Frédéric Vivien, Inria, France
- Ulrike Meier Yang, LLNL, USA

Track S (Simplicity) Program Committee

- Peyman Afshani, Aarhus University, Denmark
- Martin Aumüller, ITU Copenhagen, Denmark
- Parinya Chalmersook, University of Sheffield, UK
- Rathish Das, University of Houston, USA
- David Eppstein, University of California, Irvine, USA
- Rolf Fagerberg, University of Southern Denmark, Denmark
- José Fuentes-Sepúlveda, University of Concepción, Chile
- Christian Ikenmeyer, University of Warwick, UK
- Akitoshi Kawamura, Kyoto University, Japan
- Phillip Keldenich, TU Braunschweig, Germany
- Tsvi Kopelowitz, Bar Ilan University, Israel
- Pascal Lenzner, University of Augsburg, Germany
- Namrata, University of Liverpool, UK
- Will Rosenbaum, University of Liverpool, UK
- Sharma Thankachan, NC State University, USA
- Sebastian Wild, University of Marburg and University of Liverpool (Chair)

List of External Reviewers

Aaron Bernstein
Aaron Putterman
Aaron Schild
Abhishek Dhawan
Adam Karczmarz
Adam Polak
Aditya Anand
Adrian Vladu
Ahammed Ullah
Ajaykrishnan E S
Akash Kumar
Alberto Marchetti-Spaccamela
Aleksander Łukasiewicz
Alessandro Berti
Alex Conway
Alex Crane
Alexander Leonhardt
Alexander Lindermayr
Alexander Munteanu
Alexandra Lassota
Alexandros Hollender
Alexandros Voudouris
Ali Vakilian
Alireza Amanihamedani
Alma Ghafari
Ameet Gadekar
Amer Mouawad
Amir Abboud
Amir Azarmehr
Amit Kumar
Anannya Upasana
Anders Aamand
André Nusser
Andreas Abels
Andrew McGregor
Aniket Basu Roy
Anil Maheshwari
Anish Mukherjee
Anita Dürr
Anna Heuser
Annette Lutz
Annika Hennes
Anthony Perez
Antonios Antoniadis
Antonis Skarlatos
Antti Roeyskoe
Arad Kotzer
Archontia Giannopoulou
Arghya Bhattacharya
Argyrios Deligkas
Ariel Sapir
Arjan Cornelissen
Arnab Ganguly
Arnab Maiti
Arseny Shur
Arthur Braida
Artur Czumaj
Arturo Merino
Aryan Agarwala
Ashwin Jacob
Asif Khan
Athanasios Konstantinidis
Austin Buchanan
Avivit Levy
Baris Can Esmer
Bart de Keijzer
Ben Plosk
Benjamin Aram Berendsohn
Benjamin Bergougnoux
Benjamin Doerr
Bento Natura
Bernhard Haeupler
Bertrand Simon
Bhargav Samineni
Bodo Manthey
Bundit Laekhanukit
Byeonguk Kang
C. Seshadhri
Cameron Seth
Carlo Tosoni
Carolin Rehs
Caroline Brosse
Casper Rysgaard
Ce Jin
Chaeyoon Chung
Changjun Wang
Chao Xu
Charalampos Kokkalis

33rd Annual European Symposium on Algorithms (ESA 2025).
Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Charis Papadopoulos
Cheng Chen
Chien-Chung Huang
Chin Ho Lee
Christian Coester
Christian Komusiewicz
Christian Konrad
Christian Nöbel
Christoph Dürr
Christopher Hahn
Conrado Martínez
Cornelius Brand
Da Wei Zheng
Dana Shapira
Daniel Anderson
Daniel Gibney
Daniel Hathcock
Daniele Carnevale
David Gleich
David Naori
David Saulpic
David Soloveichik
David Wajc
Davide Bilò
Davide Cenzato
Dean Doron
Debajyoti Kar
Debarati Das
Deeksha Adil
Dekel Tsur
Dimitrios Los
Dina Sokol
Dingding Dong
Dipan Dey
Diptapriyo Majumdar
Diptarka Chakraborty
Dirk Sudholt
Dominik Kempa
Dominik Köppl
Dominik Krupke
Dominik Scheder
Dominik Schweisgut
Dongchen Li
Dorian Guyot
Dušan Knop
Dvir Fried
Eduard Eiben
Edward Pyne
Elad Roda
Elazar Goldenberg
Elisabeth Gaar
Éric Colin de Verdière
Eric Samperton
Éric Schost
Eric Sedgwick
Erik Vee
Eryk Kopczynski
Esther Galby
Ethan Epperly
Ethan Leeman
Euiwoong Lee
Eunjin Oh
Ewin Tang
Eyal Ackerman
Fabian Brandt-Tumescheit
Fabrizio Montecchiani
Fahad Panolan
Fawaz Dabbaghie
Federico Fusco
Fei Xu
Felipe A. Louza
Felix Hommelsheim
Felix Reidl
Fernando Granha Jeronimo
Florent Capelli
Florent Foucaud
Florian Hörsch
Florian Kurpicz
Francesco Silvestri
Franz Aurenhammer
Franziska Eberle
Frédéric Mazoit
Gabriel Siqueira
Gang Liu
Gary Hoppenworth
Georg Loho
George Skretas
Giannos Stamoulis
Gill Barequet
Giulio Ermanno Pibiri
Giuseppe Romana
Gokarna Sharma
Gramoz Goranci
Greg Bodwin
Greg Kuperberg
Gregory Kucherov

Gregory Sorkin
Guanghao Ye
Guillaume Marçais
Gur Lifshitz
Haiqin Wu
Hamed Hosseinpour
Hamid Zarrabi-Zadeh
Hang Liao
Hannaneh Akrami
Hans Bodlaender
Hanwen Zhang
Haotian Jiang
Harald Räcke
Hengjia Wei
Henri Riihimäki
Hoai-An Nguyen
Hoffmann Michael
Holger Dell
Hsien-Chih Chang
Huairui Chu
Huanjian Zhou
Hugo Jacob
Hung Hoang
Hung Le
Hyojeong An
Ilan Cohen
Ildikó Schlotter
Ioannis Caragiannis
Irene Parada
Irina Kostitsyna
Isaac Hair
Itai Boneh
Itai Dinur
Ivor van der Hoog
Iyad Kanj
Jacob Gilbert
Jaegun Lee
Jaehoon Chung
Jakob Moosbauer
Jakob Nogler
Jakub Radoszewski
Jan van den Brand
Janani Sundaresan
Jannik Matuschke
Javier Cembrano
Jean-Pierre von der Heydt
Jeff Phillips
Jens Schlöter
Jeongwon Moon
Jérémy Barbay
Jeremy Fineman
Jesper Jansson
Jesper Nederlof
Jesse Lerner
Jiahao Zhao
Jiaheng Wang
Jiale Chen
Jianwen Li
Jim Apple
Jing Li
Jingbang Chen
Joachim Spoerhase
Joe Sawada
Johanna Luz
Johannes Brustle
John Iacono
Jonas Kamminga
Jonas Schmidt
Jonathan Conroy
Jonathan Klawitter
José A. Soto
José Verschae
Joseph Koutsoutis
Joseph Mitchell
Josh Alman
Joshua Wang
Jueun Seo
Julien Baste
Julien Duron
Justin Dallant
Karnati Venkata Naga Sreenivasulu
Karolina Okrasa
Karthekeyan Chandrasekaran
Katrin Casel
Ke Chen
Kenneth Langedal
Kevin Schewior
Kien Huynh
Kim-Manuel Klein
Kirill Simonov
Kishen N. Gowda
Konstantinos Tsakalidis
Kostas Kollias
Kristin Knorr
Kristin Sheridan
Krisztina Szilagyi

Kunal Marwaha
Kyrian Adimora
Lars Jaffke
Lars Rohwedder
László Kozma
Laura Sanità
Laurent Viennot
Laxman Dhulipala
Leah Epstein
Lee-Ad Gottlieb
Leonidas Theocharous
Leszek Gasieniec
Liam Roditty
Lilian Salinas
Lily Chung
Lin Chen
Longkun Guo
Loukas Georgiadis
Lucas Perotin
Lukas Berner
Lukas Drexler
Lukas Geis
Lukas Hintze
Łukasz Jeż
Lynn Engelberts
Maël Dumas
Maike Buchin
Majid Mirzanezhad
Malin Rau
Mano Prakash Parthasarathi
Manoj Gupta
Manuel Cáceres
Marc Roth
Marcin Bienkowski
Marek Sokołowski
Marena Richter
Marianne Akian
Marin Bougeret
Marinella Sciortino
Marta Pozzi
Marten Maack
Martin Böhm
Martin Herold
Martin Koutecký
Martins Kokainis
Matan Kraus
Mathieu Liedloff
Matias Korman
Matteo Russo
Matthew Casey
Matthew Fahrbach
Matthias Bentert
Matthias Mnich
Mattia Odorisio
Maurice Rolvien
Max Bannach
Max Klimm
Maximilian J. Stahlberg
Mayank Goswami
Meike Neuwohner
Melanie Schmidt
Micha Sharir
Michael Bekos
Michael Dinitz
Michael Elkin
Michael Lampis
Michael Zlatin
Michał Pawłowski
Michał Włodarczyk
Michelle Döring
Mikaël Rabie
Mingxian Zhong
Minju Song
Miriam Goetze
Mirko Giacchini
Mohammad Ali Abam
Mohammad Roghani
Mohammad Saneian
Nadia Pisanti
Naonori Kakimura
Naoto Ohsaka
Narek Bojikian
Nastaran Behrooznia
Nate Veldt
Nathan Klein
Nathan Wallheimer
Naveen Garg
Neel Patel
Neeldhara Misra
Neeraj Kumar
Neil Olver
Neng Huang
Neta Singer
Nick Fischer
Nicola Prezza
Nicolas Bousquet

Nicolas Klodt
Nicolás Sanhueza-Matamala
Nicole Megow
Nikhil Kumar
Nikhil Mande
Niklas Dahlmeier
Niklas Schlomberg
Nils Morawietz
Niranka Banerjee
Noah Fleming
Nodari Sitchinava
Noriyoshi Sukegawa
O-Joung Kwon
Ofer Neiman
Ondřej Suchý
Or Vardi
Or Zamir
Palash Dey
Paloma de Lima
Pan Peng
Panagiotis Charalampopoulos
Patrick Schnider
Patrick Totzke
Pattara Sukprasert
Paul Bastide
Paul Hovland
Pawel Gawrychowski
Pedro Montealegre
Peilin Zhong
Peter Davies
Peter Kiss
Peter Kramer
Peter Sanders
Philipp Kindermann
Pierre Bergé
Pieter Kleer
Pingan Cheng
Pooria Namyar
Pranabendu Misra
Praneeth Kacham
Prantar Ghosh
Prasanna Ramakrishnan
Qingyun Chen
Rachael Colley
Radu Curticapean
Raghavendra Addanki
Ragnar Groot Koerkamp
Rahul Shah
Rajesh Chitnis
Ramanujan M. Sridharan
Ramin Kosfeld
Ran Duan
Rares-Darius Buhai
Rasmus Pagh
Rebecca Lehming
Redouane Elghazi
Renato Ferreira Pinto Junior
Rene Sitters
Reto Krummenacher
Riccardo Dondi
Richard Hladík
Richard Huang
Rico Zenklusen
Riko Jacob
Rob van Stee
Rodrigo Silveira
Rohit Vaish
Romain Cosson
Ron Safier
Roohani Sharma
Ruben Hoeksma
Rudy Zhou
Ruizhe Zhang
Runtian Ren
Ruoxu Cen
Ryuhei Mizutani
Sabine Storandt
Saket Saurabh
Saladi Rahul
Sally Dong
Sam Buss
Samin Jamalabadi
Sampson Wong
Samson Zhou
Samuel McCauley
Sander Borst
Sándor Fekete
Sarita de Berg
Sathyawageeswar Subramanian
Sebastian Berndt
Sebastian Forster
Sebastian Siebertz
Sebastian Zur
Seeun William Umboh
Seongmin Kwon
Sepehr Assadi

Sergio Cabello
Seyoon Ragavan
Shahar Lewkowicz
Shang-En Huang
Shaofeng H.-C. Jiang
Shaohua Li
Sharmila Duppala
Shay Golan
Sherry Sarkar
Shin-Ichi Tanigawa
Shirley Zhang
Shmuel Tomi Klein
Shreyas Srinivas
Shubhang Kulkarni
Shunhua Jiang
Siddharth Gollapudi
Siddharth Sheth
Simon D. Fink
Simon Martiel
Simone Di Gregorio
Simone Fioravanti
Siyue Liu
Soeren Terziadis
Sophie Spirkl
Stavros Kolliopoulos
Stefan Kratsch
Stefan Neumann
Stéphane Bessy
Stéphane Gaubert
Subhasree Patro
Sujoy Bhore
Sung-Hwan Kim
Sungjin Im
Suprovat Ghoshal
Sven Jäger
Svenja M. Griesbach
Svetlana Kulagina
Ta-Wei Tu
Taekang Eom
Takaaki Nishimoto
Takeru Utsumi
Takuya Mieno
Tarun Kathuria
Tasuku Soma
Telikepalli Kavitha
Thekla Hamm
Thiago Bergamaschi
Thomas C. Van Dijk
Thomas Erlebach
Thomas Jakobsche
Thomas Lidbetter
Thomas Schibler
Thorsten Götte
Till Fluschnik
Tim Gerlach
Tim Ophelders
Tim Randolph
Timothée Corsini
Timothy M. Chan
Tom Mccormick
Tom Peters
Tomasz Kanas
Tomasz Kociumaka
Tomasz Ponitka
Tomasz Walen
Tomohiro I
Tomohiro Koana
Tuukka Korhonen
Tzvika Geft
Uwe Schwiegelshohn
Václav Blažej
Vaggos Chatziafratis
Valentin Polishchuk
Vasileios Nakos
Vihan Shah
Viktoriia Korchemna
Vincent Cohen-Addad
Vishwa Prakash H V
Vivien Vandaele
Waldo Gálvez
Willem Fletcher
William Evans
William Kuszmaul
William Lochet
Wojciech Czerwiński
Wojciech Janczewski
Wolfgang Mulzer
Xiaoyang Xu
Xin Han
Xin Lyu
Xujin Chen
Yael Kirkpatrick
Yanheng Wang
Yaniv Sadeh
Yasuaki Kobayashi
Yasushi Kawase

Yican Sun
Yiming Zhao
Yinzhan Xu
Yixin Cao
Yonggang Jiang
Yongjie Yang
Yoshifumi Sakai
Yossi Azar
Yota Otachi
Youjung Bae
Yuanyuan Yang
Yuchen Mao
Yuhao Zhang
Yuichi Yoshida
Yukiko Yamauchi
Yurong Chen
Yusuf Kalayci
Yusuke Kobayashi
Yutaro Yamaguchi
Yuto Nakashima
Zackary Jorquera
Zhengcheng Huang
Zhongtian He
Ziad Ismaili Alaoui
Zihan Tan
Zihang Wu
Zixuan Zhu
Zoya Svitkina

Graph Decompositions and Length-Constrained Expanders

Bernhard Haeupler
INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria
ETH Zürich, Switzerland

Abstract

Graph decompositions are powerful algorithmic tools with wide applications to graph structures (e.g., spanners, hopsets, sparsifiers, oblivious routings, etc.) and network optimization algorithms, including parallel, distributed and dynamic algorithms for flow and distance problems.

Classical graph decompositions include

- low-diameter decomposition, which captures ℓ_1-quantities like lengths and costs, and
- expander decomposition, which captures ℓ_∞-quantities like flows and congestion.

This keynote starts with a brief survey of these classical decompositions, then presents length-constrained expanders and length-constrained expander decompositions – a recent and technically rich generalization that simultaneously controls length and congestion (ℓ_1 & ℓ_∞). Length-constrained expander decompositions significantly broaden and extend the range of applications for graph decompositions, and this talk will discuss several examples and ways to leverage their power.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Length-Constrained Expanders, Graph Decomposition, Network Optimization Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.1

Category Invited Talk

Funding Partially funded by the Ministry of Education and Science of Bulgaria's support for INSAIT as part of the Bulgarian National Roadmap for Research Infrastructure and through the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ERC grant agreement 949272).

References

1. Greg Bodwin, Bernhard Haeupler, and Merav Parter. Fault-tolerant spanners against bounded-degree edge failures: Linearly more faults, almost for free. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2609–2642. SIAM, 2024. doi:10.1137/1.9781611977912.93.
2. Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic. Hop-constrained oblivious routing. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 1208–1220, 2021. doi:10.1145/3406325.3451098.
3. Bernhard Haeupler, D Ellis Hershkowitz, Jason Li, Antti Roeyskoe, and Thatchaphol Saranurak. Low-step multi-commodity flow emulators. *Annual ACM Symposium on Theory of Computing (STOC)*, pages 71–82, 2024. doi:10.1145/3618260.3649689.
4. Bernhard Haeupler, D Ellis Hershkowitz, and Thatchaphol Saranurak. Maximum length-constrained flows and disjoint paths: Distributed, deterministic and fast. In *Annual ACM Symposium on Theory of Computing (STOC)*, 2023. doi:10.1145/3564246.3585202.
5. Bernhard Haeupler, D Ellis Hershkowitz, and Zihan Tan. Parallel greedy spanners. *arXiv*, 2023. arXiv:2304.08892.
6. Bernhard Haeupler, D Ellis Hershkowitz, and Zihan Tan. New structures and algorithms for length-constrained expander decompositions. *Symposium on Foundations of Computer Science (FOCS)*, pages 1634–1645, 2024. doi:10.1109/FOCS61266.2024.00102.


© Bernhard Haeupler;
licensed under Creative Commons License CC-BY 4.0



7 Bernhard Haeupler, Jonas Huebotter, and Mohsen Ghaffari. A cut-matching game for constant-hop expanders. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1651–1678. SIAM, 2025. doi:10.1137/1.9781611978322.51.

8 Bernhard Haeupler, Yonggang Jiang, Yaowei Long, Thatchaphol Saranurak, and Shengzhe Wang. Parallel $(1+\epsilon)$-approximate multi-commodity mincost flow in almost optimal depth and work. *Symposium on Foundations of Computer Science (FOCS)*, 2025.

9 Bernhard Haeupler, Yaowei Long, and Thatchaphol Saranurak. Dynamic deterministic constant-approximate distance oracles with n^{ϵ} worst-case update time. *Symposium on Foundations of Computer Science (FOCS)*, pages 2033–2044, 2024. doi:10.1109/FOCS61266.2024.00121.

10 Bernhard Haeupler, Yaowei Long, Thatchaphol Saranurak, and Shengzhe Wang. Length-constrained directed expander decomposition and length-constrained vertex-capacitated flow shortcuts. *Annual European Symposium on Algorithms (ESA)*, 2025. doi:10.4230/LIPIcs.ESA.2025.108.

11 Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 1325–1338, 2022. doi:10.1145/3519935.3520026.

12 Bernhard Haeupler, David Wajc, and Goran Zuzic. Network coding gaps for completion times of multiple unicasts. In *Symposium on Foundations of Computer Science (FOCS)*, pages 494–505. IEEE, 2020. doi:10.1109/FOCS46700.2020.00053.

Securing Dynamic Data: A Primer on Differentially Private Data Structures

Monika Henzinger
Institute of Science and Technology, Klosterneuburg, Austria

Roodabeh Safavi
Institute of Science and Technology, Klosterneuburg, Austria

Abstract

We give an introduction into differential privacy in the dynamic setting, called the continual observation setting.

2012 ACM Subject Classification Security and privacy; Theory of computation

Keywords and phrases Differential privacy, continual observation

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.2

Category Invited Talk

1 Introduction

In 2006, Netflix released an anonymized dataset of movie ratings for a public competition. Although names were removed, it was later shown that users could be re-identified by linking the data with publicly available IMDb ratings [39]. This so-called (privacy) attack demonstrated the failure of traditional anonymization techniques, especially when different anonymized datasets are combined. It also highlighted the need for stronger privacy definitions that provide rigorous guarantees even when multiple analyses (or synthetic datasets) based on the same individual's secret data are published.

This motivated the definition of differential privacy (DP), a privacy requirement for mechanisms introduced by Dwork, McSherry, Nissim, and Smith [17]. A differentially private mechanism[1] adds randomness to the output in such a way that, for any two "neighboring" datasets, the resulting output distributions are "close". Formally, the notion of neighboring datasets is defined as a binary relation on the family of datasets, and closeness of output distributions is defined by the following definition of indistinguishability:

▶ **Definition 1** ((ϵ, δ)-indistinguishability). *For $\epsilon \geq 0$ and $0 \leq \delta \leq 1$, two random variables X_0 and X_1 over a domain $\mathcal{Y}$ are (ϵ, δ)-indistinguishable if, for every subset $Y \subseteq \mathcal{Y}$,*

$$Pr[X_0 \in Y] \leq e^{\epsilon} Pr[X_1 \in Y] + \delta.$$

X_0 and X_1 are ϵ-indistinguishable if $\delta = 0$.

Two datasets are typically considered to be neighboring if they differ by the presence or absence of a single data record. With this definition of neighboring, after observing the output of a differentially private mechanism, an attacker cannot confidently determine

[1] In theory, a common way to achieve differential privacy is by adding noise drawn from continuous distributions, such as the Gaussian distribution, when computing data analysis. However, exact sampling from continuous distributions is computationally infeasible. For this reason, the differential privacy literature typically avoids the term "differentially private algorithm" and uses the term "mechanism" instead. In practice, discrete approximations of these continuous distributions – implemented with finite precision – are used instead.


© Monika Henzinger and Roodabeh Safavi;
licensed under Creative Commons License CC-BY 4.0



whether any specific data record was included in the input dataset. Let $\mathcal{X}$ denote the family of data records, and $\mathrm{MS}(\mathcal{X})$ denote the set of all multisets over $\mathcal{X}$. A differentially private mechanism is formally defined as follows:

▶ **Definition 2** ((ϵ, δ)-Differential Privacy [17]). *For $\epsilon \geq 0$ and $0 \leq \delta \leq 1$, a randomized mechanism $\mathcal{M} : MS(\mathcal{X}) \to \mathcal{Y}$ is (ϵ, δ)-differentially private (or (ϵ, δ)-DP for short) if, for every two neighboring datasets $D_0, D_1 \in MS(\mathcal{X})$, the random variables $\mathcal{M}(D_0)$ and $\mathcal{M}(D_1)$ are (ϵ, δ)-indistinguishable. $\mathcal{M}$ is ϵ-differentially private (or ϵ-DP for short) if $\delta = 0$.*

Note that there exist problems that can be solved efficiently and with high accuracy in the non-private setting, but for which no differentially private solution exists. The interior point problem is an instance of this: Suppose the data universe $\mathcal{X}$ is totally ordered and datasets are multisets of size $n \in N$ over $\mathcal{X}$. An interior point mechanism takes a dataset D as input and its goal is to return an element $y \in \mathcal{X}$ such that y is an interior point in D, i.e., $\min_{x \in D} x \leq y \leq \max_{x \in D} x$. An interior point mechanism is said to be accurate if for every input dataset, it returns an interior point of the dataset with probability at least 3/4. Differential privacy for this mechanism is defined with respect to neighboring datasets that differ in exactly one element. Bun, Nissim, Stemmer, and Vadhan [6] showed that for any $0 < \epsilon < 1/4$ and $\delta < 1/(50n^2)$, there exists no accurate interior point mechanism that satisfies (ϵ, δ)-DP.

Fundamental properties of differential privacy. Differential privacy is carefully designed to satisfy the following two crucial properties: (i) It is preserved under post-processing, and (ii) executing multiple differentially private mechanisms on the same private dataset remains differentially private, with the privacy parameters determined by those of the executed mechanisms. The first property ensures that any computation applied to the output of a differentially private mechanism does not incur additional privacy loss. For example, a privatized statistic released by a hospital can be safely reused in multiple research studies to derive new results, without further compromising the privacy of the patients. The second property, known as *composition*, provides a bound on the total privacy loss when an individual's data is used in multiple differentially private analyses. The goal of this property is to prevent attacks similar to the Netflix attack discussed before.

In *post-processing*, a subsequent mechanism that does *not* have access to the sensitive dataset uses the output of a DP mechanism as input. The so-called *post-processing lemma* states that this does not affect the differential privacy properties; informally speaking, no more privacy is lost.

▶ **Lemma 3** (Post-Processing [20]). *Let $\mathcal{M} : MS(\mathcal{X}) \to \mathcal{Y}$ be an (ϵ, δ)-DP mechanism, and $f : \mathcal{Y} \to \mathcal{Z}$ be a function. Then, the mechanism $f(\mathcal{M})$, which applies $\mathcal{M}$ to the input and then applies f to its output, is also (ϵ, δ)-DP.*

A different situation arises if the second mechanism has access to the sensitive data as well. In this case, for every output of the first mechanism, the second mechanism must also satisfy differentially private. The following two so-called *composition theorems* analyze the total privacy loss for the combination of the output of multiple mechanisms.

▶ **Theorem 4** (Basic Composition [20]). *Let $\mathcal{M}_1 : MS(\mathcal{X}) \to \mathcal{Y}_1$ and $\mathcal{M}_2 : MS(\mathcal{X}) \to \mathcal{Y}_2$ be mechanisms that satisfy (ϵ_1, δ_1)-DP and (ϵ_2, δ_2)-DP, respectively. Then, the mechanism $\mathcal{M}$, defined as $\mathcal{M}(D) = (\mathcal{M}_1(D), \mathcal{M}_2(D))$ for each $D \in MS(\mathcal{X})$, is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-DP.*

By applying basic composition inductively, we obtain that the composition of $k \in \mathbb{N}$ mechanisms, each satisfying (ϵ_0, δ_0)-DP, ensures $(k\epsilon_0, k\delta_0)$-DP. The next theorem, known as *advanced composition theorem*, shows that when ϵ_0 is small, composition also satisfies $(\sqrt{k}\sqrt{\log(1/\delta)}\epsilon_0, \delta + k\delta_0)$-DP for every $\delta \in (0, 1]$, where the first privacy parameter scales with $\sqrt{k}$ rather than k. However, unlike basic composition, advanced composition introduces a nonzero δ even if each individual mechanism is purely differentially private (i.e., $\delta_0 = 0$). This additive term can be interpreted as a small probability of privacy failure. Thus, in applications where any such failure is unacceptable and composition must satisfy pure differential privacy, basic composition must be applied.

▶ **Theorem 5** (Advanced Composition [21]). *Let $k \in \mathbb{N}$, and for each $i \in [k]$, let $\mathcal{M}_i : MS(\mathcal{X}) \to \mathcal{Y}_i$ be an (ϵ_i, δ_i)-DP mechanism. Then, for every $0 \le \delta \le 1$, the mechanism $\mathcal{M}$, defined as $\mathcal{M}(D) = (\mathcal{M}_1(D), \ldots, \mathcal{M}_k(D))$ for each $D \in MS(\mathcal{X})$, is (ϵ, δ)-DP, where*

$$\epsilon = \sum_{i=1}^{k} \epsilon_i \cdot (e^{\epsilon_i} - 1) + \sqrt{2\log(\frac{1}{\delta'}) \sum_{i=1}^{k} \epsilon_i^2} \qquad \text{and} \qquad \delta = \delta' + \sum_{i=1}^{k} \delta_i.$$

The next composition theorem guarantees differential privacy when multiple differentially private analyses are performed on disjoint subsets of a dataset. In this composition, known as the *parallel composition*, the overall privacy loss is determined by the *maximum* privacy loss of the individual mechanisms, rather than the sum. For example, consider multiple elementary schools, each with a distinct set of students. Suppose each school applies an (ϵ, δ)-differentially private mechanism to analyze its own student data. If the government later aggregates the privatized outputs from all schools for policy decisions, the overall process remains (ϵ, δ)-differentially private.

▶ **Definition 6** (Parallel Composition [38]). *Let $k \in \mathbb{N}$, and for each $i \in [k]$, let $\mathcal{M}_i : MS(\mathcal{X}) \to \mathcal{Y}_i$ be an (ϵ_i, δ_i)-DP mechanism with respect to a neighbor relation $\sim_i$. Define $\mathcal{M} : MS(\mathcal{X})^k \to \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_k$ as $\mathcal{M}(D) = (\mathcal{M}_1(D_1), \ldots, \mathcal{M}_k(D_k))$ for each $D = (D_1, \ldots, D_k) \in MS(\mathcal{X})^k$. Two sequences of datasets $D = (D_1, \ldots, D_k) \in MS(\mathcal{X})^k$ and $D' = (D'_1, \ldots, D'_k) \in MS(\mathcal{X})^k$ are considered to be neighboring if and only if there exists $i \in [k]$ such that $D_i \sim_i D'_i$ and $D_j = D'_j$ for all $j \neq i$. Then, the mechanism $\mathcal{M}$ satisfies (ϵ, δ)-DP, where $\epsilon = \max_{1 \le i \le k} \epsilon_i$ and $\delta = \max_{1 \le i \le k} \delta_i$.*

The Laplace Mechanism. One of the most fundamental differentially private mechanisms is the *Laplace Mechanism*. This mechanism computes a "simple" statistic f on the input dataset, adds Laplace noise to it, and outputs the noisy result. Here, "simple" refers to functions with low *sensitivity*, where the sensitivity of a function measures how much its output can differ over neighboring datasets:

▶ **Definition 7** (ℓ_k-Sensitivity). *Let $f : MS(\mathcal{X}) \to \mathbb{R}^d$ be a function mapping a dataset to a real number. For each $k \in \mathbb{N}$, the ℓ_k-*sensitivity of f, *denoted $\Delta_{k,f}$, is defined as*

$$\Delta_{k,f} = \max_{D, D' \in MS(\mathcal{X})} \|f(D) - f(D')\|_k ,$$

where $\|\cdot\|_k$ is the k-norm.

▶ **Definition 8** (Laplace Distribution). *The Laplace distribution with mean 0 and scale parameter b has the probability density function*

$$f_{Lap(b)}(x) = \frac{1}{2b} \exp\left(\frac{-|x|}{b}\right), \quad x \in \mathbb{R}.$$

A random variable following this distribution is denoted by $Y \sim Lap(b)$ or simply $Lap(b)$.

The following Lemma formally defines the Laplace mechanism and states a trade-off between its privacy and accuracy guarantees:

▶ **Lemma 9** (Laplace Mechanism [17]). *Let $f : MS(\mathcal{X}) \to \mathbb{R}$ be a function with ℓ_1-sensitivity $\Delta_{1,f}$. For $\epsilon \geq 0$, the Laplace mechanism $\mathcal{M}^{f,\epsilon}_{Lap}$ takes a dataset $D \in MS(\mathcal{X})$ as input and outputs $f(D) + Lap(\Delta_{1,f}/\epsilon)$. The mechanism $\mathcal{M}^{f,\epsilon}_{Lap}$ satisfies ϵ-DP. Furthermore, for every $D \in MS(\mathcal{U})$ and $t \geq 0$,*

$$\Pr\left[|\mathcal{M}^{f,\epsilon}_{Lap}(D) - f(D)| \geq t \cdot \Delta_{1,f}/\epsilon\right] \leq e^{-t}.$$

We note that the definition of accuracy depends on the application and may be different in each context. Therefore, we do not provide a general definition and instead define accuracy for each concrete application in the subsequent sections.

Differential privacy in the continual and interactive setting. The mechanisms discussed so far are referred to as *non-interactive mechanisms (NIMs)* or *batch* mechanisms, as they take a single input dataset and produce a single output. In the rest of this paper, we define settings where either multiple different queries are asked to the mechanism, resulting in an *interactive mechanism (IM)*, or the dataset is changed through update operation, resulting in a *continual mechanism (CM)*.

Note that these mechanisms are differential privacy analogues of randomized algorithms and data structures in the non-private setting: A non-interactive mechanism corresponds to a *static* (or *batch*) *algorithm*, while an interactive mechanism corresponds to *static data structure*. A continual mechanism allows both queries and updates and corresponds to *dynamic data structures*. While the adaptivity of the sequence of operations is a common concern for the correctness and running time of randomized data structures, in differential privacy, adaptivity is additionally a concern for privacy. Concretely, note that the above definitions and composition theorems only hold for non-interactive mechanisms. In the following sections, we will discuss how to define and analyze the privacy properties of interactive and continual mechanisms.

2 The Continual Setting

In many real-world applications, data is not available in a single static batch but is collected incrementally over time or the dataset is changing over time. In such settings, it is often necessary to update the output continually in order to provide timely and relevant analysis of the dataset. For example, an online movie platform may collect user interactions, such as who watches which movie, and update its recommendation model accordingly. In this scenario, the platform must ensure that the sequence of recommendations over time does not reveal sensitive information, such as whether a user has watched a specific movie. The problem of dynamically receiving data and updating the output while protecting the privacy of each data point is studied in the *continual observation model* of differential privacy [18, 10], which has recently become a popular research topic (see e.g. [8, 9, 26, 32, 33, 2, 1, 16, 31, 4, 23] for CMs for various problems on sequences of numbers, [34, 29, 46] for CMs for set cardinality, [11, 40, 14] for CMs for problems in databases, [7, 30] for CMs for various histogram queries, [42, 25, 26, 41, 22] for CMs for various graph problems, [15] for CMs for clustering problems in the Euclidean space, and [45, 5] for CMs for releasing synthetic data).

In the non-private setting, the goal is to design dynamic algorithms that are accurate, memory-efficient, and time-efficient. In the private setting, however, one must additionally ensure the privacy of the input data, which is always in conflict with accuracy. Nonetheless, with the emergence of new algorithmic techniques, several continual mechanisms have

been developed that provide meaningful trade-offs between privacy and accuracy. These mechanisms have been shown to be practical and are already deployed in real-world systems [24, 3, 12, 13, 47, 48].

Formally, a mechanism $\mathcal{M}$ in the continual observation model is defined by a randomized function $f_{\mathcal{M}}$ that maps a current state and an incoming update to a new state and an output. The state represents the internal memory of the mechanism and may, for example, include the current dataset or relevant summary statistics. For simplicity, we assume the mechanism starts from an initial state corresponding to an empty dataset, though this assumption can be relaxed.

The mechanism $\mathcal{M}$ processes a sequence of update instructions $\sigma = (x_1, x_2, \dots)$ as follows[2]: Starting from the (empty) initial state, at each step t, the mechanism applies $f_{\mathcal{M}}$ to its current state and the incoming update x_t, yielding a new internal state and an output value y_t. $\mathcal{M}$ then returns y_t. We denote the sequence of outputs $(y_1, y_2, \dots)$ generated in response to the update sequence σ by the random variable $\mathcal{M}(\sigma)$. We also refer to such a mechanism as a *continual mechanism*.

Depending on the problem definition, each update x_i can be a data record in $\mathcal{X}$ to be inserted into the dataset or an instruction that arbitrarily modifies the current dataset (e.g., it deletes certain records). For simplicity, in the remainder of this section, we assume that each update x_t is a data record in $\mathcal{X}$ to be inserted in the dataset. We note that all the definitions can naturally be extended to non-empty initial datasets and to general update operations.

Recall that in the non-interactive model, the differential privacy guarantee is defined over pairs of neighboring datasets. In the continual setting, on the other hand, the notion of neighboring is defined over sequences of data records. For example, if $\mathcal{X} = \{0, 1\}$, we can define two sequences of data records $\sigma = (x_1, \dots, x_T)$ and $\sigma' = (x'_1, \dots, x'_T)$ of equal length $T \in \mathbb{N}$ to be neighboring if they differ at exactly one position, i.e., there exists $i \in [T]$ such that $x_j = x'_j$ for all $j \neq i$. Using this definition of neighboring in the definition of DP leads to so-called *event-level privacy*.

▶ **Definition 10** ((ϵ, δ)-Differential Privacy for Continual Mechanisms [18, 10])**.** *Let $\mathcal{M}$ be a continual mechanism, and let $\mathcal{Y}$ denote the output space of $\mathcal{M}$ at each step. For $\epsilon \geq 0$ and $0 \leq \delta \leq 1$, $\mathcal{M}$ is said to be (ϵ, δ)-differentially private (or (ϵ, δ)-DP for short) until time step $T \in \mathbb{N}$ if, for every pair of neighboring sequences $\sigma, \sigma' \in \mathcal{X}^T$ and every set of output sequences $V \subseteq \mathcal{Y}^T$, we have*

$$Pr[\mathcal{M}(\sigma) \in V] \leq e^{\epsilon} Pr[\mathcal{M}(\sigma') \in V] + \delta.$$

$\mathcal{M}$ is ϵ-differentially private (or ϵ-DP for short) if $\delta = 0$.

In the above definition, the continual mechanism is differentially private against an *oblivious* adversary. In the *adaptive* setting, however, an *adversary* generates neighboring sequences over time, adaptively, based on the previous outputs of the mechanism. Specifically, given a continual mechanism $\mathcal{M}$, an adversary $\mathcal{A}$ interacts with an *oracle* holding a secret bit $b \in \{0, 1\}$. An oracle is a very limited algorithm that behaves as follows: At each time step, $\mathcal{A}$ issues a pair of updates (u_0, u_1), and the oracle forwards u_b to $\mathcal{M}$. Observing the corresponding output of $\mathcal{M}$, the adversary then adaptively generates the next pair of updates. If, at any time step, the sequences of u_0s and u_1s does not satisfy the required neighboring relation, the oracle halts the interaction.

[2] More generally, $\mathcal{M}$ can be an interleaved sequence of updates and (parametrized) queries, but for simplicity we just assume that one unparametrized query is fixed, and $\mathcal{M}$ answers that query after receiving each update.

We say that the mechanism $\mathcal{M}$ is (ϵ, δ)-differentially private against adaptive adversaries if, for every adversary $\mathcal{A}$, the distributions of output sequences induced by oracle bits $b = 0$ and $b = 1$ are (ϵ, δ)-indistinguishable.

3 Example: Continual Binary Counting

The first and most foundational problem studied in the continual observation model is the *continual binary counting problem*. In this problem, the continual mechanism $\mathcal{M}$ receives a bit x_t at each step $t \in \mathbb{N}$ and returns an estimate y_t of the cumulative sum $s_t = \sum_{i=1}^{t} x_t$. If x_t is an arbitrary real number, the problem is called *continual counting problem*. Two input sequences of the same length are considered to be neighboring if they are identical except for a single time step. The goal is to guarantee differential privacy while providing accuracy, where accuracy is measured as follows:

▶ **Definition 11** ((α, β)-Accuracy for Continual Counter). *For $T \in \mathbb{N}$, $0 < \alpha$ and $0 < \beta \leq 1$, a continual counting mechanism $\mathcal{M}$ is said to be* (α, β)-accurate for T steps *if, for every input sequence $\sigma = (x_1, \ldots, x_T) \in \mathcal{X}^T$,*

$$\Pr[\max_{t \in [T]} |y_t - s_t| \leq \alpha] \geq 1 - \beta,$$

where $s_t = \sum_{i=1}^{t} x_t$ and y_t is the output of $\mathcal{M}$ at step t. We say that $\mathcal{M}$ has additive error α *over the first T steps* with high probability *if it is (α, β)-accurate for T steps, where $\beta = 1/T^c$ for some constant $c > 0$.*

First simple mechanism (Laplace mechanism on each output): A trivial design of a continual counting mechanism $\mathcal{M}$ is to run $T \in \mathbb{N}$ instances of the static Laplace mechanism to privatize the sums s_t for $t \in [T]$. Specifically, at each step $t \in [T]$, the mechanism $\mathcal{M}$ computes s_t and outputs $y_t = s_t + \mathrm{Lap}(1/\epsilon')$, where $\mathrm{Lap}(1/\epsilon')$ is a fresh sample of Laplace noise with scale $1/\epsilon'$. Since changing a single input bit affects each s_t by at most 1, by Lemma 9, the output at each individual step is ϵ'-DP. Thus, applying basic composition (Lemma 4), the sequence of the first T outputs is $T \cdot \epsilon'$-DP. To guarantee $\mathcal{M}$ is ϵ-DP, we must therefore set $\epsilon' = \epsilon/T$. This implies that for each step $t \in [T]$, with a constant probability, $|y_t - s_t| = |\mathrm{Lap}(T/\epsilon)|$ is $\Omega(T/\epsilon)$.

Second simple mechanism (Laplace mechanism on each input) Another trivial approach is to run T instances of Laplace mechanism on the input bits x_t instead of s_t. That is, at each step $t \in [T]$, the continual counting mechanism $\mathcal{M}$ sets $\hat{x}_t = x_t + \mathrm{Lap}(1/\epsilon)$. Since two neighboring input sequences (of length $T \in \mathbb{N}$) differ on at most one coordinate, all but one Laplace mechanism receive the same input for both sequences. Thus, by Lemma 9 and Lemma 4, the sequence $(\hat{x}_1, \ldots, \hat{x}_T)$ is ϵ-DP. Since the output sequence of $\mathcal{M}$ is computed as a post-processing of $(\hat{x}_1, \ldots, \hat{x}_T)$, by Lemma 3, it is distinguishable for two neighboring input sequences. Thus, $\mathcal{M}$ is ϵ-DP. At step T, $|y_T - s_T|$ equals the sum of T i.i.d. samples from $\mathrm{Lap}(1/\epsilon)$. By a standard Chernoff bound, this implies that with high probability, $|y_T - s_T|$ is $\Omega(\sqrt{T})$.

The two naive designs for continual counting illustrate a fundamental trade-off between privacy and accuracy. In the first approach, independent Laplace noise is added to each cumulative sum $s_t = \sum_{i=1}^{t} x_i$. Since each input bit x_i contributes to up to T such sums, satisfying ϵ-differential privacy requires the Laplace scale to be T/ϵ. This, however, results in an additive error that grows linearly with T. In contrast, the second approach adds Laplace noise to each input bit x_t individually and then sums the noisy inputs in a post-processing

step. Here, each input only affects the output of a single Laplace mechanism, allowing for a smaller Laplace scale of $1/\epsilon$. However, because the output at step $t \in [T]$ includes t independent noise, the additive error grows polynomially in t.

To overcome this, Dwork, Naor, Pitassi and Rothblum [19] and Chan, Li, Shi, and Xu [10] proposed a new approach, the so-called *binary (tree) mechanism*, where each input bit is used in the input of $O(\log T)$ Laplace mechanisms, and each output is computed by summing $O(\log T)$ noisy values: Instead of adding noise to each individual bit or to every prefix sum, they add noise to sums of subsequences of the form $(x_{(k-1)\cdot 2^i+1}, \dots, x_{k\cdot 2^i})$ for every $0 \le i \le \log(T)$ and $1 \le k \le T/2^i$. More precisely, at time step $k \cdot 2^i$ (i.e., when the complete subsequence is available), they calculate and store the noisy sum $\sum_{j=(k-1)\cdot 2^i+1}^{k\cdot 2^i} x_j + \text{Lap}(\frac{\log(T)+1}{\epsilon})$. Then, to output the noisy cumulative sum at step $t \in [T]$, they partition the input sequence $(x_1, \dots, x_t)$ into at most $\log(T)$ disjoint subsequences of the above form and return the sum of their noisy values.

Another way to view this dynamic data structure is through a binary tree over time steps 1 to T, where each node stores the noisy sum of the corresponding bits in its subtree, i.e., runs the Laplace mechanism on this partial sum. At each level of the tree, the input bits are partitioned between the nodes. Thus, the (parallel) composition of Laplace mechanisms at each level is $\frac{\epsilon}{\log(T)+1}$-DP. Composing the $\log(T)+1$ levels and applying the basic composition theorem, we conclude that the joint output of all Laplace mechanisms is ϵ-DP. Since the noisy cumulative sum at every step is a post-processing of these noisy partial sums, this implies that the continual counter is ϵ-DP.

For the accuracy analysis, note that each output equals the true cumulative sum plus at most $\log(T)$ i.i.d. noise drawn from $\text{Lap}\left(\frac{\log(T)+1}{\epsilon}\right)$. Chan et al. [10] showed that the sum of these $\log(T)$ Laplace random variables is $O(\sqrt{\log(T)}\frac{\log(T)+1}{\epsilon}\sqrt{\log(1/\beta)})$ with probability at least $1-\beta$. Applying a union bound over all T time steps, the maximum additive error across all steps is $O(\log(T)^{3/2}\sqrt{\log(T/\beta)}/\epsilon)$ with probability at least $1-\beta$.

4 Interactive Setting

Consider a hospital that collected patient records during the Covid-19 pandemic, i.e., it has a static dataset. Over time, researchers may issue a sequence of statistical queries to investigate long-term effects of Covid and its potential correlation with current medical conditions. These queries are *parametrized*, meaning that they are given one or multiple parameters determining which query is asked, and these parameters are often chosen *adaptively*, meaning that the parameters for each new query may be chosen based on the responses to previous ones. To protect each patient record, queries must satisfy certain properties, e.g., an analyst cannot ask whether Mr. X had Covid or not.

In the DP literature, a mechanism that holds a static dataset and answers multiple adaptive queries is usually referred to as an *interactive mechanism*. To formally define differential privacy for the interactive setting, we need to define a *view* random variable: Let $\mathcal{M}$ be an interactive mechanism, and let $\mathcal{A}$ be an analyst (or adversary) that adaptively asks queries from $\mathcal{M}$. We denote the mechanism $\mathcal{M}$ holding dataset D by $\mathcal{M}(D)$. The *view* of the analyst $\mathcal{A}$ interacting with $\mathcal{M}(D)$, denoted $\text{View}(\mathcal{A}, \mathcal{M}(D))$, is the sequence $(r_{\mathcal{A}}, q_1, a_1, q_2, a_2, \dots)$ consisting of the analyst's internal randomness $r_{\mathcal{A}}$ and the sequence of queries q_i and corresponding answers a_i returned by $\mathcal{M}$.

▶ **Definition 12.** *An interactive mechanism $\mathcal{M}$ is said to be (ϵ, δ)-differentially private (or (ϵ, δ)-DP for short) if for every adversary $\mathcal{A}$ and every pair of neighboring datasets D_0 and D_1, the random variables View$(\mathcal{A}, \mathcal{M}(D_0))$ and View$(\mathcal{A}, \mathcal{M}(D_1))$ are (ϵ, δ)-indistinguishable. We say that $\mathcal{M}$ is ϵ-differentially private (or ϵ-DP for short) if $\delta = 0$.*

A fundamental problem in the interactive setting is *threshold testing*, where the mechanism holds a static dataset $D \in \mathrm{MS}(\{0,1\})$ consisting of binary values, and receives a sequence of integer thresholds over time. Specifically, at each time step $t \in \mathbb{N}$, the analyst sends a threshold query $\texttt{thresh}_t \in \mathbb{N}$, asking whether the sum $\sum_{x \in D} x$ exceeds the given threshold $\texttt{thresh}_t$. The mechanism responds with either $\top$ (yes) or $\bot$ (no). Once the mechanism returns $\top$, it is required to halt. For simplicity, we assume the mechanism continues to return $\top$ indefinitely after it outputs $\top$ for the first time.

Two datasets $D_0, D_1 \in \mathrm{MS}(\{0,1\})$ are considered *neighboring* if they differ in the presence or absence of one element. The goal is to provide accurate answers to threshold queries while preserving privacy, where accuracy is defined as follows:

▶ **Definition 13.** *A threshold testing mechanism $\mathcal{M}$ is said to be (α, β)-accurate until step $T \in \mathbb{N}$ if, for every input dataset $D \in MS(\{0,1\})$, with probability at least $1-\beta$, the following conditions hold for every $t \in [T]$:*

1. ***(Above threshold).*** *If $\sum_{x \in D} x \geq \texttt{thresh}_t + \alpha$, then $\mathcal{M}(D)$ returns $\top$ at step t.*
2. ***(Below threshold).*** *If for every $i \leq t$, $\sum_{x \in D} x \leq \texttt{thresh}_i - \alpha$, then $\mathcal{M}(D)$ returns $\bot$ at all steps $i \leq t$.*

A well-known continual threshold-testing mechanism is the sparse vector technique (SVT) mechanism [19, 36]. The SVT mechanism allows an analyst to issue adaptive threshold queries on a static dataset while preserving differential privacy. This mechanism is described in Algorithms 1 and 2.

In Algorithm 1, the mechanism is initialized with a dataset D and a privacy parameter ϵ. It first computes the true count $s = \sum_{x \in D} x$ and then draws a sample Z_0 from $\mathrm{Lap}(2/\epsilon)$. This sample will be repeatedly used in all future comparisons. Upon receiving a threshold query $\texttt{thresh}_t$, the mechanism proceeds as follows (see Algorithm 2): If it has already returned $\top$ in any prior step, it returns $\top$. Otherwise, using the old noise Z_0 and a fresh noise $Z_t \sim \mathrm{Lap}(4/\epsilon)$, it compares $s + Z_0 \geq \texttt{thresh}_t + Z_t$. If the inequality holds, then the mechanism returns $\top$ and sets a flag to indicate that it has exceeded a threshold; otherwise, it returns $\bot$.

▶ **Lemma 14** (Privacy [19] and Accuracy [35] of SVT)**.** *For every $\epsilon \geq 0$, the SVT mechanism described in Algorithms 1 and 2 satisfies ϵ-DP. Moreover, for every $0 < \beta \leq 1$ and $T \in \mathbb{N}$, this mechanism is $(\alpha = O(\log(T/\beta)/\epsilon), \beta)$-accurate until step T.*

Algorithm 1 $\texttt{svt}_\epsilon$ (initialization).

Input: privacy parameter $\epsilon > 0$; dataset $D \in \mathrm{MS}(\{0,1\})$.

1: $\epsilon_1 = \epsilon_2 = \epsilon/2$
2: $s = \sum_{x \in D} x$
3: $\texttt{varflag} = 0$
4: Draw $Z_0 \sim \mathrm{Lap}(1/\epsilon_1)$

Algorithm 2 $\texttt{svt}_\epsilon$ (update).

Input: threshold $\texttt{thresh}_t$
Output: an element of $\{\top, \bot\}$.
if $\texttt{varflag} = 1$ **then**
  return $\top$
end if
Draw $Z_t \sim \text{Lap}(2/\epsilon_2)$
if $s + Z_0 \geq \texttt{thresh}_t + Z_t$ **then**
  $\texttt{varflag} = 1$
  return $\top$
else
  return $\bot$
end if

5 Designing New Continual Mechanisms

In this section, we introduce new composition tools that enable a modular construction of complex continual mechanisms executing not only non-interactive mechanisms (NIMs) but also continual and interactive mechanisms as sub-mechanisms. As we mentioned before, the composition theorems presented in Section 1 only apply to NIMs. This raises a question: can interactive or continual sub-mechanisms always be decomposed into NIMs followed by post-processing, thereby enabling privacy analysis of the complex mechanism using composition theorems for NIMs and the post-processing lemma? We will first show that such a decomposition is not always possible and then present composition tools that simplify the privacy analysis of the mechanism with modular design. Due to complicated dependencies between the outputs and inputs of the sub-mechanisms and their potential influence on each other, analyzing the privacy of these mechanisms from scratch can be challenging.

Roughly speaking, interactive mechanisms can be viewed as a special case of continual mechanisms, where the first input update is a dataset and all subsequent updates are adaptively chosen queries with no effect on the dataset. When modeling an interactive mechanism as a continual mechanism, two input sequences are considered neighboring if their first element (i.e., the dataset) are neighboring, and all subsequent elements (i.e., the queries) are identical. This perspective allows us to unify continual and interactive mechanisms and refer to both of them as *continual mechanisms (CMs)*.

Continual mechanisms are not necessarily compositions of non-interactive mechanisms. We begin by demonstrating that an (ϵ, δ)-DP continual mechanism cannot, in general, be expressed as the post-processing of a composition of NIMs, each using independent random coins, such that the composition satisfies (ϵ, δ)-DP. To illustrate this, we compare the privacy analysis of the binary-tree mechanism (Section 3) and of the Sparse Vector Technique (SVT) mechanism (Section 4).

Let $T \in \mathbb{N}$ and $\epsilon \geq 0$. The binary-tree mechanism draws $2T - 1$ independent Laplace noises from $\text{Lap}((\log T + 1)/\epsilon)$, one for each node of the binary tree. Each noise is then used individually to privatize a partial sum of the input sequence, and the output at each time step is computed by combining a subset of these privatized values. This structure enables a clean decomposition of the mechanism into NIMs with independent randomness, followed by a post-processing.

In contrast, the privacy analysis of the SVT mechanism is different: given a dataset D, this mechanism samples $T+1$ independent Laplace noises: $Z_0 \sim \text{Lap}(2/\epsilon)$ and $Z_i \sim \text{Lap}(4/\epsilon)$ for each $i \in [T]$. At time step $t \in [T]$, given a threshold $\texttt{thresh}_t$, the mechanism determines the output based on the comparison $\sum_{x \in D} x + Z_t \geq \texttt{thresh}_t + Z_0$. Each noisy sum $\sum_{x \in D} x + Z_t$ can be considered as an instance of the Laplace mechanism with the privacy parameter $\epsilon/4$, and the comparison can be interpreted as a post-processing of this $\epsilon/4$-DP mechanism. However, since each data record in D contributes to the input of all T Laplace mechanisms, the composition of these mechanisms would be $\frac{T}{4\epsilon}$-DP.

To avoid this undesirable polynomial privacy parameter, the SVT mechanism is analyzed directly by comparing the view of any adversary interacting with this mechanism for any two neighboring datasets. This analysis yields an ϵ-DP guarantee. Notably, this proof crucially exploits a shared source of randomness, i.e., the random variable Z_0, to get a better privacy guarantee, something that the privacy analysis by composition cannot capture as it requires the composed mechanisms to use independent random coins.

Composition theorems for continual mechanisms. This limitation implies that, in general, when analyzing a mechanisms executing multiple continual sub-mechanisms, we cannot simply decompose each continual sub-mechanism into NIMs and then apply standard composition theorems for NIMs to derive overall privacy guarantees. This challenge has led to a line of research on extending the composition theorems for NIMs to *concurrent composition* theorems for CMs [43, 37, 44, 27, 28]. Here, the term *concurrent* captures the fact that the inputs to the composed CMs are generated concurrently: Specifically, the input to one mechanism may depend on the outputs of *all* other mechanisms up to that point.

Although we do not formally define concurrent composition in this paper, we highlight a recent work by Henzinger, Safavi, and Vadhan [28] showing that composition theorems for NIMs, such as Theorem 4 and Theorem 5, still hold in the more general setting of the concurrent composition of CMs, provided that the composed mechanisms are differentially private against *adaptive* adversaries.

Moreover, they show a separation between parallel composition in the non-interactive model (Theorem 6) and the concurrent parallel composition of CMs, where an adversary adaptively selects which mechanism receives non-identical input sequences. To address this, they develop new concurrent parallel composition theorems.

Assume a mechanism $\mathcal{M}$ uses multiple CMs as subroutines. Henzinger et al. [28] also formalize which types of *post-processing* on the outputs of these sub-mechanisms are permissible in order to preserve the privacy guarantee of the concurrent composition of the sub-mechanisms: They present a general framework for combining the given inputs and the previous outputs of sub-mechanisms to generate new inputs for the sub-mechanisms and then generate the final output based on the outcomes of the sub-mechanisms. This formalism results in a simple privacy analysis for complex mechanisms executing continual sub-mechanisms.

6 Conclusions

Continual mechanisms are the analogue of dynamic data structures in the differentially private setting. Apart from analyzing their accuracy, time, and space requirements, their privacy properties need to be proven, which can be challenging. New composition theorems for continual mechanisms simplify this task and will hopefully support the design of continual mechanisms for various applications with improved tradeoffs between privacy and accuracy.

References

1 Joel Daniel Andersson and Rasmus Pagh. Streaming private continual counting via binning. *CoRR*, abs/2412.07093, 2024. doi:10.48550/arXiv.2412.07093.

2 Joel Daniel Andersson, Rasmus Pagh, and Sahel Torkamani. Improved counting under continual observation with pure differential privacy. *CoRR*, abs/2408.07021, 2024. doi:10.48550/arXiv.2408.07021.

3 Apple. Privacy-preserving contact tracing, https://covid19.apple.com/contacttracing, 2021.

4 Jean Bolot, Nadia Fawaz, S. Muthukrishnan, Aleksandar Nikolov, and Nina Taft. Private decayed sum estimation under continual observation. *CoRR*, abs/1108.6123, 2011. arXiv:1108.6123.

5 Mark Bun, Marco Gaboardi, Marcel Neunhoeffer, and Wanrong Zhang. Continual release of differentially private synthetic data from longitudinal data collections. *Proc. ACM Manag. Data*, 2(2):94, 2024. doi:10.1145/3651595.

6 Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil Vadhan. Differentially private release and learning of threshold functions. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 634–649. IEEE, 2015. doi:10.1109/FOCS.2015.45.

7 Adrian Rivera Cardoso and Ryan Rogers. Differentially private histograms under continual observation: Streaming selection into the unknown. In *Proc. 25th AISTATS*, pages 2397–2419, 2022. URL: https://proceedings.mlr.press/v151/rivera-cardoso22a.html.

8 Ho-Leung Chan, Tak-Wah Lam, Lap-Kei Lee, and Hing-Fung Ting. Continuous monitoring of distributed data streams over a time-based sliding window. *Algorithmica*, 62(3-4):1088–1111, 2012. doi:10.1007/S00453-011-9506-5.

9 T-H Hubert Chan, Mingfei Li, Elaine Shi, and Wenchang Xu. Differentially private continual monitoring of heavy hitters from distributed streams. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 140–159. Springer, 2012. doi:10.1007/978-3-642-31680-7_8.

10 T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3):26:1–26:24, 2011. doi:10.1145/2043621.2043626.

11 Rachel Cummings, Sara Krehbiel, Kevin A. Lai, and Uthaipon Tao Tantipongpipat. Differential privacy for growing databases. In *Proc. 31st NeurIPS*, pages 8878–8887, 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/ac27b77292582bc293a51055bfc994ee-Abstract.html.

12 Aref N. Dajani, Amy D. Lauger, Phyllis E. Singer1, Daniel Kifer, Jerome P. Reiter, Ashwin Machanavajjhala, Simson L. Garfinkel, Scot A. Dahl, Matthew Graham Vishesh Karwa, Hang Kim, Philip Leclerc, Ian M. Schmutte, William N. Sexton, Lars Vilhuber, and John M. Abowd. The modernization of statistical disclosure limitation at the U.S. Census Bureau, 2017. URL: https://www2.census.gov/cac/sac/meetings/2017-09/statistical-disclosure-limitation.pdf.

13 Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3571–3580, 2017. URL: http://papers.nips.cc/paper/6948-collecting-telemetry-data-privately.

14 Wei Dong, Zijun Chen, Qiyao Luo, Elaine Shi, and Ke Yi. Continual observation of joins under differential privacy. *Proceedings of the ACM on Management of Data*, 2(3):1–27, 2024.

15 Max Dupré la Tour, Monika Henzinger, and David Saulpic. Making old things new: a unified algorithm for differentially private clustering. In *Proc. 41th ICML*, 2024.

16 Krishnamurthy Dj Dvijotham, H. Brendan McMahan, Krishna Pillutla, Thomas Steinke, and Abhradeep Thakurta. Efficient and near-optimal noise generation for streaming differential privacy. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS*

2024, Chicago, IL, USA, October 27-30, 2024, pages 2306–2317. IEEE, 2024. doi:10.1109/FOCS61266.2024.00135.

17 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 265–284. Springer, 2006. doi:10.1007/11681878_14.

18 Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proc. of the Forty-Second ACM Symp. on Theory of Computing (STOC'10)*, pages 715–724, 2010. doi:10.1145/1806689.1806787.

19 Cynthia Dwork, Moni Naor, Omer Reingold, Guy N Rothblum, and Salil Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 381–390, 2009. doi:10.1145/1536414.1536467.

20 Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014. doi:10.1561/0400000042.

21 Cynthia Dwork, Guy N. Rothblum, and Salil Vadhan. Boosting and differential privacy. In *Proceedings of the IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60, 2010. doi:10.1109/FOCS.2010.12.

22 Alessandro Epasto, Quanquan C. Liu, Tamalika Mukherjee, and Felix Zhou. The power of graph sparsification in the continual release model. *CoRR*, abs/2407.17619, 2024. doi:10.48550/arXiv.2407.17619.

23 Alessandro Epasto, Jieming Mao, Andres Muñoz Medina, Vahab Mirrokni, Sergei Vassilvitskii, and Peilin Zhong. Differentially private continual releases of streaming frequency moment estimations. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 48:1–48:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ITCS.2023.48.

24 Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067. ACM, 2014. doi:10.1145/2660267.2660348.

25 Hendrik Fichtenberger, Monika Henzinger, and Wolfgang Ost. Differentially private algorithms for graphs under continual observation. In *29th Annual European Symposium on Algorithms, ESA*, 2021. doi:10.4230/LIPIcs.ESA.2021.42.

26 Hendrik Fichtenberger, Monika Henzinger, and Jalaj Upadhyay. Constant matters: Fine-grained error bound on differentially private continual observation. In *International Conference on Machine Learning*, pages 10072–10092. PMLR, 2023. URL: https://proceedings.mlr.press/v202/fichtenberger23a.html.

27 Samuel Haney, Michael Shoemate, Grace Tian, Salil Vadhan, Andrew Vyrros, Vicki Xu, and Wanrong Zhang. Concurrent composition for interactive differential privacy with adaptive privacy-loss parameters. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1949–1963, 2023. doi:10.1145/3576915.3623128.

28 Monika Henzinger, Roodabeh Safavi, and Salil Vadhan. Concurrent composition for differentially private continual mechanisms. *arXiv preprint*, 2024. arXiv:2411.03299.

29 Monika Henzinger, A. R. Sricharan, and Teresa Anna Steiner. Private counting of distinct elements in the turnstile model and extensions. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2024, August 28-30, 2024, London School of Economics, London, UK*, volume 317 of *LIPIcs*, pages 40:1–40:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.APPROX/RANDOM.2024.40.

30 Monika Henzinger, A. R. Sricharan, and Teresa Anna Steiner. Differentially private continual release of histograms and related queries. In *The 28th International Conference on Artificial Intelligence and Statistics*, 2025. URL: https://openreview.net/forum?id=EYhGcfYpFS.

31 Monika Henzinger and Jalaj Upadhyay. Improved differentially private continual observation using group algebra. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2951–2970. SIAM, 2025. doi:10.1137/1.9781611978322.95.

32 Monika Henzinger, Jalaj Upadhyay, and Sarvagya Upadhyay. Almost tight error bounds on differentially private continual counting. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5003–5039, 2023. doi:10.1137/1.9781611977554.CH183.

33 Monika Henzinger, Jalaj Upadhyay, and Sarvagya Upadhyay. A unifying framework for differentially private sums under continual observation. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 995–1018. SIAM, 2024. doi:10.1137/1.9781611977912.38.

34 Palak Jain, Iden Kalemaj, Sofya Raskhodnikova, Satchit Sivakumar, and Adam D. Smith. Counting distinct elements in the turnstile model with differential privacy under continual observation. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/0ef1afa0daa888d695dcd5e9513bafa3-Abstract-Conference.html.

35 Palak Jain, Adam D. Smith, and Connor Wagaman. Time-aware projections: Truly node-private graph statistics under continual observation. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 127–145. IEEE, 2024. doi:10.1109/SP54263.2024.00196.

36 Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy. *arXiv preprint arXiv:1603.01699*, 2016. arXiv:1603.01699.

37 Xin Lyu. Composition theorems for interactive differential privacy. *Advances in Neural Information Processing Systems*, 35:9700–9712, 2022.

38 Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30, 2009. doi:10.1145/1559845.1559850.

39 Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008. doi:10.1109/SP.2008.33.

40 Yuan Qiu and Ke Yi. Differential privacy on dynamic data. *arXiv preprint arXiv:2209.01387*, 2022. doi:10.48550/arXiv.2209.01387.

41 Sofya Raskhodnikova and Teresa Anna Steiner. Fully dynamic algorithms for graph databases with edge differential privacy. *Proceedings of the ACM on Management of Data*, 3(2):1–28, 2025. doi:10.1145/3725236.

42 Shuang Song, Susan Little, Sanjay Mehta, Staal Vinterbo, and Kamalika Chaudhuri. Differentially private continual release of graph statistics. *arXiv preprint*, 2018. arXiv:1809.02575.

43 Salil Vadhan and Tianhao Wang. Concurrent composition of differential privacy. In *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II 19*, pages 582–604. Springer, 2021. doi:10.1007/978-3-030-90453-1_20.

44 Salil Vadhan and Wanrong Zhang. Concurrent composition theorems for differential privacy. In *55th Annual ACM Symposium on Theory of Computing*, 2023.

45 Tianhao Wang, Joann Chen, Zhikun Zhang, Dong Su, Yueqiang Cheng, Zhou Li, Ninghui Li, and Somesh Jha. Continuous release of data streams under both centralized and local differential privacy. In *CCS*, pages 1237–1253, 2021.

46 Dongdong Xie, Pinghui Wang, Quanqing Xu, Chuanhui Yang, and Rundong Li. Efficient and accurate differentially private cardinality continual releases. *Proceedings of the ACM on Management of Data*, 3(3):1–27, 2025. doi:10.1145/3725288.

47 Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018. arXiv:1812.02903.

48 Bing Zhang, Vadym Doroshenko, Peter Kairouz, Thomas Steinke, Abhradeep Thakurta, Ziyin Ma, Eidan Cohen, Himani Apte, and Jodi Spacek. Differentially private stream processing at scale. *Proc. VLDB Endow.*, 17(12):4145–4158, 2024. doi:10.14778/3685800.3685833.

Generalized Graph Packing Problems Parameterized by Treewidth

Barış Can Esmer
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, Saarland Informatics Campus, Germany

Dániel Marx
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

H-Packing is the problem of finding a maximum number of vertex-disjoint copies of H in a given graph G. H-Partition is the special case of finding a set of vertex-disjoint copies that cover each vertex of G exactly once. Our goal is to study these problems and some generalizations on bounded-treewidth graphs. The case of H being a triangle is well understood: given a tree decomposition of G having treewidth tw, the K_3-Packing problem can be solved in time $2^{\mathrm{tw}} \cdot n^{O(1)}$, while Lokshtanov et al. [*ACM Transactions on Algorithms* 2018] showed, under the Strong Exponential-Time Hypothesis (SETH), that there is no $(2-\epsilon)^{\mathrm{tw}} \cdot n^{O(1)}$ algorithm for any $\epsilon > 0$ even for K_3-Partition. Similar results can be obtained for any other clique K_d for $d \geq 3$. We provide generalizations in two directions:

- We consider a generalization of the problem where every vertex can be used at most c times for some $c \geq 1$. When H is any clique K_d with $d \geq 3$, then we give upper and lower bounds showing that the optimal running time increases to $(c+1)^{\mathrm{tw}} \cdot n^{O(1)}$. We consider two variants depending on whether a copy of H can be used multiple times in the packing.
- If H is not a clique, then the dependence of the running time on treewidth may not be even single exponential. Specifically, we show that if H is any fixed graph where not every 2-connected component is a clique, then there is no $2^{o(\mathrm{tw} \log \mathrm{tw})} \cdot n^{O(1)}$ algorithm for H-Partition, assuming the Exponential-Time Hypothesis (ETH).

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Graph Packing, Graph Partitioning, Parameterized Complexity, Treewidth, Pathwidth, pw-SETH, Single-Exponential Lower Bound, Slightly Superexponential Lower Bound

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.3

Related Version *Full Version*: `http://arxiv.org/abs/2509.06091`

1 Introduction

Parameterized complexity theory has proven instrumental in systematically understanding the computational complexity of various combinatorial problems under different parameterizations. Parameterization by treewidth implies tractability for a large number of fundamental algorithmic problems. A prominent line of research has emerged around classifying the complexity of classical NP-hard graph problems under this parameterization framework [8, 7].

In their influential work, Lokshtanov et al. [8] studied six classical combinatorial problems for which parameterized algorithms are known where the running-time dependence on treewidth is single exponential. They showed that, under Strong Exponential Time Hypothesis (SETH), the running times are optimal in the base of the exponent. Following this work, significant efforts have been devoted to generalizing and extending these results. In particular, five out of six problems in [8] have been put into a wider context and generalized to an infinite family of problems: q-Coloring was generalized into H-homomorphism problems [1, 10, 9], Independent Set (equivalent to Vertex Cover), MaxCut and Odd Cycle Transversal [2] into H-homomorphism deletion problem, and Dominating Set into general (σ, ρ)-dominating set problems [5]. Curticapean and Marx [3] showed tight lower bounds for the problem of

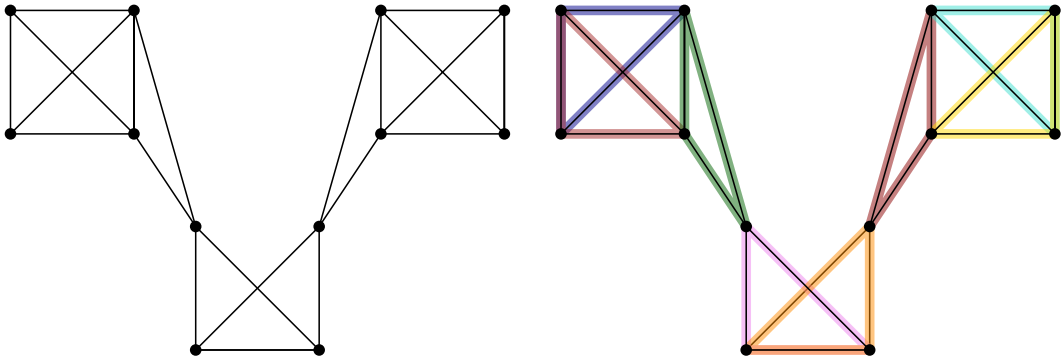

© Barış Can Esmer and Dániel Marx;
licensed under Creative Commons License CC-BY 4.0



counting perfect matchings, which was later extended into general factor problems. However, from the initial six problems studied by Lokshtanov et al. [8], the triangle packing problem has remained unaddressed by prior generalization efforts.

The $2^{tw} \cdot n^{\mathcal{O}(1)}$ running time for triangle packing stated in [8] is actually valid for any clique packing problem, instead of triangle with just three vertices. In this paper, we investigate further generalizations of the triangle packing problem. This line of research naturally gives rise to two conceptually distinct directions:

1. **Allowing vertices to be covered multiple times:** The motivation for this generalization stems naturally from closely related problems such as fractional packing, where vertices can inherently contribute fractionally or multiple times to different packings. To illustrate, Figure 1.1 shows a graph that does not admit a triangle partition, yet allows a collection of triangles that cover each vertex exactly twice. Motivated by this observation, we formally define and explore a generalized packing problem, where each vertex of G can be covered at most c times, for any fixed integer $c \geq 1$. Unlike $c = 1$ case, this generalization demands careful consideration of the definition, specifically on whether a copy of H may appear multiple times in the solution.
2. **Considering packings of arbitrary graphs:** The second natural direction involves generalizing triangle packing by replacing triangles with an arbitrary fixed graph H. This problem has already been studied in the literature under the name of H-partition in [6]. Specifically, in [6] the authors prove the NP-Hardness of the H-partition problem where H has a connected component with at least three vertices. In this paper we let H be a connected graph with at least three vertices and study the complexity of the H-partition problem parameterized by treewidth.



(a) The graph under consideration.

(b) Each color represents a triangle in the collection. Observe that each vertex is covered exactly twice.

Figure 1.1 A graph that does not admit a triangle partition, but allows a collection of triangles such that each vertex is covered exactly twice.

1.1 Our Results

Let H be a fixed graph and $c \geq 1$ be an integer. Given a graph G, a subgraph Z of G is called a copy of H if Z is isomorphic to H. Moreover, if Z is a copy of H in G, we say that Z covers v if $v \in V(Z)$.

▶ **Definition 1.** *We say that a multiset (respectively, set)* $\mathcal{S} = \{(V_1, E_1), \ldots, (V_k, E_k)\}$ *of subgraphs of* G *is a* (c, H)*-multi-packing (respectively,* (c, H)*-single-packing) in* G *if*

1. *each* (V_i, E_i) *is isomorphic to* H *for* $1 \leq i \leq k$
2. *each vertex* v *of* G *is covered at most* c *times by the subgraphs in* $\mathcal{S}$*.*

The collection $\mathcal{S}$ *is called a* (c, H)*-multi-partition (respectively,* (c, H)*-single-partition) of* G *if each vertex* $v \in V(G)$ *is covered exactly* c *times.*

Observe that when c is equal to 1, two copies of H are not allowed to have any common vertices. Therefore, in this case, the notions of (c, H)-single-packing and $\big(c, H\big)$-multi-packing coincide, and we write H-packing to simpify the notation. We define H-partition in a similar way. The first problem we introduce, `Generalized Graph Packing`(H), asks the maximum size of an H-packing in G.

`Generalized Graph Packing`(H)
Input: A graph G
Output: The size of a largest H-packing in G.

For a fixed H, we show that `Generalized Graph Packing`(H) can be solved in time $2^{\mathcal{O}(w \cdot \log w)} \cdot n^{\mathcal{O}(1)}$ for graphs of treewidth w.

▶ **Theorem 2.** *Let H be an arbitrary graph such that it contains at least 3 vertices. Then,* `Generalized Graph Packing`(H) *can be solved in time $2^{\mathcal{O}(w \cdot \log(w))} \cdot n^{\mathcal{O}(1)}$ for all n-vertex graphs G where w is the treewidth of G.*

Then, we define partitioning problems in which each vertex of G must be covered the same number of times. For an arbitrary graph H, we let $c = 1$ which results in vertex disjoint copies of H.

`Generalized Graph Partitioning`(H)
Input: A graph G
Question: Is there an H-packing of G such that each vertex is covered exactly once?

Observe that `Generalized Graph Partitioning`(H) is a special case of the problem `Generalized Graph Packing`(H), which implies an algorithm with running time $2^{\mathcal{O}(w \cdot \log w)} \cdot n^{\mathcal{O}(1)}$ for the `Generalized Graph Partitioning`(H) problem on graphs of treewidth w. We show that this running time cannot be improved for many choices of H.

▶ **Theorem 3.** *Let H be an arbitrary graph with at least 3 vertices such that H is not a block graph. Then, there is no algorithm for* `Generalized Graph Partitioning`(H) *problem, that solves all instances G in time $2^{o(w \cdot \log w)} \cdot n^{\mathcal{O}(1)}$ where w is the treewidth of G, unless ETH fails.*

When H is a clique, we consider the more general problem in which a vertex $v \in V(G)$ can be covered more than once, at most c times for some $c \geq 1$. Therefore, we distinguish between two variants of the problem: one where each clique can be selected at most once, and another without restriction. The `Multi-Clique Packing`(c, d) problem asks the maximum size of a $\big(c, K_d\big)$-multi-packing in G.

`Multi-Clique Packing`(c, d)
Input: A graph G
Output: The size of a largest $\big(c, K_d\big)$-multi-packing in G.

Similar to the `Multi-Clique Packing`(c, d) problem, the `Single-Clique Packing`(c, d) problem asks the maximum size of a (c, K_d)-single-packing in G.

`Single-Clique Packing`(c, d)
Input: A graph G
Output: The size of a largest (c, K_d)-single-packing in G.

In the full version of the paper we show that `Single-Clique Packing`(c, d) admits a single-exponential time algorithm where the same algorithmic result also applies to `Multi-Clique Packing`(c, d) with slight modifications.

▶ **Theorem 4.** *Let $c \geq 1$ and $d \geq 3$ be integers. Then,* `Single-Clique Packing`(c, d) *can be solved in time $(c+1)^w \cdot n^{\mathcal{O}(1)}$ for all n-vertex graphs G given together with a tree decomposition of width at most w.*

Moreover, we define partitioning problems where H is a clique.

`Multi-Clique Partitioning`(c, d)
Input: A graph G
Question: Is there a $\big(c, K_d\big)$-multi-partition of G?

`Single-Clique Partitioning`(c, d)
Input: A graph G
Question: Is there a (c, K_d)-single-partition of G?

Similarly, we prove that this running time is optimal up to polynomial factors in the size of the input graph.

▶ **Theorem 5.** *Let $c \geq 1$ and $d \geq 3$ be integers. If there exists an $\varepsilon > 0$ such that* `Multi-Clique Partitioning`(c, d) *can be solved in time $(c+1-\varepsilon)^w \cdot n^{\mathcal{O}(1)}$ for all n-vertex graphs G given together with a path decomposition of width at most w, then the* **pw**-*SETH fails.*

Finally, we show that the lower bound result in Theorem 5 can also be transferred similarly.

▶ **Theorem 6.** *Let $c \geq 1$ and $d \geq 3$ be integers. If there exists an $\varepsilon > 0$ such that* `Single-Clique Partitioning`(c, d) *can be solved in time $(c+1-\varepsilon)^w \cdot n^{\mathcal{O}(1)}$ for all n-vertex graphs G given together with a path decomposition of width at most w, then the* **pw**-*SETH fails.*

2 Technical Overview

In this section, we will give an overview of the techniques and ideas presented in the paper.

2.1 Preliminaries

A graph H is k-connected if for each $A \subseteq V(H)$ such that $|A| = k-1$ it holds that $H \setminus A$ is connected. A graph is also called biconnected if it is 2-connected, and a block is a maximal 2-connected component of H. Similarly, a graph H is called a block graph if every block of H is a clique.

A vertex v of a connected graph H is a cutvertex if $G \setminus v$ is disconnected. In this paper, we refer to both the blocks of a graph and the nodes of the corresponding block tree interchangeably, with a slight abuse of notation. While formally distinct, we find it convenient to treat them as equivalent entities for the sake of clarity and brevity. For a vertex $h \in V(H)$ and a block $B \in \mathcal{B}_H$, we write $h \in B$ if $h \in V(B)$.

▶ **Definition 7.** *For a function f and a set X, we let $f\restriction_X$ denote the restriction of f to $X \cap \operatorname{dom}(f)$. Similarly, we let $f\restriction_{-X}$ denote the restriction of f to $\operatorname{dom}(f) \setminus X$. Finally, for $v \notin \operatorname{dom}(f)$ and a value y, we let $f\restriction_{[v\to y]}$ denote a function g which is defined as*

$$g(x) = \begin{cases} f(x) & \text{if } x \in \operatorname{dom}(f), \\ y & \text{if } x = v. \end{cases}$$

We use the Iverson bracket $[\![P]\!]$, which is defined to be 1 if the proposition P is true and 0 otherwise.

2.2 Gadgets

In this paper, we derive our lower bounds via reductions from well-studied base problems whose intractability is established under standard complexity hypotheses. Each reduction makes use of compact, purpose-built gadgets – small graphs whose behavior can be precisely engineered. Embedding these gadgets into our constructions yields intuitive, transparent proofs that highlight the underlying ideas without excessive technical overhead.

More formally, we define a gadget G as a graph with designated portal vertices $\{p_1, \ldots, p_\ell\} \subseteq V(G)$, where the vertices $\operatorname{int}(G) := (G \setminus \{p_1, \ldots, p_\ell\})$ are called internal vertices. We say that a graph E is an extension of a gadget G if E contains G as an induced subgraph, where each internal vertex v of G satisfies

$$N_E(v) \subseteq V(G).$$

In other words, in an extension E, only the portal vertices of G can have neighbors outside of G.

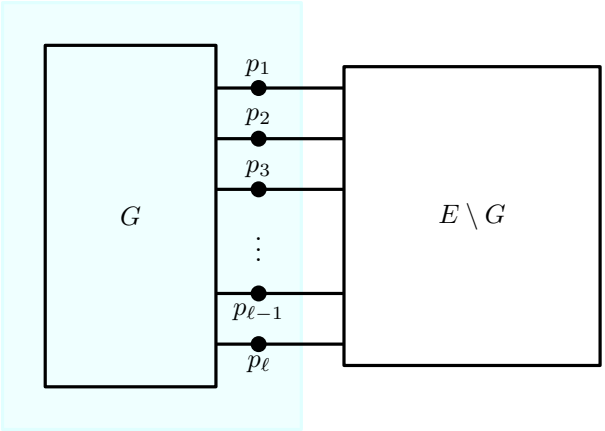


Figure 2.1 Illustration of an extension of gadget G with ℓ portal vertices: the blue region on the left highlights G along with its portals, while the rectangle on the right depicts the remainder $E \setminus G$.

We also require gadgets to behave in a structured way. Whenever a copy of H intersects the gadget, it must lie entirely within the gadget when connected to a larger graph. We formalize this in the following definition.

▶ **Definition 8.** *Let G be a gadget and P_G be the set of its portal vertices. For a graph H and $c \geq 1$, we say that G is (c, H)-internally coherent if, for any extension E of G and any (c, H)-multi-partition / (c, H)-single-partition $\mathcal{Z}$ of E, the following holds: If there exists $Z \in \mathcal{Z}$ that contains an internal vertex of G, then all vertices in Z must belong to G. Formally,*

$$\Big(V(Z) \cap \big(V(G) \setminus P_G\big)\Big) \neq \emptyset \implies V(Z) \subseteq V(G).$$

Next, we define the relation realized by a gadget.

▶ **Definition 9.** *Let H be a graph, $c \geq 1$ and G be a gadget with portal vertices $P_G = \{p_1, \ldots, p_\ell\}$. We say G dist-(c, H)-realizes (respectively, arb-(c, H)-realizes) a relation $R \subseteq \{0, \ldots, c\}^\ell$ if the following holds:*

$r \in R \iff$ *There exists a (c, H)-single-packing (respectively, $\big(c, H\big)$-multi-packing) $\mathcal{Z}$ of G such that $\mathcal{Z}$ covers each internal vertex of G exactly c times, and each portal vertex p_i exactly r_i times for $1 \leq i \leq \ell$.*

We say that G strict-(c, H)-realizes R if it both dist-(c, H) and arb-(c, H)-realizes R.

▶ Remark 10. When $c = 1$, the notions of dist-(c, H)-realization and arb-(c, H)-realization coincide. In this case, we simply say that the relation R is H-realized by G, instead of using the term "strict-$(1, H)$".

We say a relation $R \subseteq \{0, \ldots, c\}^\ell$ is (x, d)-regular for $d \geq 1$ and $0 \leq x \leq d - 1$ if for each $r \in R$, the weight of r is equivalent to $x \mod d$, i.e. $\mathbf{w}(r) \coloneqq \left(\sum_{i \in [\ell]} r_i\right) = x \mod d$. Moreover, a relation R has weight X if $\left(\max_{r \in R} \mathbf{w}(r)\right) = X$. Recall that a gadget is a small, engineered graph used to enforce specific behaviors in a reduction. We streamline the lower bound constructions by building general-purpose gadgets that can realize any relation. The descriptions of the gadgets are deferred to the full version of the paper. This way, we avoid repetitive constructions and keep the focus on the core ideas.

▶ **Lemma 11.** *Let H be an arbitrary graph, $\ell \geq 1$ be an integer and $R \subseteq \{0, 1\}^\ell$ be a relation that is $(x, |H|)$-regular for some $0 \leq x \leq |H| - 1$. Then, there exists a $(1, H)$-internally coherent gadget G that H-realizes the relation R and the size of G is bounded by some function of ℓ. Moreover, for relations with constant weight, it holds that $\mathbf{pw}(G) = \mathcal{O}(\ell)$.*

When $c \geq 1$, we require the relation to have more structure, i.e., the relation should be $(0, |H|)$-regular. However, this restriction can be easily handled in our lower bound constructions.

▶ **Lemma 12.** *Let $c \geq 1$, H be a clique, $\ell \geq 1$ be a constant and $R \subseteq \{0, \ldots, c\}^\ell$ be a $(0, |H|)$-regular relation. Then, there exists a (c, H)-internally coherent gadget G that strict-(c, H)-realizes the relation R. Moreover, the size of G is bounded by some function of ℓ.*

2.3 Single-Exponential Lower Bound

In this section, we give an overview of how the above-described gadgets are used to prove Theorem 6. Classically, when proving conditional lower bounds based on SETH, one needs to find a reduction from SAT. However, this usually involves carrying out repetitive, unnecessary work that is not specific to the problem one is working on. One of the strengths of the framework introduced by Lampis [7] is removing the need for such repetitive constructions.

The primal graph of a CSP instance ψ is a graph G that has a vertex for each variable of ψ, and there is an edge between $x, y \in V(G)$ if x and y appear together in a constraint. The pathwidth of the CSP instance ψ is defined to be the pathwidth of its primal graph. Similarly, by path decomposition of ψ, we mean a path decomposition of the primal graph of ψ. The following lemma from [7] implies that we can assume the path decomposition to be nice.

▶ **Lemma 13** (Lemma 2.1 in [7], restated for CSPs). *There is a linear-time algorithm that takes as input a CSP formula ψ with n variables and m constraints and a path decomposition of its primal graph of width p and outputs a nice path decomposition $B_1, \ldots, B_t$ of ψ containing at most $t = \mathcal{O}(p \cdot m)$ bags, as well as an injective function b from the set of constraints of ψ to $[t]$ such that for each constraint c, $B_{b(c)}$ contains all the variables of c.*

The following conjecture from [7], called **pw**-SETH, will form the basis of our hardness results.

▶ **Conjecture 14** (Conjecture 1.1 from [7].). *For all $\epsilon > 0$ we have the following: there exists no algorithm which takes as input a $3-SAT$ instance ϕ on n variables and a path decomposition of its primal graph of width* **pw** *and correctly decides if ϕ is satisfiable in time $(2-\epsilon)^{\mathbf{pw}} n^{O(1)}$.*

Moreover, we have the following result from [7], which proves the equivalence of falsifying **pw**-SETH and finding a faster algorithm for 2-CSP.

▶ **Theorem 15** (Theorem 3.2 from [7], shortened). *For each $B \geq 3$ the following statements are equivalent:*

1. *The* **pw**-*SETH is false.*
2. *There exist $\epsilon > 0, b > 0$ and an algorithm that takes as input a 2-CSP instance ψ on alphabet $[B]$, together with a path decomposition of ψ, and decides if ψ is satisfiable in time $(B-\epsilon)^{\mathbf{pw}} \cdot |\psi|^b$.*

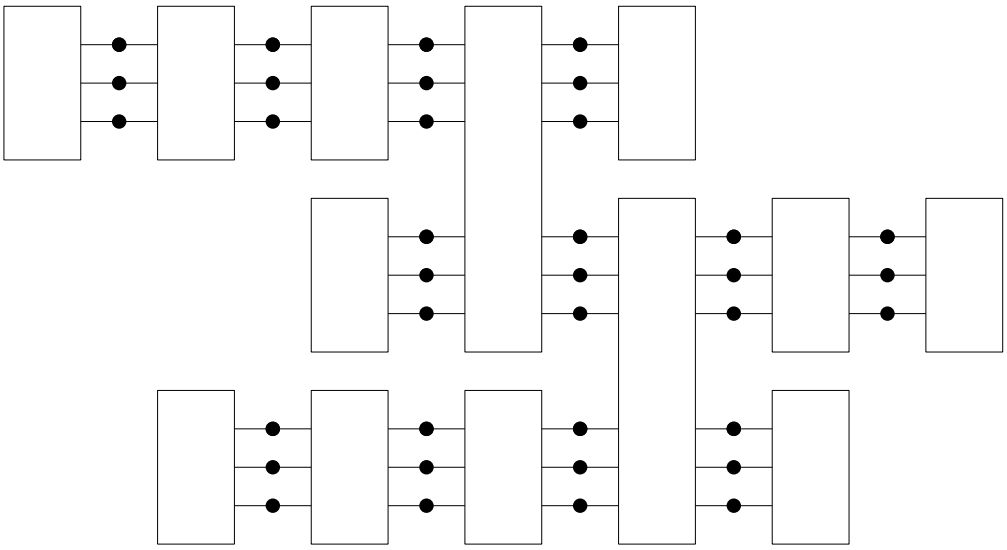


Figure 2.2 A high-level description of the lower bound construction used to prove Theorem 5. The coverage counts of all vertices in a column represent an assignment to the variables of the 2-CSP instance. The rectangles depict the gadgets that enforce the constraints; observe that they interact locally with the vertices, which allows us to bound the pathwidth of the constructed instance.

Motivated by Theorem 15, Section 3 presents a reduction from the 2-CSP problem to the `Single-Clique Partitioning`(c,d) problem. The core idea is to encode each variable's B possible assignments using a set of vertices whose coverage count in a (c, K_d)-single-packing represents the chosen value. We then introduce gadgets each enforcing a single constraint

of the 2-CSP instance via carefully specified relations so that only coverage patterns corresponding to satisfying assignments are possible. By designing these gadgets to interact locally, we ensure the pathwidth of the resulting instance increases by at most a constant, completing the reduction.

2.4 Slightly Superexponential Lower Bound

We now give a high-level description of the proof of Theorem 3. Let H be a graph with at least 3 vertices that is not a block graph. Next, we will describe the lower bound for the `Generalized Graph Partitioning`(H) problem which is stated in Theorem 3. Recall that `Generalized Graph Partitioning`(H) is solvable in slightly super-exponential time for any fixed graph H, i.e., there is an algorithm with running time $\mathcal{O}^*\left(2^{O(tw\cdot\log tw)}\right)$.[1] We show that this running time is optimal under ETH, i.e. there exists no algorithm for `Generalized Graph Partitioning`(H) with running time $\mathcal{O}^*\left(2^{o(tw\cdot\log(tw))}\right)$. To that end, we use an auxiliary problem for which such a lower bound was presented in [4]. Specifically, we define the $k \times k$ Permutation Independent Set problem where given a graph G on a vertex set $[k] \times [k]$, we ask whether there exists an independent set X in G that contains exactly one vertex from each row and each column.

$k \times k$ Permutation Independent Set
Input: A graph G on the vertex set $[k] \times [k]$
Question: Is there an independent set X of G such that X induces a permutation on $[k] \times [k]$?

The following hardness result was presented for the analogous $k \times k$ Permutation Clique problem in [4] which translates easily to $k \times k$ Permutation Independent Set. Below we state it for $k \times k$ Permutation Independent Set.

▶ **Theorem 16** (Theorem 14.14 in [4]). *Unless ETH fails, $k \times k$ Permutation Independent Set cannot be solved in time $2^{o(k\cdot\log k)}$.*

To prove Theorem 3, we reduce the $k \times k$ Permutation Independent Set problem to `Generalized Graph Partitioning`(H). The construction relies on structural properties of H. Let B be a block in H with a minimum separator S such that $B \setminus S$ has at least two connected components. Since H contains a non-clique block, such B and S exist. We partition S into two subsets, U and D, and create k copies of each. The construction in the proof of Theorem 3 ensures that any $|H|$-packing includes k copies of H, each covering exactly one copy of U and one of D. This yields $k! = 2^{\mathcal{O}(k\cdot\log k)}$ configurations, corresponding to all permutations of a set of size k. Additionally, for each edge e in the $k \times k$ Permutation Independent Set instance, we introduce a gadget to prevent e from being included in the set of vertices induced by this permutation. Moreover, these gadgets are designed to preserve pathwidth by interacting only locally. The detailed proof of Theorem 3 is deferred to the full version of the paper.

[1] The $O^*(f(k))$ notation suppresses polynomial factors in the input size n; that is, $O^*(f(k)) = O(f(k)\cdot n^c)$ for some constant c.

2.5 Algorithmic Results

Recall that a tree decomposition of a graph $G = (V, E)$ is a pair $(T, \{X_t\}_{t \in T})$, where T is a tree whose every node t is assigned a bag $X_t \subseteq V(G)$, satisfying the following properties:

1. For every vertex $v \in V$, there exists at least one bag X_t such that $v \in X_t$.
2. For every edge $(u, v) \in E$, there exists a bag X_t such that $\{u, v\} \subseteq X_t$.
3. For every vertex $v \in V$, the set of nodes $\{t \in V(T) \mid v \in X_t\}$ induces a connected subtree of T.

The *width* of a tree decomposition is the size of its largest bag minus one, and the *treewidth* of G is the minimum width over all possible tree decompositions of G. A nice tree decomposition is a rooted tree decomposition in which each node is one of the four types: leaf, introduce, forget or join. We refer the reader to [4] for more details.

The algorithmic results in Theorems 2 and 4 follow a fairly standard dynamic programming framework over tree decompositions. We define a suitable set of states for each bag in the decomposition, capturing the essential information needed to extend partial solutions. The main challenge of the approach lies in carefully designing these states and proving the correctness of the update rules that determine how states transition from one bag to the next. Usually what determines the running time is the state representation and transitions to the specific structural constraints of our problem.

In particular, the single-exponential algorithm for the `Single-Clique Packing`(c, d) problem defines the state of a bag based on how many times each vertex in the bag is covered by a partial solution. This yields $(c+1)^\ell$ states for a bag of size ℓ, naturally leading to a running time of $(c+1)^{tw} \cdot n^{\mathcal{O}(1)}$.

In contrast, the slightly superexponential algorithm for the `Generalized Graph Packing`(H) problem requires keeping track of a partition of the bag, in which each part corresponds to a partial copy of H. This results in $\mathcal{O}(\ell^\ell)$ possible states for a bag of size ℓ, leading to an overall running time of $2^{\mathcal{O}(tw \cdot \log tw)} \cdot n^{\mathcal{O}(1)}$. The algorithms are presented in detail in the full version of the paper.

3 Lower Bounds for Clique Partitioning Problems

In this section we prove Theorems 5 and 6. Theorem 15 says that in order to prove a conditional lower bound based on **pw**-SETH, one can start the reduction from the 2-CSP problem. In the following, we will present a reduction from the 2-CSP problem to the `Multi-Clique Partitioning`(c, d). Subsequently, we will describe another reduction from `Multi-Clique Partitioning`(c, d) to `Single-Clique Partitioning`(c, d) problem, and prove Theorems 5 and 6.

3.1 Lower Bounds

We first prove Theorem 5. To that end, intuitively, we show that a fast algorithm for the `Multi-Clique Partitioning`(c, d) problem implies a fast algorithm for the 2-CSP problem.

▶ **Lemma 17.** *Let $c \geq 1$ and $d \geq 3$ be integers. Suppose there exists an $\varepsilon > 0$ such that* `Multi-Clique Partitioning`(c, d) *can be solved in time $(c + 1 - \varepsilon)^{\mathbf{pw}(G)} \cdot n^{\mathcal{O}(1)}$ for all n-vertex graphs G given together with a path decomposition of width at most $\mathbf{pw}(G)$. Then, there exist $\epsilon', b' > 0$, an integer $B \geq 1$ and an algorithm that takes as input a 2-CSP instance ψ on alphabet $[B]$, together with a path decomposition of ψ, and decides if ψ is satisfiable in time $(B - \epsilon')^{\mathbf{pw}} \cdot |\psi|^{b'}$.*

Proof. Let ε, b be as in the lemma statement and let $\mathcal{A}$ be the hypothetical algorithm for `Multi-Clique Partitioning`(c, d). Moreover, we let H denote the clique K_d and let ℓ be the smallest integer that is a multiple of $|H|$ such that

$$(1-\varepsilon)\cdot|H| < \frac{\varepsilon}{2}\cdot(\ell-|H|)\,. \tag{3.1}$$

Observe that ℓ is a constant that only depends on ε and H.

Let $B = (c+1)^{\ell-|H|}$ and $\varepsilon' = \frac{\varepsilon}{2}$. We will now present a reduction from 2-CSP with alphabet size B to `Multi-Clique Partitioning`(c, d). The idea is as follows: in order to make use of regular relations, we will represent each variable of the 2-CSP instance by ℓ vertices. Note that each of the ℓ vertices can be covered between 0 and c times, which can be thought of as the state of a vertex. In total, ℓ vertices combined give rise to $(c+1)^\ell$ states. These states can also be visualized as vectors $r \in \{0,\ldots,c\}^\ell$. Next, we consider the following subset of vectors

$$Z \coloneqq \{x \in (c+1)^\ell \mid w(x) = 0 \mod |H|\}.$$

Observe that we have $|Z| \geq (c+1)^{\ell-|H|} = B$, because we can append to each vector in $r' \in \{0,\ldots,c\}^{\ell-|H|}$ at most $|H|$ many 1's so that the new vector $r \in \{0,\ldots,c\}^\ell$ constructed this way satisfies $w(r) = 0 \mod |H|$. Hence there exists an injective map $\Phi\colon [B] \to Z$ where we define $W \coloneqq \operatorname{im}(\Phi)$. We use the set W to simulate B many assignments to a variable of the 2-CSP instance. Observe that W, as a relation, is $(0, |H|)$-regular.

Construction of the `Multi-Clique Partitioning`(c, d) instance. Let ψ be a 2-CSP instance with variables $x_1,\ldots,x_n$, contraints $C_1,\ldots,C_m$ and alphabet $[D]$. Let $\mathcal{P} = (B_1,\ldots,B_t)$ be a nice path decomposition of width p. Finally, let $b\colon [m] \to [t]$ be a function that maps each constraint C_i of ψ to a bag such that $B_{b(i)}$ contains the variables occurring in C_i. We will construct an instance of the `Multi-Clique Partitioning`(c, d) problem as follows:

1. For each $1 \leq i \leq n$, define $l(i) \in [t]$ to be the smallest integer such that $x_i \in B_{l(i)}$. Similarly, let $r(i)$ be the largest integer such that $x_i \in B_{r(i)}$. For each $i \in [n]$ and $l(i) \leq j \leq r(i)+1$, introduce ℓ vertices $\{a_1^{i,j},\ldots,a_\ell^{i,j}\}$.
2. Define the relation

   $$W^C \coloneqq \operatorname{Compl}_c\big(W\big).$$

   Observe that W is $(0, |H|)$-regular by construction. The same also holds for W^C, because for each $x \in W^C$ such that $x = \operatorname{Compl}_c\big(s\big)$ for $s \in W$, we have

   $$w(x) = \Big(\ell\cdot c - w(s)\Big) = 0 \mod |H|,$$

   where the last equivalence holds because ℓ and $w(s)$ are both equivalent to $0 \mod |H|$. Introduce two gadgets L^i and R^i that arb-(c, H)-realize the relation W^C and W, respectively, which exist by Lemma 12. Then, identify the portal vertices of L^i with $(a_1^{i,l(i)},\ldots,a_\ell^{i,l(i)})$, and similarly, identify the portal vertices of R^i with the vertices $\{a_1^{i,r(i)+1},\ldots,a_\ell^{i,r(i)+1}\}$.
3. Let $j \in [t]$. We say that j represents s for $s \in [m]$ if $b(s) = j$. In that case, we define S_j to be the relation, and i_1 and i_2 to be the indices of the variables associated with the constraint C_s. We define the new relation

   $$R_j \coloneqq \left\{\Phi(u_1)\odot\Phi(u_2)\odot\operatorname{Compl}_c\big(\Phi(u_1)\big)\odot\operatorname{Compl}_c\big(\Phi(u_2)\big) \;\middle|\; (u_1,u_2)\in S_j\right\}.$$

Observe that R_j is $(0, |H|)$-regular because for each $x \in R_j$ we have

$$w(x) = (2 \cdot \ell \cdot c) = 0 \mod |H|$$

since ℓ is a multiple of $|H|$. Hence, we can introduce a gadget $N^j_{i_1,i_2}$ by Lemma 12 that arb-(c, H)-realizes R_j. Then, we identify the portal vertices of $N^j_{i_1,i_2}$ with the vertices

$$\left(a_1^{i_1,j}, \ldots, a_\ell^{i_1,j}\right) \odot \left(a_1^{i_2,j}, \ldots, a_\ell^{i_2,j}\right) \odot \left(a_1^{i_1,j+1}, \ldots, a_\ell^{i_1,j+1}\right) \odot \left(a_1^{i_2,j+1}, \ldots, a_\ell^{i_2,j+1}\right). \quad (3.2)$$

Next, we define the relation COPY $\subseteq \{0, \ldots, c\}^{2\cdot\ell}$ where

$$\text{COPY} \coloneqq \{\boldsymbol{u} \odot \text{Compl}_c(\boldsymbol{u}) \mid \boldsymbol{u} \in W\} \quad (3.3)$$

and let

$$\Gamma_j \coloneqq \begin{cases} \{i_1, i_2\} & \text{if } j \text{ represents } s, \\ \emptyset & \text{otherwise.} \end{cases}$$

Observe that COPY is $(0, |H|)$-regular because for each $x \in$ COPY we have

$$w(x) = \ell \cdot c = 0 \mod |H|.$$

By Lemma 12, there exists a gadget that arb-(c, H)-realizes COPY. Next, for each $i \in \left([n] \setminus \Gamma_j\right)$ such that $l(i) \leq j \leq r(i)$, we introduce a gadget F_i^j that arb-(c, H)-realizes the relation COPY and identifies the portal vertices of F_i^j with

$$\left(a_1^{i,j}, \ldots, a_\ell^{i,j}\right) \odot \left(a_1^{i,j+1}, \ldots, a_\ell^{i,j+1}\right). \quad (3.4)$$

Finally, for each $i \in [n]$ and $j \in [t]$ such that $l(i) \leq j \leq r(i)$, we define the gadget that covers i at step j as

$$K_i^j \coloneqq \begin{cases} N^j_{i_1,i_2} & \text{if } j \text{ represents } s \in [m] \text{ and } i \in \{i_1, i_2\} \\ F_i^j & \text{otherwise.} \end{cases}$$

This is the whole construction of the `Multi-Clique Partitioning`(c, d) instance which we call G.

Equivalence of the instances. We now prove that ψ is satisfiable if and only if G admits a $\left(c, K_d\right)$-multi-partition, by establishing each direction separately.

Suppose that there exists an assignment $(\alpha_1, \ldots, \alpha_n)$ to $(x_1, \ldots, x_n)$ such that ψ is satisfied. In the following, we will describe a $\left(c, K_d\right)$-multi-partition $\mathcal{K}$ of G. For each $i \in [n]$, define $\boldsymbol{a}_i \coloneqq \Phi(\alpha_i) \in W$. Now let $j \in [t]$. Observe that since $\boldsymbol{a}_i \in W$ for each $i \in [n]$, it holds that there exists a $\left(c, K_d\right)$-multi-packing of L^i that covers $(a_1^{i,l(i)}, \ldots, a_\ell^{i,l(i)})$ according to $\text{Compl}_c(\boldsymbol{a}_i)$. Moreover, there exists a $\left(c, K_d\right)$-multi-packing of R^i that covers $(a_1^{i,r(i)+1}, \ldots, a_\ell^{i,l(i)+1})$ according to $\boldsymbol{a}_i$. In both cases, the internal vertices of the gadgets are covered exactly c times.

Now let $j \in [t]$. For all $i \in \left([n] \setminus \Gamma_j\right)$, there exists a $\left(c, K_d\right)$-multi-packing of F_i^j that covers

$$\left(a_1^{i,j}, \ldots, a_\ell^{i,j}\right) \odot \left(a_1^{i,j+1}, \ldots, a_\ell^{i,j+1}\right) \quad (3.5)$$

according to $\boldsymbol{a}_i \odot \mathrm{Compl}_c(\boldsymbol{a}_i)$, because $\boldsymbol{a}_i \in W$. Moreover, if j represents s for some $s \in [m]$, and x_{i_1}, x_{i_2} are the variables corresponding to C_s, then there exists a (c, K_d)-multi-packing that covers

$$\left(a_1^{i_1,j}, \ldots, a_\ell^{i_1,j}\right) \odot \left(a_1^{i_2,j}, \ldots, a_\ell^{i_2,j}\right) \odot \left(a_1^{i_1,j+1}, \ldots, a_\ell^{i_1,j+1}\right) \odot \left(a_1^{i_2,j+1}, \ldots, a_\ell^{i_2,j+1}\right) \quad (3.6)$$

according to $\boldsymbol{a}_{i_1} \odot \boldsymbol{a}_{i_2} \odot \mathrm{Compl}_c(\boldsymbol{a}_{i_1}) \odot \mathrm{Compl}_c(\boldsymbol{a}_{i_2})$. Again, in both cases, the internal vertices of the gadgets are covered exactly c times. Next, we prove that the remaining vertices are covered c times as well.

▷ Claim 18. It holds that for each $i \in [n]$, $j \in [t]$ such that $l(i) \leq j \leq r(i) + 1$ and $x \in [\ell]$, $a_x^{i,j}$ is covered exactly c times by $\mathcal{K}$.

Proof. We prove the claim by induction on j. Let $j = 1$, $i \in [n]$ and suppose that $l(i) \leq j \leq r(i) + 1$. Since $1 \leq l(i) \leq j = 1$, we have that $l(i) = 1 = j$. Consider the vertices $\left(a_1^{i,1}, \ldots, a_\ell^{i,1}\right)$, which are covered according to $\mathrm{Compl}_c(\boldsymbol{a}_i)$ by L^i. Moreover, $\left(a_1^{i,1}, \ldots, a_\ell^{i,1}\right)$ is covered according to $\boldsymbol{a}_i$ by K_i^1 which follows from (3.5) or (3.6), depending on whether $i \in \Gamma_1$ or not, respectively. All in all, $a_x^{i,1}$ is covered exactly c times for $x \in [\ell]$.

Now suppose that the claim holds for $1 < j \leq t$, and we will prove the claim for $j + 1$. Let $i \in [n]$ such that $l(i) \leq j + 1 \leq r(i) + 1$. Consider the vertices $\left(a_1^{i,j+1}, \ldots, a_\ell^{i,j+1}\right)$, which are covered according to $\mathrm{Compl}_c(\boldsymbol{a}_i)$ by K_i^j. This follows from (3.5) and (3.6). Now, observe that we have either $j < r(i)$ or $j = r(i)$. In both cases, by using the arguments in the $j = 1$ case, one can conclude that $\left(a_1^{i,j+1}, \ldots, a_\ell^{i,j+1}\right)$ is covered according to $\boldsymbol{a}_i$ by K_i^{j+1}. Therefore, all in all, it holds that each vertex $a_x^{i,j}$ is covered exactly c times for $x \in [\ell]$. ◁

Now for the reverse implication, suppose that G has a (c, K_d)-multi-partition. Since each gadget used in the construction of G is (c, K_d)-internally coherent, this implies that for each gadget there is a (c, K_d)-multi-packing that covers its interval vertices exactly c times, and its portal vertices according to the relation associated with it. In particular, for each $i \in [n]$, let $\boldsymbol{b}_i \in W^C$ denote the vector such that L^i covers the vertices $\{a_1^{i,l(i)}, \ldots, a_\ell^{i,l(i)}\}$ according to $\boldsymbol{b}_i$.

By induction, one can show that for each $j \in [t]$ and $i \in [n]$ such that $l(i) \leq j \leq r(i)$, the tuple $(a_1^{i,j}, \ldots, a_\ell^{i,j})$ is covered by K_i^j according to $\boldsymbol{z}_i$ where $\boldsymbol{z}_i = \mathrm{Compl}_c(\boldsymbol{b}_i)$. Hence, we let $\alpha_i \coloneqq \Phi^{-1}(\boldsymbol{z}_i) \in B$. Next, we will prove that $A \coloneqq (\alpha_1, \ldots, \alpha_n)$ is a satisfying assignment for ψ. To that end, let $s \in [m]$ and $j = b(s)$. To show that C_s is satisfied by A, let x_{i_1} and x_{i_2} be the variables associated with C_s. By the above discussion, we know that $\left(a_1^{i_1,j}, \ldots, a_\ell^{i_1,l}\right)$ and $\left(a_1^{i_2,j}, \ldots, a_\ell^{i_2,j}\right)$ is covered according to $\boldsymbol{z}_{i_1}$ and $\boldsymbol{z}_{i_2}$, respectively. By the definition of R_j, $a_{i_1} = \Phi^{-1}(\boldsymbol{z}_{i_1})$ and $a_{i_2} = \Phi^{-1}(\boldsymbol{z}_{i_2})$ satisfy C_s. Therefore, the assignment A satisfies all constraints of ψ, and ψ is satisfied if and only if G has a (c, K_d)-multi-partition.

Pathwidth and size of the constructed instance. To bound the pathwidth of G, we will create a path decomposition by following $\{B_j\}_{j \in [t]}$. Specifically, for each $j \in [t]$, we first create a new path decomposition $\mathcal{X} = (X_1, \ldots, X_t)$ where each X_j is a copy of B_j and we replace each $x_i \in B$ with the vertices $a_1^{i,j}, \ldots, a_\ell^{i,j}$. Note that the size of each X_j is at most $p \cdot \ell$ for $j \in [t]$.

In the following, we will add the remaining vertices of G to the bags in $\mathcal{X}$ such that $\mathcal{X}$ is a valid path decomposition of G. Let $j \in [t]$. For each $i \in [n] \setminus \Gamma_j$ such that $l(i) \leq j \leq r(i)$, we replace the vertices $\{a_x^{i,j}\}_{x \in [\ell]}$ with $\{a_x^{i,j+1}\}_{x \in [\ell]}$ as follows. After the bag X_j ,we first insert the bag $X'_{j,i} \coloneqq \left(X_j \cup V\left(F_i^j\right)\right)$, and then add another bag $X''_{j,i}$ where we replace

$\{a_x^{i,j}\}_{x\in[\ell]}$ in $X'_{j,i}$ with $\{a_x^{i,j+1}\}_{x\in[\ell]}$, and finally, another bag $X'''_{j,i}$ where we remove the vertices $V\left(F_i^j\right)$ from $X''_{j,i}$. For a fixed $j \in [t]$, we keep adding the bags iteratively for each $i \in [n] \setminus \Gamma_j$ such that $l(i) \leq j \leq r(i)$, until all vertices in all the gadgets are contained in the bag decomposition. Note that, by our construction, edges that are adjacent to a vertex in F_i^j are also covered by either $X'_{j,i}$ or $X''_{j,i}$.

Finally, for each $s \in [m]$ and $j = b(s)$, we add the gadgets $N^j_{i_1,i_2}$ to the tree decomposition in a similar way. Since the size of each $N^j_{i_1,i_2}$ and F_i^j is a function of ℓ, it is bounded by a constant. All in all, the pathwidth of G increases at most by a constant. We have

$$\mathbf{pw}(G) = l \cdot p + \mathcal{O}(1).$$

Finally, since t and m are bounded by a polynomial of n, it follows from the construction that

$$V(G) = n^{\mathcal{O}(1)}.$$

Running Time. Constructing the graph G from ψ takes time polynomial in n. By our assumption on $\mathcal{A}$, the whole reduction takes time

$$\begin{aligned}
(c+1)^{(1-\varepsilon)\cdot\mathbf{pw}(G)} \cdot V(G)^b &= (c+1)^{(1-\varepsilon)\cdot\ell\cdot p} \cdot n^{\mathcal{O}(1)} \\
&= (c+1)^{(1-\varepsilon)\cdot(\ell-|H|)\cdot p} \cdot (c+1)^{(1-\varepsilon)\cdot|H|\cdot p} \cdot n^{\mathcal{O}(1)} \\
&= B^{(1-\varepsilon)\cdot p} \cdot (c+1)^{(1-\varepsilon)\cdot|H|\cdot p} \cdot n^{\mathcal{O}(1)} \\
&< B^{(1-\varepsilon)\cdot p} \cdot (c+1)^{\frac{\varepsilon}{2}\cdot(\ell-|H|)\cdot p} \cdot n^{\mathcal{O}(1)} \\
&= B^{(1-\varepsilon)\cdot p} \cdot B^{\frac{\varepsilon}{2}\cdot p} \cdot n^{\mathcal{O}(1)} \\
&= B^{(1-\varepsilon')\cdot p} \cdot |\psi|^{b'}
\end{aligned}$$

where the inequality follows from (3.1) and b' is a constant. ◀

The proof of Theorem 5 follows from Theorem 15 and Lemma 17. We note here that the same construction can be used to prove Theorem 6, because the gadgets used in Lemma 17 strict-(c, K_d)-realize their relations (see Definition 9 and Lemma 12). However, by presenting a simple reduction, we demonstrate that a fast algorithm for `Single-Clique Partitioning`(c, d) implies a fast algorithm for `Multi-Clique Partitioning`(c, d), which is used to prove Theorem 6 in a more formal way.

▶ **Lemma 19.** *Let $c \geq 1$ and $d \geq 3$ be integers. Suppose there exists an $\varepsilon > 0$ such that* `Single-Clique Partitioning`(c, d) *can be solved in time $(c + 1 - \varepsilon)^{\mathbf{pw}(G)} \cdot n^{\mathcal{O}(1)}$ for all n-vertex graphs G given together with a path decomposition of width at most $\mathbf{pw}(G)$. Then, there exist $\epsilon' > 0$ and $b' > 0$ such that* `Multi-Clique Partitioning`(c, d) *can be solved in time $(c+1-\varepsilon')^{\mathbf{pw}(G)} \cdot n^{b'}$ for all n-vertex graphs G given together with a path decomposition of width at most $\mathbf{pw}(G)$.*

Proof. Let ε and b be as in the statement of the lemma. Moreover, let $\mathcal{A}$ be the hypothetical algorithm for `Single-Clique Partitioning`(c, d).

Construction of the `Single-Clique Partitioning`(c, d) instance. Let G be an instance of `Multi-Clique Partitioning`(c, d). Moreover, we let H denote the clique K_d and construct a `Single-Clique Partitioning`(c, d) instance G' as follows. Let G' have the same vertex set as G. We call these vertices the original vertices of G'. Moreover, for each clique X in G of size d, we add a gadget E_X that dist-(c, H)-realizes the relation $\mathrm{EQ}_d^{[0,c]}$. We identify the portal vertices of E_X with $V(X)$. This is the whole construction.

Equivalence of Instances. We will now prove that G admits a $\big(c, K_d\big)$-multi-partition if and only if G' admits a (c, K_d)-single-partition.

Suppose that G has a $\big(c, K_d\big)$-multi-partition which is denoted by $\mathcal{Z}$. For each clique X in G, let a_x denote the number of occurences of X in $\mathcal{Z}$. We construct a (c, K_d)-single-partition $\mathcal{K}$ for G' as follows. For each clique X in G, consider a (c, K_d)-single-packing of E_X such that the portal vertices of E_X are covered a_X times. Add this (c, K_d)-single-packing to $\mathcal{K}$. Since the gadgets are disjoint, the copies of K_d are also disjoint. The internal vertices of the gadgets are covered exactly c times. Moreover, the original vertices of G' are also covered c times, since each equality gadget simulates a clique. Hence $\mathcal{K}$ is a valid (c, K_d)-single-partition and therefore G' is a yes instance.

Now suppose that G' has a (c, K_d)-single-partition $\mathcal{K}$. Then, for each clique X in G, let a_X denote the number of times E_X covers its portal vertices. We construct a $\big(c, K_d\big)$-multi-packing $\mathcal{Z}$ by including each X exactly a_X times in $\mathcal{Z}$. Moreover, $\mathcal{Z}$ covers each vertex exactly c times, hence it is a valid $\big(c, K_d\big)$-multi-partition. Therefore, G is a yes-instance.

Running Time. First, we show that the pathwidth of G' is $p + \mathcal{O}(1)$. Take the path decomposition of G and for each clique X, let B_X denote the bag that contains the vertices of X. Note that since X is a clique, there exists such a bag. Moreover, we may assume each bag B_X is unique for each clique X, duplicating bags if necessary. Then, we simply add the vertices of E_X to the bag B_X, increasing its size at most by a constant. Therefore, we get a new path decomposition for G' where the size of a bag is at most $p + \mathcal{O}(1)$. Hence, it holds that $\mathbf{pw}(G') = p + \mathcal{O}(1)$.

Constructing the graph takes time polynomial in $n = V(G)$, and we also have $V(G') = n^{\mathcal{O}(1)}$. Therefore, the whole reduction takes time

$$\begin{aligned}(c+1-\varepsilon)^{\mathbf{pw}(G')} \cdot V(G')^b &= (c+1-\varepsilon)^{p+\mathcal{O}(1)} \cdot n^{\mathcal{O}(1)} \\ &= (c+1-\varepsilon)^{p} \cdot n^{\mathcal{O}(1)} \\ &= (c+1-\varepsilon)^{\mathbf{pw}(G)} \cdot n^{b'}\end{aligned}$$

where b' is a constant. ◀

The proof of Theorem 6 follows from Theorem 5 and Lemma 19.

References

1 Barış Can Esmer, Jacob Focke, Dániel Marx, and Paweł Rzążewski. Fundamental Problems on Bounded-Treewidth Graphs: The Real Source of Hardness. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:17, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2024.34.

2 Barış Can Esmer, Jacob Focke, Dániel Marx, and Paweł Rzążewski. List Homomorphisms by Deleting Edges and Vertices: Tight Complexity Bounds for Bounded-Treewidth Graphs. In Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms (ESA 2024)*, volume 308 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2024.39.

3 Radu Curticapean and D. Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In *SODA*, 2016. doi:10.1137/1.9781611974331.CH113.

4 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-21275-3.

5 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight Complexity Bounds for Counting Generalized Dominating Sets in Bounded-Treewidth Graphs Part II: Hardness Results. *ACM Transactions on Computation Theory*, page 3708509, January 2025. doi:10.1145/3708509.

6 P. Hell and D. G. Kirkpatrick. On generalized matching problems. *Information Processing Letters*, 12(1):33–35, February 1981. doi:10.1016/0020-0190(81)90073-9.

7 Michael Lampis. The Primal Pathwidth SETH. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 1494–1564. Society for Industrial and Applied Mathematics, January 2025. doi:10.1137/1.9781611978322.47.

8 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known Algorithms on Graphs of Bounded Treewidth Are Probably Optimal. *ACM Transactions on Algorithms*, 14(2):1–30, June 2018. doi:10.1145/3170442.

9 Karolina Okrasa, Marta Piecyk, and Pawel Rzążewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.74.

10 Karolina Okrasa and Pawel Rzążewski. Fine-grained complexity of the graph homomorphism problem for bounded-treewidth graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. doi:10.1137/20M1320146.

Optimal Quantum Algorithm for Estimating Fidelity to a Pure State

Wang Fang
School of Informatics, University of Edinburgh, UK

Qisheng Wang
School of Informatics, University of Edinburgh, UK

Abstract

We present an *optimal* quantum algorithm for fidelity estimation between two quantum states when one of them is pure. In particular, the (square root) fidelity of a mixed state to a pure state can be estimated to within additive error ε by using $\Theta(1/\varepsilon)$ queries to their state-preparation circuits, achieving a quadratic speedup over the folklore $O(1/\varepsilon^2)$. Our approach is technically simple, and can moreover estimate the quantity $\sqrt{\mathrm{tr}(\rho\sigma^2)}$ that is not common in the literature. To the best of our knowledge, this is the *first* query-optimal approach to fidelity estimation involving mixed states.

2012 ACM Subject Classification Theory of computation → Quantum query complexity; Theory of computation → Quantum information theory

Keywords and phrases Quantum computing, fidelity estimation, quantum algorithms, quantum query complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.4

Related Version *Full Version*: `https://arxiv.org/abs/2506.23650` [6]

Funding *Wang Fang*: Supported by the Engineering and Physical Sciences Research Council under Grant EP/X025551/1.
Qisheng Wang: Supported by the Engineering and Physical Sciences Research Council under Grant EP/X026167/1.

1 Introduction

The fidelity between quantum states [24, 13] is a closeness measure that is commonly used in quantum physics and quantum computing [20, 34, 12, 33]. Formally, for two (mixed) quantum states ρ and σ, their (square root) fidelity is defined by (see [20, Equation (9.53)])

$$\mathrm{F}(\rho,\sigma) = \mathrm{tr}\left(\sqrt{\sqrt{\sigma}\rho\sqrt{\sigma}}\right).$$

The fidelity is generally bounded by $0 \leq \mathrm{F}(\rho,\sigma) \leq 1$. In particular, when $\mathrm{F}(\rho,\sigma) = 1$, the two states ρ and σ are identical; and when $\mathrm{F}(\rho,\sigma) = 0$, the two states are orthogonal.

The estimation of fidelity turns out to be central in quantum property testing (cf. [18]). The earliest approach is now known as the SWAP test [5], allowing us to estimate to within additive error ε the squared fidelity $\mathrm{F}^2(\rho,\sigma) = \mathrm{tr}(\rho\sigma)$ when ρ and σ are pure states, using $O(1/\varepsilon^2)$ samples of ρ and σ. Moreover, the squared fidelity $\mathrm{F}^2(\rho,\sigma)$ for pure states ρ and σ can be estimated using $O(1/\varepsilon)$ queries to their state-preparation circuits through the SWAP test [5] equipped with quantum amplitude estimation [4]. The approach based on the SWAP test has been found to have various applications [21]. Fidelity estimation for pure states was also considered in some restricted models: a direct fidelity estimation [7] was proposed when only Pauli measurements are allowed and a distributed approach was developed in [1]. Recently, query-optimal and sample-optimal quantum algorithms for estimating the fidelity $\mathrm{F}(\rho,\sigma) = \sqrt{\mathrm{tr}(\rho\sigma)}$ for pure states ρ and σ have been found in [25] and [28], respectively.


© Wang Fang and Qisheng Wang;
licensed under Creative Commons License CC-BY 4.0

Table 1 Complexity of estimating the fidelity $\mathrm{F}(\rho,\sigma)$.

Reference	Condition	Complexity	Notes
[5]	One of ρ and σ is pure	$O(1/\varepsilon^4)$ samples	/
[5, 4]		$O(1/\varepsilon^2)$ queries	/
This Work		$\mathbf{\Theta(1/\varepsilon)}$ **queries**	**Optimal**
[25]	Both ρ and σ are pure	$\Theta(1/\varepsilon)$ queries	Optimal
[28]		$\Theta(1/\varepsilon^2)$ samples	Optimal
[29, 26, 10]	Either ρ or σ is of rank r	$\mathrm{poly}(r,1/\varepsilon)$ queries	/
[10]		$\mathrm{poly}(r,1/\varepsilon)$ samples	/
[15]	$\rho,\sigma \geq I/\kappa$	$\mathrm{poly}(\kappa)\cdot\widetilde{\Theta}(1/\varepsilon)$ queries	Almost optimal in ε
		$\mathrm{poly}(\kappa,1/\varepsilon)$ samples	/

Although fidelity estimation is known to be BQP-complete when one of the quantum states is pure [22], fidelity estimation for mixed states is QSZK-hard in general [31, 32], which means that there is no polynomial-time quantum algorithm for fidelity estimation unless BQP = QSZK. Nevertheless, efficient quantum algorithms for fidelity estimation for low-rank states were recently discovered [29] and later improved [26, 10] with time complexity $\mathrm{poly}(r,1/\varepsilon)$, where r is the rank of the states. Quantum algorithms for fidelity estimation for well-conditioned states was recently proposed with query complexity $\mathrm{poly}(\kappa)\cdot\widetilde{O}(1/\varepsilon)$ [15], achieving almost optimal dependence on the precision ε, where κ is the condition number such that $\rho,\sigma \geq I/\kappa$.

In this paper, we present an optimal quantum algorithm for estimating the fidelity of a mixed state ρ to a pure state $\sigma = |\psi\rangle\langle\psi|$, using queries to the state-preparation circuits of both states. The input model is known as the "*purified quantum query access*" model, where we assume query access to (the controlled version of) a quantum unitary oracle that prepares the purification of the input quantum state, and its inverse. This input model is commonly employed in quantum computational complexity [31] and quantum algorithms [8]. Our main result is stated in the following theorem.

▶ **Theorem 1** (Estimating fidelity to a pure state, Theorem 6 simplified)**.** *Given purified quantum query access to a mixed state ρ and a pure state $\sigma = |\psi\rangle\langle\psi|$, the fidelity $\mathrm{F}(\rho,\sigma) = \sqrt{\langle\psi|\rho|\psi\rangle}$ can be estimated to within additive error ε with query complexity $O(1/\varepsilon)$.*

Prior to the result of Theorem 1, we are only aware of a folklore quantum algorithm based on the SWAP test (which was mentioned above and will be explained in detail in Section 3.1). This folklore approach computes $\mathrm{F}^2(\rho,\sigma) = \langle\psi|\rho|\psi\rangle$ to within additive error ε with query complexity $O(1/\varepsilon)$, thereby resulting in a query complexity of $O(1/\varepsilon^2)$ for estimating $\mathrm{F}(|\psi\rangle,\rho)$ to within additive error ε. By comparison, Theorem 1 exhibits a quadratic improvement.

The quantum query algorithm given in Theorem 1 generalizes the pure-state fidelity estimation in [25] where unitary oracles are required to directly prepare the pure states. In comparison, our result in Theorem 1 allows the unitary oracles to have redundant output qubits (see Section 5 for more details).

Our approach is actually optimal, as it can be used to estimate the fidelity between two pure states and the quantum query lower bound $\Omega(1/\varepsilon)$ is due to [2, 19] (noted in [25]). In Section 6, we further show that the optimality holds even if ρ is of an arbitrary rank.

We compare fidelity estimations for pure states in Table 1.

As a bonus, our approach in Theorem 1 can further estimate the quantity $\sqrt{\mathrm{tr}(\rho\sigma^2)}$ with optimal query complexity (see Section 4.2), which is of independent interest as this quantity is not common in the literature.

Organization of this paper. We give a formal definition of the purified quantum query access model and the problem statement of fidelity estimation in Section 2. Then, we review previous approaches in Section 3 with their subroutines. In Section 4, we present our approach and its generalization. An implication of our results is discussed in Section 5. Lower bounds for fidelity estimation are given in Section 6. Finally, a brief discussion is drawn in Section 7.

2 Problem Settings

In this paper, we assume purified quantum query access to the input quantum states. This input model is widely used in the literature, e.g., [8, 23, 11, 10, 30, 27, 16].

▶ **Definition 2** (Purified quantum query access). *For a mixed quantum state ρ, purified quantum query access to ρ means query access to a unitary oracle U that prepares a purification of ρ. Specifically, suppose U_{AB} acts on two subsystems A and B, then $\rho_{\mathsf{A}} = \mathrm{tr}_{\mathsf{B}}(|\rho\rangle\langle\rho|_{\mathsf{AB}})$, where $|\rho\rangle_{\mathsf{AB}} = U_{\mathsf{AB}}|0\rangle_{\mathsf{AB}}$. Moreover, we are allowed to use queries to (controlled-)U and its inverse.*

We consider the problem of fidelity estimation, formally stated as follows.

▶ **Problem 1** (Fidelity estimation). *Given purified quantum query access to two mixed quantum states ρ and σ, the task is to compute $\mathrm{F}(\rho,\sigma)$ to within additive error ε.*

In particular, this paper focuses on the case of Problem 1 where $\sigma = |\psi\rangle\langle\psi|$ is pure.

3 Warm-Ups

As a warm-up, we review the previous approaches to fidelity estimation involving pure states, together with their useful subroutines.

3.1 SWAP Test

It is well known that the SWAP test [5] can be used to estimate the fidelity $\mathrm{F}(\rho,\sigma)$ when one of ρ and σ is pure. In particular, if $\sigma = |\psi\rangle\langle\psi|$ is pure, the SWAP test on input ρ and σ (see Figure 1) outputs a bit $x \in \{0,1\}$ such that (adapted from [14, Proposition 9])

$$\Pr[x=0] = \frac{1+\mathrm{F}^2(\rho,|\psi\rangle)}{2}. \tag{1}$$

With this, we can estimate $\mathrm{F}(\rho,|\psi\rangle)$. Specifically, suppose that $\tilde{p}$ is an estimate of $\Pr[x=0]$ such that $|\tilde{p}-\Pr[x=0]| \le \delta$. Then, it can be shown that $\sqrt{2\tilde{p}-1}$ is an estimate of $\mathrm{F}(\rho,|\psi\rangle)$ with $|\sqrt{2\tilde{p}-1} - \mathrm{F}(\rho,|\psi\rangle)| \le \Theta(\sqrt{\delta})$. An estimate of $\mathrm{F}(\rho,|\psi\rangle)$ to within additive error ε can be obtained by setting $\delta = \Theta(\varepsilon^2)$.

To estimate $\mathrm{F}(\rho,|\psi\rangle)$ when given purified quantum query access to ρ and $|\psi\rangle$, we need the quantum subroutine for amplitude estimation [4].

▶ **Theorem 3** (Quantum amplitude estimation, [4]). *Suppose that U is a unitary operator such that $U|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}} = \sqrt{p}|0\rangle_{\mathsf{A}}|\varphi_0\rangle_{\mathsf{B}} + \sqrt{1-p}|1\rangle_{\mathsf{A}}|\varphi_1\rangle_{\mathsf{B}}$, where $p \in [0,1]$ and $|\varphi_0\rangle, |\varphi_1\rangle$ are normalized pure states. Then, we can estimate p to within additive error δ using $O(1/\delta)$ queries to (controlled-)U and its inverse.*

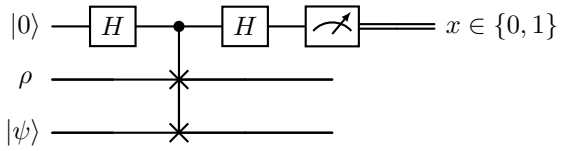


Figure 1 The SWAP test for estimating $\mathrm{F}(\rho, |\psi\rangle)$.

By Theorem 3, we can therefore obtain an estimate of $\mathrm{F}(\rho, |\psi\rangle)$ to within additive error ε using $O(1/\varepsilon^2)$ queries to the state-preparation circuits of ρ and $|\psi\rangle$, based on the SWAP test in Figure 1.

3.2 Pure-State Fidelity Estimation

Recently, a new approach was proposed in [25] for estimating the fidelity $\mathrm{F}(|\varphi\rangle, |\psi\rangle)$ between two pure quantum states $|\varphi\rangle$ and $|\psi\rangle$, improving the folklore approach in Section 3.1. However, the approach in [25] assumes a slightly more restricted input model, where U_φ and U_ψ are two unitary quantum circuits that respectively prepare $|\varphi\rangle$ and $|\psi\rangle$, i.e., $U_\varphi|0\rangle = |\varphi\rangle$ and $U_\psi|0\rangle = |\psi\rangle$.

The key idea of [25] is to *encode* the value of $\mathrm{F}(|\varphi\rangle, |\psi\rangle)$ in the amplitudes of a quantum state, rather than the $\sqrt{(1+\mathrm{F}^2(|\varphi\rangle, |\psi\rangle))/2}$ (in Equation (1)). Specifically, $\mathrm{F}(|\varphi\rangle, |\psi\rangle)$ can be encoded by the unitary operator $W = U_\varphi^\dagger U_\psi$ such that

$$W|0\rangle = \mathrm{F}(|\varphi\rangle, |\psi\rangle)|0\rangle + |\perp\rangle, \tag{2}$$

where $|\perp\rangle$ is an (unnormalized) pure state that is orthogonal to $|0\rangle$. The circuit W is depicted in Figure 2 and it outputs a bit $x \in \{0,1\}$ such that

$$\Pr[x = 0] = \mathrm{F}^2(|\varphi\rangle, |\psi\rangle). \tag{3}$$

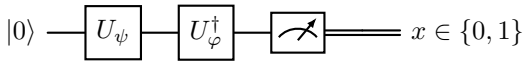


Figure 2 The quantum circuit for estimating $\mathrm{F}(|\varphi\rangle, |\psi\rangle)$.

Finally, the value of $\mathrm{F}(|\varphi\rangle, |\psi\rangle)$ can be estimated by the square root amplitude estimation provided in [25].

▶ **Theorem 4** (Quantum square root amplitude estimation, [25, Theorem I.5]). *Suppose that U is a unitary operator such that $U|0\rangle_\mathsf{A}|0\rangle_\mathsf{B} = \sqrt{p}|0\rangle_\mathsf{A}|\varphi_0\rangle_\mathsf{B} + \sqrt{1-p}|1\rangle_\mathsf{A}|\varphi_1\rangle_\mathsf{B}$, where $p \in [0,1]$ and $|\varphi_0\rangle, |\varphi_1\rangle$ are normalized pure states. Then, we can estimate $\sqrt{p}$ to within additive error δ using $O(1/\delta)$ queries to (controlled-)U and its inverse.*

For convenience, throughout this paper, we use $\mathsf{SqrtAmpEst}(U, \varepsilon)$ *to denote (the returned value of) the square root amplitude estimation process.*

Compared to Theorem 3, Theorem 4 gives an estimate of $\sqrt{p}$ (instead of p), thereby allowing us to skip taking square roots. By Theorem 4, we can therefore obtain an estimate of $\mathrm{F}(|\varphi\rangle, |\psi\rangle)$ to within additive error ε using $O(1/\varepsilon)$ queries to the state-preparation circuits of $|\varphi\rangle$ and $|\psi\rangle$, based on the construction of W in Equation (2).

4 Our Approach

For the task of estimating the fidelity $\mathrm{F}(\rho, |\psi\rangle)$ of a mixed state to a pure state, the approach in Section 3.1 only gives a query complexity of $O(1/\varepsilon^2)$, while the approach in Section 3.2, however, turns out not to be applicable (as a more restricted input model is assumed). In this section, we present a simple quantum algorithm that estimates $\mathrm{F}(\rho, |\psi\rangle)$ to within additive error ε using $O(1/\varepsilon)$ queries to the state-preparation circuits of ρ and $|\psi\rangle$.

Suppose that the purified quantum query access to ρ and $|\psi\rangle$ is given by two quantum circuits U and V, respectively:

$$U_{\mathsf{AB}}|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}} = |\rho\rangle_{\mathsf{AB}}, \tag{4}$$

$$V_{\mathsf{A'B'}}|0\rangle_{\mathsf{A'}}|0\rangle_{\mathsf{B'}} = |\psi\rangle_{\mathsf{A'}}|\psi'\rangle_{\mathsf{B'}}, \tag{5}$$

where $\rho_{\mathsf{A}} = \mathrm{tr}_{\mathsf{B}}(|\rho\rangle\langle\rho|_{\mathsf{AB}})$ and $|\psi'\rangle_{\mathsf{B'}}$ is any pure state. Here, A, B, $\mathsf{A'}$, and $\mathsf{B'}$ are subscripts of subsystems for clarity, where the subsystems A and $\mathsf{A'}$ have the same dimension. Without loss of generality, we can also assume that the subsystems B and $\mathsf{B'}$ have the same dimension.[1]

4.1 Fidelity to a Pure State

Our idea is to encode $\mathrm{F}(\rho, |\psi\rangle)$ in the amplitudes of an efficiently preparable quantum state, and then take use of the square root amplitude estimation in Theorem 4. To this end, we present a quantum circuit

$$W = (V_{\mathsf{AB}}^{\dagger} \otimes I_{\mathsf{A'B'}}) \cdot \mathsf{SWAP}_{\mathsf{BB'}} \cdot (U_{\mathsf{AB}} \otimes V_{\mathsf{A'B'}}). \tag{6}$$

This circuit is depicted in Figure 3 and it outputs two bits $x_{\mathsf{A}}, x_{\mathsf{B}} \in \{0, 1\}$ such that

$$\Pr[x_{\mathsf{A}} = x_{\mathsf{B}} = 0] = \mathrm{F}^2(\rho, |\psi\rangle). \tag{7}$$

Equation (7) is comparable to Equation (1) for the SWAP test and Equation (3) for the pure-state fidelity estimation, and it can be verified by the following proposition.

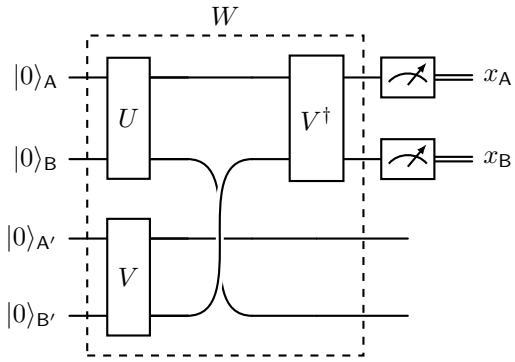


Figure 3 The encoding unitary operator W for $\mathrm{F}(\rho, |\psi\rangle)$.

▶ **Proposition 5.** *$\|(\langle 0|_{\mathsf{A}}\langle 0|_{\mathsf{B}} \otimes I_{\mathsf{A'B'}})W|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}}|0\rangle_{\mathsf{A'}}|0\rangle_{\mathsf{B'}}\|^2 = \mathrm{F}^2(\rho, |\psi\rangle)$, where W is defined by Equation (6), U_{AB} is defined by Equation (4), and $V_{\mathsf{A'B'}}$ is defined by Equation (5).*

[1] If the subsystem B has a larger dimension than $\mathsf{B'}$, then $V_{\mathsf{A'B'}} \otimes I_{\mathsf{C}}$ (for certain subsystem C) can be a purified quantum query oracle for $|\psi\rangle$ with the ancilla system $\mathsf{B'C}$ having the same dimension as B. A similar construction also applies to the case that the subsystem $\mathsf{B'}$ has a larger dimension than B.

Proof. This can be shown by direct calculations.

$$
\begin{aligned}
&\|(\langle 0|_{\mathsf{A}}\langle 0|_{\mathsf{B}} \otimes I_{\mathsf{A}'\mathsf{B}'})W|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}}|0\rangle_{\mathsf{A}'}|0\rangle_{\mathsf{B}'}\|^2 \\
&= \left\|(\langle 0|_{\mathsf{A}}\langle 0|_{\mathsf{B}} \otimes I_{\mathsf{A}'\mathsf{B}'})(V_{\mathsf{AB}}^{\dagger} \otimes I_{\mathsf{A}'\mathsf{B}'}) \cdot \mathsf{SWAP}_{\mathsf{BB}'} \cdot (U_{\mathsf{AB}} \otimes V_{\mathsf{A}'\mathsf{B}'})|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}}|0\rangle_{\mathsf{A}'}|0\rangle_{\mathsf{B}'}\right\|^2 \\
&= \|(\langle \psi|_{\mathsf{A}}\langle \psi'|_{\mathsf{B}} \otimes I_{\mathsf{A}'\mathsf{B}'}) \cdot \mathsf{SWAP}_{\mathsf{BB}'} \cdot |\rho\rangle_{\mathsf{AB}}|\psi\rangle_{\mathsf{A}'}|\psi'\rangle_{\mathsf{B}'}\|^2 \\
&= \|(\langle \psi|_{\mathsf{A}} \otimes I_{\mathsf{BA}'} \otimes \langle \psi'|_{\mathsf{B}'}) \cdot |\rho\rangle_{\mathsf{AB}}|\psi\rangle_{\mathsf{A}'}|\psi'\rangle_{\mathsf{B}'}\|^2 \\
&= \|(\langle \psi|_{\mathsf{A}} \otimes I_{\mathsf{BA}'}) \cdot |\rho\rangle_{\mathsf{AB}}|\psi\rangle_{\mathsf{A}'}\|^2 \\
&= \langle \rho|_{\mathsf{AB}}\langle \psi|_{\mathsf{A}'} \cdot (|\psi\rangle\langle\psi|_{\mathsf{A}} \otimes I_{\mathsf{BA}'}) \cdot |\rho\rangle_{\mathsf{AB}}|\psi\rangle_{\mathsf{A}'} \\
&= \langle \rho|_{\mathsf{AB}} \cdot (|\psi\rangle\langle\psi|_{\mathsf{A}} \otimes I_{\mathsf{B}}) \cdot |\rho\rangle_{\mathsf{AB}} \\
&= \operatorname{tr}\big((|\psi\rangle\langle\psi|_{\mathsf{A}} \otimes I_{\mathsf{B}}) \cdot |\rho\rangle\langle\rho|_{\mathsf{AB}}\big) \\
&= \operatorname{tr}\Big(\operatorname{tr}_{\mathsf{B}}\big((|\psi\rangle\langle\psi|_{\mathsf{A}} \otimes I_{\mathsf{B}}) \cdot |\rho\rangle\langle\rho|_{\mathsf{AB}}\big)\Big) \\
&= \operatorname{tr}\big(|\psi\rangle\langle\psi|_{\mathsf{A}} \cdot \operatorname{tr}_{\mathsf{B}}(|\rho\rangle\langle\rho|_{\mathsf{AB}})\big) \\
&= \operatorname{tr}(|\psi\rangle\langle\psi|_{\mathsf{A}} \cdot \rho_{\mathsf{A}}) \\
&= \mathrm{F}^2(\rho, |\psi\rangle).
\end{aligned}
$$

◀

According to Proposition 5, we can write

$$
W|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}}|0\rangle_{\mathsf{A}'}|0\rangle_{\mathsf{B}'} = \mathrm{F}(\rho, |\psi\rangle)|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}}|\phi\rangle_{\mathsf{A}'\mathsf{B}'} + \sqrt{1 - \mathrm{F}^2(\rho, |\psi\rangle)}|\phi^{\perp}\rangle_{\mathsf{ABA}'\mathsf{B}'} \tag{8}
$$

for some normalized pure states $|\phi\rangle$ and $|\phi^{\perp}\rangle$ such that $(|0\rangle\langle 0|_{\mathsf{A}} \otimes |0\rangle\langle 0|_{\mathsf{B}} \otimes I_{\mathsf{A}'\mathsf{B}'})|\phi^{\perp}\rangle = 0$. With this, we can therefore estimate $\mathrm{F}(\rho, |\psi\rangle)$ using the square root amplitude estimation (Theorem 4). We formally state the result and give its rigorous proof as follows.

Algorithm 1 Quantum query algorithm for estimating fidelity to a pure state.

Input: Quantum oracles U and V that prepare n-qubit purifications of a k-qubit mixed state ρ and a k-qubit pure state $|\psi\rangle\langle\psi|$, respectively (as well as $U^{\dagger}$, $V^{\dagger}$, and their controlled versions); the desired additive error $\varepsilon \in (0, 1)$.

Output: An estimate of $\mathrm{F}(|\psi\rangle, \rho)$ to within additive error ε with probability at least $2/3$.

1: Let unitary operator

$$
\begin{aligned}
W' = (I_{\mathsf{C}} \otimes |0\rangle\langle 0|_{\mathsf{A}} \otimes |0\rangle\langle 0|_{\mathsf{B}} \otimes I_{\mathsf{A}'\mathsf{B}'} + X_{\mathsf{C}} \otimes (I_{\mathsf{AB}} - |0\rangle\langle 0|_{\mathsf{A}} \otimes |0\rangle\langle 0|_{\mathsf{B}}) \otimes I_{\mathsf{A}'\mathsf{B}'}) \\
\cdot (V_{\mathsf{AB}}^{\dagger} \otimes I_{\mathsf{A}'\mathsf{B}'}) \cdot \mathsf{SWAP}_{\mathsf{BB}'} \cdot (U_{\mathsf{AB}} \otimes V_{\mathsf{A}'\mathsf{B}'}).
\end{aligned}
$$

2: **return** $\mathsf{SqrtAmpEst}(W', \varepsilon)$ by Theorem 4.

▶ **Theorem 6** (Estimating fidelity to a pure state)**.** *Suppose that U and V are quantum unitary operators that prepare n-qubit purifications of a k-qubit mixed state $\rho = \operatorname{tr}_{\mathsf{B}}(|\rho\rangle\langle\rho|_{\mathsf{AB}})$ and a k-qubit pure state $|\psi\rangle\langle\psi| = |\psi\rangle\langle\psi|_{\mathsf{A}'}$, respectively, as in Equations (4) and (5). For $\varepsilon \in (0, 1)$, there is a quantum query algorithm that estimates $\mathrm{F}(\rho, |\psi\rangle)$ to within additive error ε with probability at least $2/3$ using $O(1/\varepsilon)$ queries to (controlled-)U, (controlled-)V, and their inverses.*

Proof. The formal algorithm is given in Algorithm 1. To make the proof rigorous, we introduce an auxiliary system C that consists of one qubit. Let

$$
W' = \big((I_{\mathsf{C}} \otimes |0\rangle\langle 0|_{\mathsf{A}} \otimes |0\rangle\langle 0|_{\mathsf{B}} + X_{\mathsf{C}} \otimes (I_{\mathsf{AB}} - |0\rangle\langle 0|_{\mathsf{A}} \otimes |0\rangle\langle 0|_{\mathsf{B}})) \otimes I_{\mathsf{A}'\mathsf{B}'}\big)W, \tag{9}
$$

where W is defined by Equation (6). According to Equation (8), we have

$$W'|0\rangle_{\mathsf{C}}|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}}|0\rangle_{\mathsf{A}'}|0\rangle_{\mathsf{B}'} = \mathrm{F}(|\psi\rangle,\rho)|0\rangle_{\mathsf{C}}|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}}|\phi\rangle_{\mathsf{A}'\mathsf{B}'} + \sqrt{1-\mathrm{F}(|\psi\rangle,\rho)^2}|1\rangle_{\mathsf{C}}|\phi^{\perp}\rangle_{\mathsf{ABA}'\mathsf{B}'}.$$

Finally, we can obtain an estimate $\tilde{x} = \mathsf{SqrtAmpEst}(W',\varepsilon)$ of $\mathrm{F}(|\psi\rangle,\rho)$ to within additive error ε (i.e., $|\tilde{x} - \mathrm{F}(|\psi\rangle,\rho)| < \varepsilon$) with probability at least $2/3$ using $O(1/\varepsilon)$ queries to W'. According to Equations (6) and (9), one query to W' consists of one query to U and two queries to V. Therefore, the way we obtain $\tilde{x}$ uses $O(1/\varepsilon)$ queries to both U and V. ◀

4.2 Generalization to Two Mixed States

In this subsection, we naturally extend the unitary operator in Figure 3 to the scenario where V prepares a purification of an arbitrary mixed state σ. That is, we replace Equation (5) with

$$V_{\mathsf{A}'\mathsf{B}'}|0\rangle_{\mathsf{A}'}|0\rangle_{\mathsf{B}'} = |\sigma\rangle_{\mathsf{A}'\mathsf{B}'}, \tag{10}$$

where $\sigma_{\mathsf{A}'} = \mathrm{tr}_{\mathsf{B}'}(|\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})$. In this case, it turns out that $\sqrt{\mathrm{tr}(\rho\sigma^2)}$ is encoded in the amplitude. Note that when σ is pure, $\sigma^2 = \sigma$, the quantity $\sqrt{\mathrm{tr}(\rho\sigma^2)}$ equals the fidelity $\mathrm{F}(\rho,\sigma)$. Specifically, the circuit in Figure 3 outputs two bits $x_{\mathsf{A}}, x_{\mathsf{B}} \in \{0,1\}$ such that

$$\Pr[x_{\mathsf{A}} = x_{\mathsf{B}} = 0] = \mathrm{tr}(\rho\sigma^2), \tag{11}$$

which generalizes Equation (7). This can be verified by the following proposition.

▶ **Proposition 7.** *$\|(\langle 0|_{\mathsf{A}}\langle 0|_{\mathsf{B}} \otimes I_{\mathsf{A}'\mathsf{B}'})W|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}}|0\rangle_{\mathsf{A}'}|0\rangle_{\mathsf{B}'}\|^2 = \mathrm{tr}(\rho\sigma^2)$, where W is defined by Equation (6), U_{AB} is defined by Equation (4), and $V_{\mathsf{A}'\mathsf{B}'}$ is defined by Equation (10).*

Proof. This can be shown by direct calculations generalizing the proof of Proposition 5.

$$\begin{aligned}
&\|(\langle 0|_{\mathsf{A}}\langle 0|_{\mathsf{B}} \otimes I_{\mathsf{A}'\mathsf{B}'})W|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}}|0\rangle_{\mathsf{A}'}|0\rangle_{\mathsf{B}'}\|^2 \\
&= \|(\langle\sigma|_{\mathsf{AB}} \otimes I_{\mathsf{A}'\mathsf{B}'}) \cdot \mathsf{SWAP}_{\mathsf{BB}'} \cdot |\rho\rangle_{\mathsf{AB}}|\sigma\rangle_{\mathsf{A}'\mathsf{B}'}\|^2 \\
&= \langle\rho|_{\mathsf{AB}}\langle\sigma|_{\mathsf{A}'\mathsf{B}'} \cdot \mathsf{SWAP}_{\mathsf{BB}'} \cdot (|\sigma\rangle\langle\sigma|_{\mathsf{AB}} \otimes I_{\mathsf{A}'\mathsf{B}'}) \cdot \mathsf{SWAP}_{\mathsf{BB}'} \cdot |\rho\rangle_{\mathsf{AB}}|\sigma\rangle_{\mathsf{A}'\mathsf{B}'} \\
&= \mathrm{tr}\big(\mathsf{SWAP}_{\mathsf{BB}'}(|\sigma\rangle\langle\sigma|_{\mathsf{AB}} \otimes I_{\mathsf{A}'\mathsf{B}'})\mathsf{SWAP}_{\mathsf{BB}'} \cdot (|\rho\rangle\langle\rho|_{\mathsf{AB}} \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\big) \\
&= \mathrm{tr}\big(\mathsf{SWAP}_{\mathsf{AA}'}(I_{\mathsf{AB}} \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\mathsf{SWAP}_{\mathsf{AA}'} \cdot (|\rho\rangle\langle\rho|_{\mathsf{AB}} \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\big) \\
&= \mathrm{tr}\Big(\mathrm{tr}_{\mathsf{B}}\big(\mathsf{SWAP}_{\mathsf{AA}'}(I_{\mathsf{AB}} \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\mathsf{SWAP}_{\mathsf{AA}'} \cdot (|\rho\rangle\langle\rho|_{\mathsf{AB}} \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\big)\Big) \\
&= \mathrm{tr}\big(\mathsf{SWAP}_{\mathsf{AA}'}(I_{\mathsf{A}} \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\mathsf{SWAP}_{\mathsf{AA}'} \cdot (\mathrm{tr}_{\mathsf{B}}(|\rho\rangle\langle\rho|_{\mathsf{AB}}) \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\big) \\
&= \mathrm{tr}\big(\mathsf{SWAP}_{\mathsf{AA}'}(I_{\mathsf{A}} \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\mathsf{SWAP}_{\mathsf{AA}'} \cdot (\rho_{\mathsf{A}} \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\big) \\
&= \mathrm{tr}\big((I_{\mathsf{A}} \otimes \langle\sigma|_{\mathsf{A}'\mathsf{B}'}) \cdot \mathsf{SWAP}_{\mathsf{AA}'}(I_{\mathsf{A}} \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\mathsf{SWAP}_{\mathsf{AA}'} \cdot (\rho_{\mathsf{A}} \otimes |\sigma\rangle_{\mathsf{A}'\mathsf{B}'})\big) \\
&= \mathrm{tr}\Big(\big((I_{\mathsf{A}} \otimes \langle\sigma|_{\mathsf{A}'\mathsf{B}'})\mathsf{SWAP}_{\mathsf{AA}'}(I_{\mathsf{A}} \otimes |\sigma\rangle_{\mathsf{A}'\mathsf{B}'})\big)^2 \cdot \rho_{\mathsf{A}}\Big).
\end{aligned}$$

The proof is completed by showing that $(I_{\mathsf{A}} \otimes \langle\sigma|_{\mathsf{A}'\mathsf{B}'})\mathsf{SWAP}_{\mathsf{AA}'}(I_{\mathsf{A}} \otimes |\sigma\rangle_{\mathsf{A}'\mathsf{B}'}) = \sigma_{\mathsf{A}}$. To see this, note that

$$
\begin{aligned}
&(I_{\mathsf{A}} \otimes \langle\sigma|_{\mathsf{A}'\mathsf{B}'})\mathsf{SWAP}_{\mathsf{A}\mathsf{A}'}(I_{\mathsf{A}} \otimes |\sigma\rangle_{\mathsf{A}'\mathsf{B}'}) \\
=\ & \mathrm{tr}_{\mathsf{A}'\mathsf{B}'}\big(\mathsf{SWAP}_{\mathsf{A}\mathsf{A}'} \cdot (I_{\mathsf{A}} \otimes |\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\big) \\
=\ & \mathrm{tr}_{\mathsf{A}'}\Big(\mathsf{SWAP}_{\mathsf{A}\mathsf{A}'} \cdot \big(I_{\mathsf{A}} \otimes \mathrm{tr}_{\mathsf{B}'}(|\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})\big)\Big) \\
=\ & \mathrm{tr}_{\mathsf{A}'}\big(\mathsf{SWAP}_{\mathsf{A}\mathsf{A}'} \cdot (I_{\mathsf{A}} \otimes \sigma_{\mathsf{A}'})\big) \\
=\ & \mathrm{tr}_{\mathsf{A}'}\big((\sigma_{\mathsf{A}} \otimes I_{\mathsf{A}'}) \cdot \mathsf{SWAP}_{\mathsf{A}\mathsf{A}'}\big) \\
=\ & \sigma_{\mathsf{A}} \cdot \mathrm{tr}_{\mathsf{A}'}(\mathsf{SWAP}_{\mathsf{A}\mathsf{A}'}) \\
=\ & \sigma_{\mathsf{A}} \cdot I_{\mathsf{A}} \\
=\ & \sigma_{\mathsf{A}}. \qquad \blacktriangleleft
\end{aligned}
$$

Therefore, with a similar analysis, we can estimate the quantity $\sqrt{\mathrm{tr}(\rho\sigma^2)}$ by Proposition 7 with the same complexity as Theorem 6, stated as follows.

▶ **Theorem 8.** *Suppose that U and V are quantum unitary operators that prepare n-qubit purifications of a k-qubit mixed states $\rho = \mathrm{tr}_{\mathsf{B}}(|\rho\rangle\langle\rho|_{\mathsf{A}\mathsf{B}})$ and $\sigma = \mathrm{tr}_{\mathsf{B}'}(|\sigma\rangle\langle\sigma|_{\mathsf{A}'\mathsf{B}'})$, respectively, as in Equations (4) and (10). For $\varepsilon \in (0,1)$, there is a quantum query algorithm that estimates $\sqrt{\mathrm{tr}(\rho\sigma^2)}$ to within additive error ε with probability at least $2/3$ using $O(1/\varepsilon)$ queries to (controlled-)U, (controlled-)V, and their inverses.*

To provide an intuitive understanding of our encoding scheme, we restructure the circuit in Figure 3 into the form shown in Figure 4. This revised visualization explicitly reveals the

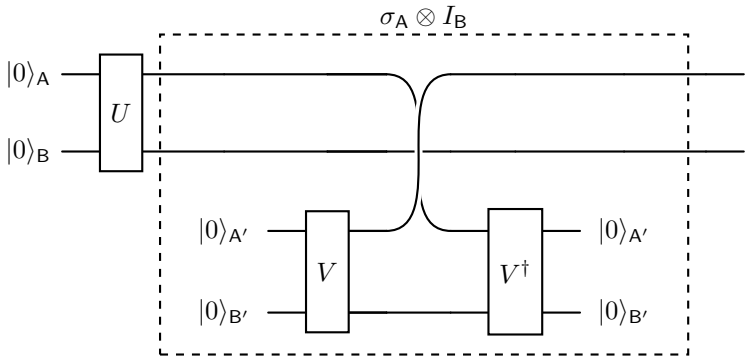


Figure 4 Restructured circuit of Figure 3, illustrating the operator $\sigma_{\mathsf{A}} \otimes I_{\mathsf{B}}$ and its action on the purification $|\rho\rangle_{\mathsf{A}\mathsf{B}}$.

construction of an operator $\sigma_{\mathsf{A}} \otimes I_{\mathsf{B}}$ acting on the purification $|\rho\rangle_{\mathsf{A}\mathsf{B}}$ of ρ_{A}. Since

$$\|(\sigma_{\mathsf{A}} \otimes I_{\mathsf{B}})|\rho\rangle_{\mathsf{A}\mathsf{B}}\|^2 = \mathrm{tr}\big(\rho\sigma^2\big),$$

we clearly see that $\sqrt{\mathrm{tr}(\rho\sigma^2)}$ is encoded into the amplitude of an efficiently preparable quantum state. While the encoding pattern for σ in Figure 4 had previously appeared in [17], people mainly focused on operator properties without explicitly deriving the amplitude structure of the resulting pure state.

5 An Implication: Generalized Pure-State Fidelity Estimation

Our quantum algorithm provided in Theorem 1 suggests a new approach to estimating the fidelity between pure states, where the pure states are given through purified quantum query access, or equivalently prepared by (purified) quantum channels. The purified quantum query

access to a (mixed) quantum state can be understood as a purified version of the quantum channel that prepares the quantum state (cf. [9]). Formally, a quantum channel $\mathcal{E}$ is said to prepare a quantum state ρ, if $\rho = \mathcal{E}(|0\rangle\langle 0|)$. A purified version of $\mathcal{E}$ is a unitary operator U acting on two subsystems A and B such that $\mathcal{E}(\sigma) = \mathrm{tr}_{\mathsf{B}}(U(\sigma_{\mathsf{A}} \otimes |0\rangle\langle 0|_{\mathsf{B}})U^{\dagger})$ for every mixed quantum state σ. Then, ρ can be prepared by the unitary operator U in the way that its purification is prepared by $|\rho\rangle_{\mathsf{AB}} = U|0\rangle_{\mathsf{A}}|0\rangle_{\mathsf{B}}$ and ρ is obtained by tracing out the subsystem B, i.e., $\rho = \mathrm{tr}_{\mathsf{B}}(|\rho\rangle\langle\rho|_{\mathsf{AB}})$. In the following, we state the special case of Theorem 1 for pure-state fidelity estimation.

▶ **Corollary 9** (Pure-state fidelity estimation). *Given purified quantum query access to two pure states $|\varphi\rangle$ and $|\psi\rangle$, the fidelity $\mathrm{F}(|\varphi\rangle, |\psi\rangle) = |\langle\varphi|\psi\rangle|$ can be estimated to within additive error ε with query complexity $O(1/\varepsilon)$.*

Corollary 9 generalizes the pure-state fidelity estimation in [25, Theorem IV.2] to a more general input model. By comparison, the prior approach for pure-state fidelity estimation [25] requires that the pure states should be prepared by a unitary quantum channel. Specifically, Ref. [25] assumes two unitary oracles U_{φ} and U_{ψ} such that $U_{\varphi}|0\rangle = |\varphi\rangle$ and $U_{\psi}|0\rangle = |\psi\rangle$ (see Section 3.2 for details). The input model employed in [25] is more restricted than the purified quantum query access model. For example, $U_{\varphi} \otimes I$ always provides purified quantum query access to $|\varphi\rangle$, where I is the identity operator; however, it is unlikely that U_{φ} could be implemented by purified quantum query access to $|\varphi\rangle$.

In addition to adopting a more general input model than [25], Corollary 9 attains the same optimal query complexity of $O(1/\varepsilon)$, where the lower bound (noted in [25]) is implied in [2, 19] (see Theorem 10).

6 Lower Bounds

For completeness, we discuss the lower bounds for estimating the fidelity. In fact, a lower bound of $\Omega(1/\varepsilon)$ is implied in [2, 19] when both quantum states are pure (as noted in [25]).

▶ **Theorem 10** (Adapted from [2, 19]). *Given purified quantum query access to quantum states ρ and $|\psi\rangle$, any quantum query algorithm that estimates $\mathrm{F}(\rho, |\psi\rangle)$ to within additive error ε requires query complexity $\Omega(1/\varepsilon)$, even if ρ is pure.*

We strengthen this lower bound so that it applies to the case where ρ is a mixed quantum state of an arbitrary rank.

▶ **Theorem 11.** *Given purified quantum query access to quantum states ρ and $|\psi\rangle$, any quantum query algorithm that estimates $\mathrm{F}(\rho, |\psi\rangle)$ to within additive error ε requires query complexity $\Omega(1/\varepsilon)$, even if ρ is of an arbitrary rank.*

To prove Theorem 11, we need the following tool.

▶ **Theorem 12** ([3, Theorem 4]). *Let p and q be probability distributions over n elements. Then, any quantum algorithm that determines whether an unknown unitary oracle U is*

$$U_p = \sum_{j=1}^{n} \sqrt{p_j}|j\rangle \text{ or } U_q = \sum_{j=1}^{n} \sqrt{q_j}|j\rangle$$

requires $\Omega(1/d_{\mathrm{H}}(p,q))$ queries to U, where

$$d_{\mathrm{H}}(p,q) = \sqrt{\frac{1}{2}\sum_{j=1}^{n}\left(\sqrt{p_j} - \sqrt{q_j}\right)^2}$$

is the Hellinger distance.

Now we are ready to prove the lower bounds.

Proof of Theorem 11. Let $r = \mathrm{rank}(\rho) \geq 2$ be an arbitrary rank parameter. Now we reduce the distinguishing problem in Theorem 12 to the estimation of $\mathrm{F}(\rho, |\psi\rangle)$. Consider the two probability distributions $p^{\pm}$ such that

$$p_1^{\pm} = p \pm \varepsilon,\ p_j^{\pm} = \frac{1 - p \mp \varepsilon}{r - 1} \text{ for } 2 \leq j \leq r, \text{ and } p_j^{\pm} = 0 \text{ for } r < j \leq n,$$

where $p \in (0, 1)$ is an arbitrary constant. It can be verified that

$$d_{\mathrm{H}}(p^+, p^-) = \sqrt{1 - \sqrt{p^2 - \varepsilon^2} - \sqrt{(1-p)^2 - \varepsilon^2}} \leq O(\varepsilon).$$

To distinguish whether an unknown unitary oracle U is U_{p^+} or U_{p^-}, we can prepare a mixed state ρ that is either of the following two mixed states

$$\rho_{\pm} = (p \pm \varepsilon)|0\rangle\langle 0| + \sum_{j=2}^{r} \frac{1 - p \mp \varepsilon}{r - 1} |j\rangle\langle j|,$$

using one query to U. By taking $|\psi\rangle = |0\rangle$, we can distinguish which the case is by noting that the fidelity $\mathrm{F}(\rho_{\pm}, |0\rangle) = \sqrt{p \pm \varepsilon} = \sqrt{p} \pm \Theta(\varepsilon)$. Therefore, any quantum algorithm for estimating $\mathrm{F}(\rho, |\psi\rangle)$ to within additive error $\Theta(\varepsilon)$ requires query complexity $\Omega(1/d_{\mathrm{H}}(p^+, p^-)) = \Omega(1/\varepsilon)$, even if ρ is of rank r. ◀

7 Discussion

In this paper, we present an optimal quantum algorithm for estimating the fidelity of a mixed state to a pure state, given purified quantum query access. Our approach is simple, which moreover estimates the quantity $\sqrt{\mathrm{tr}(\rho\sigma^2)}$ that has not been commonly considered in the literature. Here, we raise some questions for future research.

1. Can we encode $\sqrt{\mathrm{tr}(\rho\sigma)}$ rather than $\sqrt{\mathrm{tr}(\rho\sigma^2)}$ in the amplitudes? In addition to its relationship with fidelity estimation, this may also estimate the *Frobenius norm* of a quantum state, $\|\rho\|_F = \sqrt{\mathrm{tr}(\rho^2)}$, which is also the square root of purity.
2. Can we estimate the fidelity $\mathrm{F}(\rho, |\psi\rangle)$ with *optimal* sample complexity, generalizing the result of [28]?

References

1 Anurag Anshu, Zeph Landau, and Yunchao Liu. Distributed quantum inner product estimation. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 44–51, 2022. doi:10.1145/3519935.3519974.

2 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. doi:10.1145/502090.502097.

3 Aleksandrs Belovs. Quantum algorithms for classical probability distributions. In *Proceedings of the 27th Annual European Symposium on Algorithms*, pages 16:1–16:11, 2019. doi:10.4230/LIPIcs.ESA.2019.16.

4 Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In Samuel J. Lomonaco, Jr. and Howard E. Brandt, editors, *Quantum Computation and Information*, volume 305 of *Contemporary Mathematics*, pages 53–74. AMS, 2002. doi:10.1090/conm/305/05215.

5 Harry Buhrman, Richard Cleve, John Watrous, and Ronald de Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001. doi:10.1103/PhysRevLett.87.167902.

6 Wang Fang and Qisheng Wang. Optimal quantum algorithm for estimating fidelity to a pure state. ArXiv e-prints, 2025. doi:10.48550/arXiv.2506.23650.

7 Steven T. Flammia and Yi-Kai Liu. Direct fidelity estimation from few Pauli measurements. *Physical Review Letters*, 106(23):230501, 2011. doi:10.1103/PhysRevLett.106.230501.

8 András Gilyén and Tongyang Li. Distributional property testing in a quantum world. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference*, pages 25:1–25:19, 2020. doi:10.4230/LIPIcs.ITCS.2020.25.

9 András Gilyén, Seth Lloyd, Iman Marvian, Yihui Quek, and Mark M. Wilde. Quantum algorithm for Petz recovery channels and pretty good measurements. *Physical Review Letters*, 128(22):220502, 2022. doi:10.1103/PhysRevLett.128.220502.

10 András Gilyén and Alexander Poremba. Improved quantum algorithms for fidelity estimation. ArXiv e-prints, 2022. arXiv:2203.15993.

11 Tom Gur, Min-Hsiu Hsieh, and Sathyawageeswar Subramanian. Sublinear quantum algorithms for estimating von Neumann entropy. ArXiv e-prints, 2021. arXiv:2111.11139.

12 Masahito Hayashi. *Quantum Information Theory: Mathematical Foundation*. Springer, 2017. doi:10.1007/978-3-662-49725-8.

13 Richard Jozsa. Fidelity for mixed quantum states. *Journal of Modern Optics*, 41(12):2315–2323, 1994. doi:10.1080/09500349414552171.

14 Hirotada Kobayashi, Keiji Matsumoto, and Tomoyuki Yamakami. Quantum Merlin-Arthur proof systems: are multiple Merlins more helpful to Arthur? *Chicago Journal of Theoretical Computer Science*, 2009:3, 2009. doi:10.4086/cjtcs.2009.003.

15 Nana Liu, Qisheng Wang, Mark M. Wilde, and Zhicheng Zhang. Quantum algorithm for matrix geometric means. *npj Quantum Information*, 11:101, 2025. doi:10.1038/s41534-025-00973-7.

16 Yupan Liu and Qisheng Wang. On estimating the trace of quantum state powers. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 947–993, 2025. doi:10.1137/1.9781611978322.28.

17 Guang Hao Low and Isaac L. Chuang. Hamiltonian Simulation by Qubitization. *Quantum*, 3:163, 2019. doi:10.22331/q-2019-07-12-163.

18 Ashley Montanaro and Ronald de Wolf. A survey of quantum property testing. In *Theory of Computing Library*, number 7 in Graduate Surveys, pages 1–81. University of Chicago, 2016. doi:10.4086/toc.gs.2016.007.

19 Ashwin Nayak and Felix Wu. The quantum query complexity of approximating the median and related statistics. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 384–393, 1999. doi:10.1145/301250.301349.

20 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

21 Harumichi Nishimura. A survey: SWAP test and its applications to quantum complexity theory. In Shin-ichi Minato, Takeaki Uno, Norihito Yasuda, Takashi Horiyama, Ken-ichi Kawarabayashi, Shigeru Yamashita, and Hirotaka Ono, editors, *Algorithmic Foundations for Social Advancement: Recent Progress on Theory and Practice*, pages 243–261. Springer, 2025. doi:10.1007/978-981-96-0668-9_16.

22 Soorya Rethinasamy, Rochisha Agarwal, Kunal Sharma, and Mark M. Wilde. Estimating distinguishability measures on quantum computers. *Physical Review A*, 108(1):012409, 2023. doi:10.1103/PhysRevA.108.012409.

23 Sathyawageeswar Subramanian and Min-Hsiu Hsieh. Quantum algorithm for estimating α-renyi entropies of quantum states. *Physical Review A*, 104(2):022428, 2021. doi:10.1103/PhysRevA.104.022428.

24 Armin Uhlmann. The "transition probability" in the state space of a *-algebra. *Reports on Mathematical Physics*, 9(2):273–279, 1976. doi:10.1016/0034-4877(76)90060-4.

25 Qisheng Wang. Optimal trace distance and fidelity estimations for pure quantum states. *IEEE Transactions on Information Theory*, 70(12):8791–8805, 2024. doi:10.1109/TIT.2024.3447915.

26 Qisheng Wang, Ji Guan, Junyi Liu, Zhicheng Zhang, and Mingsheng Ying. New quantum algorithms for computing quantum entropies and distances. *IEEE Transactions on Information Theory*, 70(8):5653–5680, 2024. doi:10.1109/TIT.2024.3399014.

27 Qisheng Wang and Zhicheng Zhang. Fast quantum algorithms for trace distance estimation. *IEEE Transactions on Information Theory*, 70(4):2720–2733, 2024. doi:10.1109/TIT.2023.3321121.

28 Qisheng Wang and Zhicheng Zhang. Sample-optimal quantum estimators for pure-state trace distance and fidelity via samplizer. ArXiv e-prints, 2024. doi:10.48550/arXiv.2410.21201.

29 Qisheng Wang, Zhicheng Zhang, Kean Chen, Ji Guan, Wang Fang, Junyi Liu, and Mingsheng Ying. Quantum algorithm for fidelity estimation. *IEEE Transactions on Information Theory*, 69(1):273–282, 2023. doi:10.1109/TIT.2022.3203985.

30 Xinzhao Wang, Shengyu Zhang, and Tongyang Li. A quantum algorithm framework for discrete probability distributions with applications to Rényi entropy estimation. *IEEE Transactions on Information Theory*, 70(5):3399–3426, 2024. doi:10.1109/TIT.2024.3382037.

31 John Watrous. Limits on the power of quantum statistical zero-knowledge. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 459–468, 2002. doi:10.1109/SFCS.2002.1181970.

32 John Watrous. Zero-knowledge against quantum attacks. *SIAM Journal on Computing*, 39(1):25–58, 2009. doi:10.1137/060670997.

33 John Watrous. *The Theory of Quantum Information*. Cambridge University Press, 2018. doi:10.1017/9781316848142.

34 Mark M. Wilde. *Quantum Information Theory*. Cambridge University Press, 2013. doi:10.1017/9781316809976.

External-Memory Priority Queues with Optimal Insertions

Gerth Stølting Brodal
Aarhus University, Denmark

Michael T. Goodrich
University of California, Irvine, CA, USA

John Iacono
Université libre de Bruxelles, Belgium

Jared Lo
University of Hawai'i at Mānoa, Honolulu, HI, USA

Ulrich Meyer
Goethe University Frankfurt am Main, Germany

Victor Pagan
University of Hawai'i at Mānoa, Honolulu, HI, USA

Nodari Sitchinava
University of Hawai'i at Mānoa, Honolulu, HI, USA

Rolf Svenning
Aarhus University, Denmark

Abstract

We present an external-memory priority queue structure supporting Insert and DeleteMin with amortized $\mathcal{O}(1)$ and $\mathcal{O}(\lg N)$ comparisons, respectively, and amortized $\mathcal{O}\left(\frac{1}{B}\right)$ and $\mathcal{O}\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os, respectively. Here, M is the size of the internal memory, B is the block size of I/Os between internal and external memory, and N is the number of elements in the priority queue just before an operation is performed. Previous external-memory priority queues required amortized $\mathcal{O}(\lg N)$ comparisons and $\mathcal{O}\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os for both Insert and DeleteMin. The construction requires the minimal assumption $M \geq 2B$.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases priority queues, external memory, cache aware, amortized complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.5

Funding Work initiated while attending the Third AlgoPARC Workshop on Parallel Algorithms and Data Structures at the University of Hawaii at Manoa, in part supported by the National Science Foundation under Grant No. 2452276.
Gerth Stølting Brodal: Supported by Independent Research Fund Denmark, grant 9131-00113B.
Michael T. Goodrich: Supported by NSF grant 2212129.
John Iacono: Research supported by the Fonds de la Recherche Scientifique – FNRS.
Jared Lo: Supported by NSF grant 2432018.
Ulrich Meyer: Supported by Deutsche Forschungsgemeinschaft (DFG) – ME 2088/5-2 (FOR 2975 - Algorithms, Dynamics, and Information Flow in Networks).
Victor Pagan: Supported by NSF grant 2432018.
Nodari Sitchinava: Supported by NSF grant 2432018.
Rolf Svenning: Supported by Independent Research Fund Denmark, grant 9131-00113B.

1 Introduction

Priority queues are fundamental data structures with a host of applications. For example, algorithmic applications for priority queues range from classic results for sorting (such as heapsort) [13, 31, 32] and network optimization [19] to recent instance-optimal results for graph algorithms [22, 23]. In addition, priority queues have been developed for external-memory models [3, 6, 12, 18, 25, 26], ideal-cache models [4, 9], concurrent models [29], and RAM models [30]. Thus, it is desirable to design efficient priority queue data structures.



© Gerth Stølting Brodal, Michael T. Goodrich, John Iacono, Jared Lo, Ulrich Meyer, Victor Pagan, Nodari Sitchinava, and Rolf Svenning;
licensed under Creative Commons License CC-BY 4.0

Throughout the paper, we assume a priority queue stores a multi-set of elements, where each element is a $\langle key, value \rangle$ pair, and the keys are from some totally ordered universe. In the following, a comparison between two elements refers to the comparison of the two elements' keys.

Part of the reason priority queues are so broadly applicable is that they are defined in terms of two simple and useful operations:

- Insert(e) inserts an element e.
- DeleteMin() extracts and returns an element from the priority queue with minimum key. If several elements have equal keys, an arbitrary one of those is returned. This operation requires the priority queue to be non-empty before the operation.

Moreover, in internal memory, it is possible to design priority queues that hold N elements and achieve $\mathcal{O}(1)$ time for Insert and $\mathcal{O}(\lg N)$ time for DeleteMin, either in the worst case [14] or amortized [13, 31][1]. Unfortunately, prior to this work, we are not aware of any efficient data structures in external-memory models [1, 20, 21] that can match the performance of these classic internal-memory data structures.

Internal-memory priority queues

Williams introduced the binary heap in 1964 [32], which supports Insert and DeleteMin with $\mathcal{O}(\lg N)$ comparisons. Since then, many priority queues have been proposed; see, e.g., the survey by Brodal [7]. Vuillemin [31] introduced the binomial queue and demonstrated that it supports a sequence of N Insert and DeleteMin operations using a total of $\mathcal{O}(N \lg N)$ comparisons. Shortly after, Brown [13] gave a detailed analysis of binomial queues. Binomial queues support a sequence of I Insert and D DeleteMin operations in total $\mathcal{O}(I + D \lg I)$ comparisons. Carlson, Munro, and Poblete [14] showed how to achieve binomial queues supporting Insert with worst-case $\mathcal{O}(1)$ comparisons and DeleteMin with worst-case $\mathcal{O}(\lg N)$ comparisons. Fibonacci heaps, introduced by Fredman and Tarjan [19], achieve the same amortized performance as binomial queues for Insert and DeleteMin operations. Additionally, they support the DecreaseKey operation in amortized $\mathcal{O}(1)$ time and comparisons, allowing the key of an element to be replaced with a smaller key. Relaxed heaps, proposed by Driscoll, Gabow, Shrairman, and Tarjan [16], matched these bounds in the worst case.

External-memory model and priority queues

Aggarwal and Vitter [1] introduced the external-memory model as a model of computation focusing on the communication between two levels of memory: an *internal memory* of size M and an infinite *external memory*, where an *I/O* transfers a block of B consecutive memory cells between the two levels of memory. The *I/O cost* of an algorithm is the number of I/Os the algorithm performs. The minimal assumption is that $B \geq 1$ and $M \geq 2B$.

Aggarwal and Vitter proved that comparison-based sorting of N elements in the external-memory model can be done with $\mathcal{O}\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os, and proved a matching lower bound for comparison-based sorting. Since sorting can be performed using a priority queue by performing N Insert operations followed by N DeleteMin operations, the amortized I/O cost of either Insert or DeleteMin must be $\Omega\left(\frac{1}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os. Arge [3], Kumar

[1] $\lg x$ denotes the binary logarithm of x.

and Schwabe [27], and Fadel, Jakobsen, Katajainen, and Teuhola [18] presented external-memory priority queues supporting Insert and DeleteMin in amortized $\mathcal{O}\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os, assuming $M \geq 2B$. Brengel, Crauser, Ferragina and Meyer [6] presented a simple external-memory priority queue denoted an *array heap* with matching I/O bounds, assuming $B\log_{M/B}\frac{N}{B} \leq M$. Brodal and Katajainen [12] achieved a matching worst-case I/O cost. Since the number of DeleteMin operations is always no more than the number of Insert operation, we can charge the deletion cost to the insertions, i.e., we can restate the amortized costs to be $\mathcal{O}\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os for Insert and $\mathcal{O}(1/B)$ I/Os for DeleteMin (or even zero). The question we address in this paper is whether we can swap the two I/O costs, i.e., achieve amortized $\mathcal{O}(1/B)$ I/Os for Insert and $\mathcal{O}\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os for DeleteMin, which is relevant when the number of Insert operations is asymptotically larger than the number of DeleteMin operations, i.e., not all elements inserted into the priority queue are eventually also extracted. Table 1 summarizes previous results, where we let $\mathrm{Sort}(N) = \frac{N}{B}\log_{M/B}\frac{N}{B}$; hence, $\frac{1}{N}\mathrm{Sort}(N) = \frac{1}{B}\log_{M/B}\frac{N}{B}$.

Frigo, Leiserson, Prokop, and Ramachandran [20, 21] introduced the *ideal-cache* model by augmenting the external-memory model with an optimal offline paging mechanism. In this model, instead of having each algorithm explicitly transfer data between the two levels of memory, the data transfers are assumed to be performed by an optimal offline paging mechanism. While this might seem like a strong assumption, they demonstrated that under reasonable resource augmentation assumptions, a more realistic paging mechanism, e.g., the one implementing *least recently used (LRU)* replacement policy, is competitive with an optimal one. Consequently, ideal-cache model allows the design of so-called *cache-oblivious* algorithms – algorithms that are oblivious to the parameters M and B. Frigo et al. [20, 21] presented cache-oblivious sorting algorithms achieving optimal I/O cost, provided a *tall-cache assumption* of $M \geq B^2$. Brodal and Fagerberg [8] proved that the same bounds can be achieved under the weaker tall cache assumption $M \geq B^{1+\varepsilon}$ for any constant $\varepsilon > 0$, with a constant overhead of $\frac{1}{\varepsilon}$ in the I/O bounds. Brodal and Fagerberg [10] proved that this trade-off between the tall-cache assumption and the I/O cost is inherent to cache-oblivious algorithms. Cache-oblivious priority queues were presented by Arge, Bender, Demaine, Holland-Minkley, and Munro [4] and Brodal and Fagerberg [9] achieving $\mathcal{O}\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os for Insert and DeleteMin, under the tall cache assumptions $M \geq B^2$ and $M \geq B^{1+\varepsilon}$, respectively.

In internal memory, several priority queues support Insert and DeleteMin in amortized $\mathcal{O}(\lg N)$ time, and additionally a DecreaseKey operation in amortized $\mathcal{O}(1)$ time (some also in the worst-case and Insert in $\mathcal{O}(1)$ time). A natural question is if it is possible to achieve similar results in external memory, i.e., N Insert and DeleteMin operations with $\mathcal{O}\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os and DecreaseKey with $\mathcal{O}\left(\frac{1}{B}\right)$ I/Os. Eenberg, Larsen, and Yu [17] proved a lower bound that this is not possible. An external-memory priority queue structure by Kumar and Schwabe [27] supports each of the three operations with amortized $\mathcal{O}\left(\frac{1}{B}\lg\frac{N}{B}\right)$ I/Os. Similar bounds were achieved cache-obliviously by Brodal, Fagerberg, Meyer, and Zeh [11] and Chowdhury and Ramachandran [15]. External-memory priority queues with improved DecreaseKey operations were presented by Iacono, Jacob, and Tsakalidis [25], who support Update (a combination of Insert and DecreaseKey) and DeleteMin in amortized $\mathcal{O}\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ and $\mathcal{O}\left(\left\lceil\frac{M^\varepsilon}{B}\log_{M/B}\frac{N}{B}\right\rceil\log_{M/B}\frac{N}{B}\right)$ I/Os, respectively, and by Jiang and Larsen [26], who support Insert, DeleteMin and DecreaseKey in expected amortized time $\mathcal{O}\left(\frac{1}{B}\lg\frac{N}{B}/\lg\lg N\right)$ I/Os, assuming $M > B\lg^{0.01} N$.

Table 1 Selected previous and new comparison and I/O bounds for priority queues. $\mathcal{O}_A$ denotes amortized bounds, whereas $\mathcal{O}$ denotes worst-case bounds.

	Comparisons		I/Os	
	Insert	DeleteMin	Insert	DeleteMin
Internal memory				
Binary heap [32]	$\mathcal{O}(\lg N)$	$\mathcal{O}(\lg N)$		
Binomial queue [13, 31]	$\mathcal{O}_A(1)$	$\mathcal{O}_A(\lg N)$		
Binomial queue [14]	$\mathcal{O}(1)$	$\mathcal{O}(\lg N)$		
External memory				
Buffer tree [3]	$\mathcal{O}_A(\lg N)$	$\mathcal{O}_A(\lg N)$	$\mathcal{O}_A\left(\frac{1}{N}\text{Sort}(N)\right)$	$\mathcal{O}_A\left(\frac{1}{N}\text{Sort}(N)\right)$
Buffered multiway heap [18, 27]				
Brodal-Katajainen [12]	$\mathcal{O}(\lg N)$	$\mathcal{O}(\lg N)$	$\mathcal{O}\left(\frac{1}{N}\text{Sort}(N)\right)$	$\mathcal{O}\left(\frac{1}{N}\text{Sort}(N)\right)$
New (Theorem 1)	$\mathcal{O}_A(1)$	$\mathcal{O}_A(\lg N)$	$\mathcal{O}_A\left(\frac{1}{B}\right)$	$\mathcal{O}_A\left(\frac{1}{N}\text{Sort}(N)\right)$
Cache oblivious				
Arge et al. [4], Funnel-heap [9]	$\mathcal{O}_A(\lg N)$	$\mathcal{O}_A(\lg N)$	$\mathcal{O}_A\left(\frac{1}{N}\text{Sort}(N)\right)$	$\mathcal{O}_A\left(\frac{1}{N}\text{Sort}(N)\right)$

Result

In this paper, we present an external-memory priority queue that is optimal both with respect to the amortized I/Os and comparisons performed, achieving the following result.

▶ **Theorem 1.** *There exists an external-memory priority queue supporting* Insert *with amortized* $\mathcal{O}(1)$ *comparisons and* $\mathcal{O}\left(\frac{1}{B}\right)$ *I/Os, and* DeleteMin *with amortized* $\mathcal{O}(\lg N)$ *comparisons and* $\mathcal{O}\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ *I/Os, where* N *is the current number of elements in the priority queue. The space usage is* $\mathcal{O}\left(\frac{N}{B}\right)$ *blocks. The memory size only needs to satisfy the minimal requirement* $M \geq 2B$.

We achieve this result using what can intuitively be considered an element "juggling" scheme, where we maintain three types of data structures – two in internal memory and one in external memory, transferring elements between them as needed. In a nutshell, our result is obtained by maintaining an internal-memory priority queue with the $\mathcal{O}(M)$ smallest elements supporting Insert and DeleteMin with $\mathcal{O}(1)$ and $\mathcal{O}(\lg M)$ comparisons. We also maintain an insert-buffer in internal memory as a buffer between the internal-memory priority queue and an external-memory structure comprising a forest of multi-way heaps. Specifically, in external memory, we store multi-way heap structures with buffers at the nodes, somewhat similar to an approach by Fadel, Jakobsen, Katajainen, and Teuhola [18] but adapted to support the insertion of batches of $\mathcal{O}(M)$ elements using $\mathcal{O}\left(\frac{M}{B}\right)$ I/Os. Whereas nodes in this previous construction store sorted buffers, in our construction, we maintain the forest of heaps in external memory to use partially sorted buffers (*semi-sorted* at internal nodes and *lazy semi-sorted* at the leaves) to avoid insertions requiring amortized $\Omega(\lg M)$ comparisons.

2 External-memory priority queue

In Sections 3–7, we describe the details of an external-memory priority queue achieving Lemma 2 below, where we bound the total number of comparisons and I/Os performed by a sequence of priority queue operations in terms of the total numbers of Insert and DeleteMin operations performed, denoted I and D, respectively. In Section 8, we then apply global rebuilding to make the bounds depend on the current number of elements in the priority queue, achieving the amortized bounds in Theorem 1.

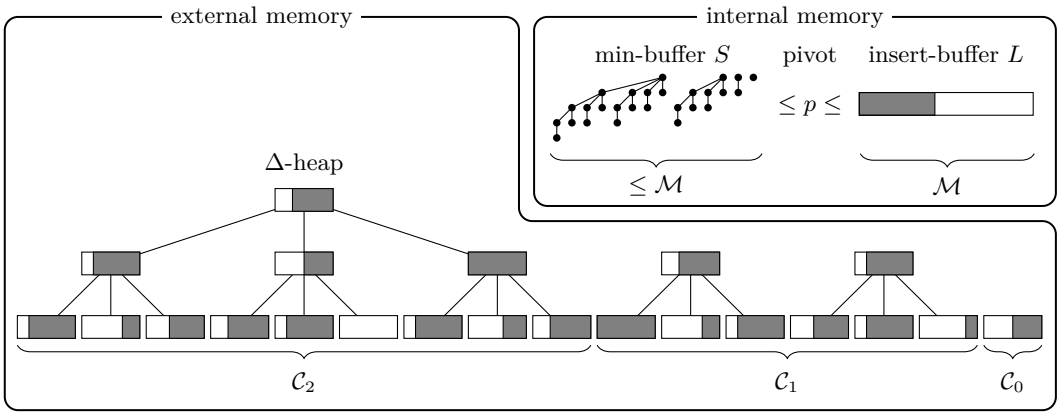


Figure 1 External-memory priority queue. All rectangles are buffers of capacity $\mathcal{M}$, where the gray area illustrates elements in buffers.

▶ **Lemma 2.** *There exists an external-memory priority queue supporting a sequence of I* Insert *and D* DeleteMin *operations, using a total of $\mathcal{O}(I + D \lg I)$ comparisons and $\mathcal{O}\left(\frac{1}{B}\left(I + D \log_{M/B} \frac{I}{B}\right)\right)$ I/Os, assuming $M \geq 2B$.*

Our priority queue consists of an internal-memory part and an external-memory part; see Figure 1 for an overview. Each element is stored exactly once in our data structure. In internal memory we store a *min-buffer*, a *pivot* element, and an *insert-buffer*, where the elements in the min-buffer are all smaller than or equal to the pivot, and the elements in the insert-buffer and external memory are all larger than or equal to the pivot element. By default, Insert and DeleteMin are performed in internal memory without performing any I/Os. If internal memory contains too many elements, $\Theta(M)$ elements are moved from the insert-buffer to the external part. To perform DeleteMin when the min-buffer is empty, $\Theta(M)$ smallest elements from external memory are moved to internal memory. We call moving $\Theta(M)$ elements between internal and external memory a *transfer*. In the subsequent sections, we describe the two parts in detail. We let $\mathcal{M}$ be a parameter for our construction, such that $\mathcal{M}$ is even, $\mathcal{M}$ is divisible by B, and $\mathcal{M} = \Theta(M)$. We choose $\mathcal{M}$ such that the $\mathcal{O}(\mathcal{M})$ space internal-memory data structure in Section 3 fits into internal memory. To be able to choose $\mathcal{M}$, M is required to be sufficiently large and the internal memory to be able to hold a sufficiently large number of blocks of size B. If $M = \mathcal{O}(1)$, then an internal-memory data structure like a binomial queue already achieves the result of Lemma 2, and if the number of blocks fitting in internal memory is too small we can simulate a smaller block size with a constant blow up in the I/O complexity. I.e., in the following we w.l.o.g. can assume that M and M/B are sufficiently large to be able to choose $\mathcal{M}$.

3 The internal-memory part

The internal-memory part consists of a *min-buffer* S, a *pivot* element p, and an *insert-buffer* L. The pivot is larger than or equal to any element in the min-buffer, and smaller than or equal to any element in the insert-buffer and external memory. The insert-buffer is an unordered array of at most $\mathcal{M}$ elements. The min-buffer stores the overall at most $\mathcal{M}$ smallest elements in the priority queue, and is implemented using an internal-memory linear-space priority queue supporting Insert in amortized $\mathcal{O}(1)$ time and DeleteMin in amortized $\mathcal{O}(\lg n)$

time, where n is the number of elements in the min-buffer. The min-buffer can, e.g., be implemented using a binomial queue [31] or the implicit post-order heap by Harvey and Zatloukal [24]. The total space for the internal part is $\mathcal{O}(\mathcal{M})$.

INSERT(e) first compares e to the pivot p. If $e \leq p$, e is inserted in the min-buffer using the min-buffer's INSERT operation. Otherwise, e is appended to the insert-buffer. If the min-buffer overflows, i.e., gets size $\mathcal{M}+1$, we find a new pivot p' by applying the linear-time internal-memory median finding algorithm by Blum, Floyd, Pratt, Rivest, and Tarjan [5] to the elements in the min-buffer. The median, i.e., the $\left(\frac{\mathcal{M}}{2}+1\right)$th smallest element, becomes the new pivot p', the $\frac{\mathcal{M}}{2}$ larger elements and the old pivot p are appended to the insert-buffer, and the $\frac{\mathcal{M}}{2}$ smaller elements are inserted into a new empty min-buffer structure using its INSERT operations. If the insert-buffer overflows, i.e., gets size $> \mathcal{M}$ (due to the insertion of a single element or the old pivot p and $\frac{\mathcal{M}}{2}$ elements being moved from the min-buffer to the insert-buffer), we transfer $\mathcal{M}$ elements from the insert-buffer to the external part (see Section 4).

For DELETEMIN, if the min-buffer is non-empty, the smallest element is extracted from the min-buffer using the min-buffer's DELETEMIN operation and returned. Otherwise, the min-buffer is empty and the pivot p is the smallest element to be returned. To establish a new pivot p' and min-buffer, the $\frac{\mathcal{M}}{2}$ smallest elements are extracted from external memory (see Section 4) and transferred to the insert-buffer (except when the external part is empty). This ensures that the $\frac{\mathcal{M}}{2}$ smallest elements in the insert-buffer are now smaller than or equal to all elements in the external part. The $\frac{\mathcal{M}}{2}$th smallest element in the insert-buffer is selected as the new pivot p' in linear time using the internal-memory selection algorithm [5]. If the insert-buffer contains $< \frac{\mathcal{M}}{2}$ elements, we set the pivot p to be $+\infty$. The at most $\frac{\mathcal{M}}{2}-1$ elements smaller than or equal to the new pivot p' in the insert-buffer are deleted from the insert-buffer and inserted into the min-buffer structure using its INSERT operation. The old pivot p is returned as the extracted minimum.

Note that a transfer either moves $\mathcal{M}$ elements from the internal to the external part, or $\frac{\mathcal{M}}{2}$ elements from the external to the internal part, i.e., the external memory always contains a multiple of $\frac{\mathcal{M}}{2}$ elements. Note also that the size of the insert-buffer is unchanged during DELETEMIN, provided that there are elements in external memory. Otherwise, the number of elements in the insert-buffer decreases by $\frac{\mathcal{M}}{2}$.

Initially, the pivot $p = +\infty$, such that all elements are inserted into the min-buffer until the priority queue contains $\mathcal{M}+1$ elements, where a real element is selected as the pivot and $\frac{\mathcal{M}}{2}$ elements are moved to the insertion-buffer and a new min-buffer is created for the $\frac{\mathcal{M}}{2}$ smallest elements using repeated insertions.

4 The external-memory part

The external-memory part supports the insertion of a batch of $\mathcal{M}$ elements and the extraction of the $\mathcal{M}/2$ smallest elements. It consists of a set of Δ-*heaps*, where $\Delta = \frac{\mathcal{M}}{B} \geq 2$. A Δ-heap with height h is a tree where all leaves have depth h (the root having depth zero), and all non-leaves have exactly Δ children. Each node contains a buffer storing up to $\mathcal{M}$ elements. The elements must satisfy *extended heap order*, i.e., the elements in the buffer of a node u are smaller than or equal to any element in the buffers of the children of u. A Δ-heap is a generalization of a binary heap [32] and is very similar to the external-memory priority queue of Fadel, Jakobsen, Katajainen and Teuhola [18], except that we do not require buffers to be sorted (this would not be possible when insertions only perform amortized $\mathcal{O}(1)$ comparisons.) The buffers in a Δ-heap store elements in various degrees of sortedness to be able to achieve the stated insertion cost: non-leaves are *semi-sorted*, whereas leaves are *lazy semi-sorted* as discussed in Sections 5 and 6, respectively.

The Δ-heaps are maintained as a sequence of collections $\mathcal{C}_0, \ldots, \mathcal{C}_H$, where $\mathcal{C}_h$ contains all Δ-heaps with height h. We maintain the invariant (**I1**) that $|\mathcal{C}_h| < \Delta$ and that the total number of leaves in the Δ-heaps is $\leq \frac{I}{\mathcal{M}}$ (Lemma 7), where I is the total number of insertions. This invariant ensures that the maximum height H of a Δ-heap is $\leq \lfloor \log_\Delta \frac{I}{\mathcal{M}} \rfloor$. The buffers of all nodes must satisfy the following invariant (**I2**): The buffer of a node u stores at most $\mathcal{M}$ elements. If no descendant of u stores elements, there is no lower bound on the buffer size of u. Otherwise, the buffer of u contains at least $\frac{\mathcal{M}}{2}$ elements. Together with the extended heap order, this invariant implies that the $\frac{\mathcal{M}}{2}$ smallest elements in a Δ-heap are stored in the root buffer, except if the Δ-heap contains fewer than $\frac{\mathcal{M}}{2}$ elements, in which case all elements are stored in the root buffer.

When $\mathcal{M}$ elements are transferred from the internal to the external part, the $\mathcal{M}$ elements become a single node Δ-heap with height zero that is added to collection $\mathcal{C}_0$. If the collection is full, that is, $|\mathcal{C}_0| = \Delta$, then a new root r is created for a Δ-heap of height one with each leaf of $\mathcal{C}_0$ as its children. Invariant **I2** is established for r by recursively *pulling* the $\frac{\mathcal{M}}{2}$ smallest elements from its children and inserting them in the buffer (the pull operation is described below). This leaves $\mathcal{C}_0$ empty, and r is promoted to $\mathcal{C}_1$. As in a Δ-ary number system, this promotion may propagate up through the collections, repeatedly combining Δ Δ-heaps with height h to a Δ-heap with height $h+1$.

For a node with $< \frac{\mathcal{M}}{2}$ elements in its buffer, a *pull* operation moves $\frac{\mathcal{M}}{2}$ elements from its children to the buffer of the node while preserving extended heap order, using Lemma 5 in Section 5. Note that this will make the buffer contain at least $\frac{\mathcal{M}}{2}$ elements and less than $\mathcal{M}$. If a child buffer gets size $< \frac{\mathcal{M}}{2}$, we recursively apply the pull operation to the child, provided that not all subtrees below the child are exhausted.

To extract the $\frac{\mathcal{M}}{2}$ smallest elements from the external part, first, for each collection $\mathcal{C}_i$, the $\frac{\mathcal{M}}{2}$ smallest elements are found by considering the elements in the buffers of the roots by applying Lemma 5 to the roots of the collection. As the elements are just identified but not pulled from the roots, we call this a *virtual pull*. Second, the $\frac{\mathcal{M}}{2}$ smallest elements among the $(H+1)\frac{\mathcal{M}}{2}$ candidates from the virtual pulls are found by applying a linear-time selection algorithm, Corollary 3 below. These elements are removed from their respective roots and transferred to the internal part. For each root buffer now containing $< \frac{\mathcal{M}}{2}$ elements, we recursively pull $\frac{\mathcal{M}}{2}$ elements from its children, to the extent possible.

Note that the total number of elements in external memory is always a multiple of $\frac{\mathcal{M}}{2}$, but this is not necessarily true for each Δ-heap due to the extractions of the smallest $\frac{\mathcal{M}}{2}$ elements across all Δ-heaps.

5 Semi-sorted lists

Given two sequences X and Y, we say $X \preceq Y$, if for every $x \in X$ and $y \in Y$: $x \leq y$. Note that $X \preceq Y$ implies no specific order within each individual sequence X or Y.

A buffer in external memory is a linked list of blocks $b_1, b_2, \ldots, b_\delta$, where each block contains B elements, except possibly the first and last blocks, which contain $\leq B$ elements. In the following, we simply denote such a linked list as a *list*. We define a list to be *semi-sorted* if $b_i \preceq b_{i+1}$ for every $1 \leq i < \delta$. All buffers of non-leaf nodes in a Δ-heap are stored as semi-sorted lists. The following are useful tools for working with semi-sorted buffers.

Blum, Floyd, Pratt, Rivest, and Tarjan [5] proved that the kth smallest element in a list with N elements can be found using $\mathcal{O}(N)$ comparisons using a double recursion making repeated use of scanning lists. Analyzed in the external-memory model, this immediately implies that selection can be performed in a linear number of I/Os, and we have the following corollary.

▶ **Corollary 3** (Blum et al. [5]). *Given a list of N elements and an integer $1 \leq k \leq N$, the list can be partitioned into two lists X and Y, with $|X| = k$ and $X \preceq Y$, using $\mathcal{O}(N)$ comparisons and $\mathcal{O}\left(\frac{N}{B}\right)$ I/Os.*

▶ **Lemma 4** (Constructing semi-sorted buffer). *An unsorted list of $B\delta \leq \mathcal{M}$ elements can be converted into a semi-sorted list using $\mathcal{O}(B\delta \lg \delta)$ comparisons and $\mathcal{O}(\delta)$ I/Os.*

Proof. Recursively apply [5] to partition an unsorted list with δB elements into two subproblems with $\lceil \delta/2 \rceil B$ and $\lfloor \delta/2 \rfloor B$ elements, until the subproblems are of size exactly B. The result follows since there are $\lceil \lg \delta \rceil$ levels of recursion with a linear number of comparisons per level, and we only perform I/Os to read the initial list into internal memory and to write the final list to external memory, since the problem fits in internal memory. ◀

▶ **Lemma 5** (Pulling from semi-sorted lists). *Given at most $\Delta = \frac{\mathcal{M}}{B}$ semi-sorted lists, a semi-sorted list of the $\frac{\mathcal{M}}{2}$ smallest elements of their union can be computed in $\mathcal{O}(\mathcal{M} \lg \Delta)$ comparisons and $\mathcal{O}\left(\frac{\mathcal{M}}{B}\right)$ I/Os.*

Proof. Read the first block from each semi-sorted list into internal memory and insert the largest element of each block into a binary min-heap Q [32]. Then, delete the smallest element from Q, read the next block from the corresponding semi-sorted list, and insert the largest element of this block into Q. Let X denote the elements in blocks whose maximums have been deleted from Q, and Y the elements in blocks whose maximums are in Q. Repeat this process of reading blocks until $|X| \geq \frac{\mathcal{M}}{2}$. In total, $\Theta\left(\frac{\mathcal{M}}{B}\right)$ blocks are read, where the maximum is computed of each block and each block requires $\mathcal{O}(1)$ heap operations on Q, where $|Q| \leq \Delta$. This initial phase requires a total of $\mathcal{O}\left(\mathcal{M} + \frac{\mathcal{M}}{B} \cdot \lg \Delta\right)$ comparisons and $\Theta\left(\frac{\mathcal{M}}{B}\right)$ I/Os.

Let x be the last element deleted from Q. We have $\max(X) \leq x$, since when the lists are semi-sorted, elements will be extracted in non-decreasing order from Q. From each list, we have a block represented in Q with maximum $\geq x$, so all remaining unread elements in the lists are also $\geq x$. Thus, the $\frac{\mathcal{M}}{2}$ smallest elements can be found by a single partition of $X \cup Y$ using $\mathcal{O}(\mathcal{M})$ comparisons and $\mathcal{O}\left(\frac{\mathcal{M}}{B}\right)$ I/Os per Corollary 3. They can then be converted to a semi-sorted list using additional $\mathcal{O}(\mathcal{M} \lg \Delta)$ comparisons per Lemma 4. ◀

6 Lazy semi-sorted lists

Since I insertions can create $\Theta(I/\mathcal{M})$ leaf buffers of size $\mathcal{M}$, they cannot all be semi-sorted, as that would require $\Theta(I \lg \Delta)$ comparisons. The lower bound of $\Omega(\mathcal{M} \lg \Delta)$ comparisons for the multiple selection problem for each buffer is due to Dobkin and Munro [2, Theorem 1]. Instead, a leaf buffer with $\mathcal{M}$ elements is stored as a *lazy semi-sorted* list, see Figure 2. A lazy semi-sorted list stores the $\mathcal{M}$ elements as a sequence of *chunks* $c_1 \preceq c_2 \preceq \cdots \preceq c_d$, where $d = \left\lceil \lg \frac{\mathcal{M}}{B} \right\rceil + 1$ and, initially, the elements inside a chunk are stored in arbitrary order. Chunk c_1 stores B elements, c_i stores $B2^{i-2}$ elements for $2 \leq i < d$, and c_d stores the largest $\mathcal{M} - B2^{d-2}$ elements. If each chunk is converted into a semi-sorted list, and all chunks are concatenated, we have a semi-sorted list for the buffer. The basic idea is to exploit that a Δ-heap always accesses the blocks of a semi-sorted buffer left-to-right, implying that we can delay expanding a chunk into a semi-sorted sequence of blocks using Lemma 4 until the first block of the (semi-sorted version of the) chunk is accessed, hence the name lazy semi-sorted. The initial partitioning of a buffer into chunks is done by recursively partitioning the half with the smaller elements using Corollary 3.

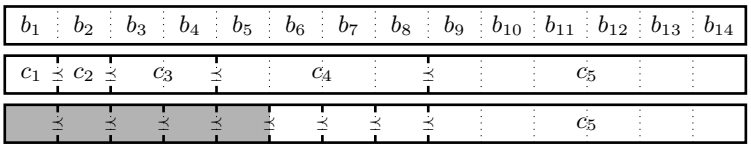


Figure 2 A lazy semi-sorted list. Top: Initial unsorted list of blocks $b_1, \ldots, b_{\mathcal{M}/B}$. Middle: Initial chunks $c_1 \preceq \cdots \preceq c_5$. Bottom: The state after the first five blocks $b_1, \ldots, b_5$ in the semi-sorted list have been accessed, i.e., the chunks $c_1, \ldots, c_4$ have been made into semi-sorted sublists.

▶ **Lemma 6** (Lazy access cost). *Initializing a lazy semi-sorted list of $\mathcal{M}$ elements requires $\mathcal{O}(\mathcal{M})$ comparisons and $\mathcal{O}(\frac{\mathcal{M}}{B})$ I/Os. The cost of accessing the first $k \leq \frac{\mathcal{M}}{B}$ blocks of the lazy semi-sorted list requires $\mathcal{O}(Bk \lg k)$ comparisons and $\mathcal{O}(k)$ I/Os.*

Proof. To initialize the lazy semi-sorted list, all $\mathcal{M}$ elements are first read into internal memory using $\mathcal{O}(\frac{\mathcal{M}}{B})$ I/Os. No further I/Os are performed, except for eventually writing the output to external memory. The initial chunks, with elements in arbitrary order, are constructed one at a time by repeatedly applying Corollary 3 to find the appropriate number of largest elements. By constructing the chunks in the order $c_d, c_{d-1}, \ldots c_0$ from the largest to the smallest, the number of remaining elements decreases geometrically, leading to a total of $\mathcal{O}(\mathcal{M})$ comparisons.

If the k first blocks of elements have been accessed, then chunks $c_1, \ldots, c_{\lceil \lg k \rceil + 1}$ have been semi-sorted, i.e., fewer than $2k$ blocks. Note that c_1 and c_2 only consist of a single block, i.e., they are already semi-sorted at initialization. By applying Lemma 4, semi-sorting the remaining $\lceil \lg k \rceil - 1$ accessed chunks requires $\mathcal{O}\left(\sum_{i=1}^{\lceil \lg k \rceil - 1} B2^i \lg 2^i\right) = \mathcal{O}(Bk \lg k)$ comparisons and $\mathcal{O}\left(\sum_{i=1}^{\lceil \lg k \rceil - 1} 2^i\right) = \mathcal{O}(k)$ I/Os. ◀

The idea of lazy semi-sorted buffers is that they allow us to charge the non-linear comparison cost of constructing a semi-sorted buffer to the pull operations on the buffer, in particular, the pull operations triggered by deletions.

7 Analysis

When Insert and DeleteMin can be performed entirely within the internal part, no I/Os are performed. The number of comparisons is $\mathcal{O}(1)$ for Insert and $\mathcal{O}(\lg \min(\mathcal{M}, I))$ for DeleteMin, plus the cost of moving elements from the min-buffer to the insert-buffer. If an insertion causes the min-buffer to overflow, we spend $\mathcal{O}(\mathcal{M})$ comparisons moving elements to the insert-buffer and building a new min-buffer. This can only happen once every $\frac{\mathcal{M}}{2}$ insertion, so the total cost for this is $\mathcal{O}(I)$ comparisons and no I/Os.

A transfer either moves $\mathcal{M}$ elements from the internal to the external part, or $\frac{\mathcal{M}}{2}$ elements from the external to the internal part. The following lemma bounds the number of transfers in each direction in terms of I and D.

▶ **Lemma 7** (Bounding transfers). *The number of transfers from the internal to the external part is $\leq \frac{I}{\mathcal{M}}$. The number of transfers from the external to the internal part is $\leq \frac{2D}{\mathcal{M}}$.*

Proof. We first consider the number of transfers from the internal to the external part. Let $\phi \geq 0$ be the number of elements in the min-buffer beyond the first $\frac{\mathcal{M}}{2}$ elements plus the number of elements in the insert-buffer. Insertions cause ϕ to increase by at most one, and deletions do not cause ϕ to increase. A transfer to the external part decreases ϕ by $\mathcal{M}$, whereas a transfer to the internal part of $\frac{\mathcal{M}}{2}$ elements leaves ϕ unchanged, since the min-buffer gets size $\frac{\mathcal{M}}{2} - 1$ and the size of the insert-buffer does not change. It follows that ϕ only

increases during the I insertions, where the total increase is $\leq I$, and each transfer to the external part decreases ϕ by $\mathcal{M}$, i.e., the number of transfers to the external part is at most $\frac{I}{\mathcal{M}}$.

The size of the min-buffer can only decrease below $\frac{\mathcal{M}}{2}$ due to a deletion, where it decreases by one. (When the size of the min-buffer decreases because half of the elements are moved to the insert-buffer, the min-buffer changes size from $\mathcal{M}+1$ to $\frac{\mathcal{M}}{2}$.) A transfer to the internal part occurs when the size of the min-buffer is zero before the deletion, but then the min-buffer has size $\frac{\mathcal{M}}{2}-1$ after the deletion and the next $\frac{\mathcal{M}}{2}-1$ deletions do not cause a transfer to the internal part. It follows that at most every $\frac{\mathcal{M}}{2}$ deletion can cause a transfer to the internal part. The first transfer to the internal part can only happen after a transfer to the external part has occurred, which first can happen after the pivot $p \neq +\infty$, where the size of the min-buffer is $\frac{\mathcal{M}}{2}$, i.e., there must have been at least $\frac{\mathcal{M}}{2}$ deletions before the first transfer to the internal part. It follows that the number of transfers to the internal part is at most $\frac{2D}{\mathcal{M}}$. ◀

The key part of the analysis is threefold. First, we bound the cost to restore invariants **I1** and **I2** whenever elements are removed from the internal nodes of the external part. Second, we bound the cost of *accessing* the lazy semi-sorted leaves. The access cost of a leaf refers to the cost of semi-sorting its chunks. Third, we bound the cost involved with transfers between internal and external memory. We define the *height* of a node in a Δ-heap to be the height of the subtree rooted at the node (leaves have height zero).

▶ **Lemma 8** (Total pull cost excluding leaf access). *The total number of pulls is $\mathcal{O}\big(\frac{I}{\mathcal{M}\Delta}+\frac{DH}{\mathcal{M}}\big)$ and their total cost is $\mathcal{O}\Big(I\frac{\lg\Delta}{\Delta}+D\lg\frac{I}{\mathcal{M}}\Big)$ comparisons and $\mathcal{O}\big(\frac{I}{\mathcal{M}}+\frac{DH}{B}\big)$ I/Os, excluding the cost of accessing the leaves.*

Proof. A pull at a node increases the height of $\frac{\mathcal{M}}{2}$ elements from the buffers of the children by one, except the last pull at a node that exhausts all subtrees at the children, where at most $\frac{\mathcal{M}}{2}$ elements have their height increased by one. By Lemma 7, at most $\frac{I}{\mathcal{M}}$ leaves are created, i.e., the maximum height of a Δ-heap is $H \leq \log_\Delta \frac{I}{\mathcal{M}}$. For a height h, $1 \leq h \leq H$, we bound the number of pulls P_h to nodes with height h by bounding the number of elements that could have been moved to a height $\geq h$ or transferred to the internal memory, the latter being bounded by D by Lemma 7. Since there are $\leq \frac{I}{\mathcal{M}}$ leaves, the number of nodes at height h is $\leq \frac{I}{\mathcal{M}\Delta^h}$, implying that the number of elements that could ever have been pulled to height h (and possibly further up) is at most $D+\mathcal{M}\sum_{j=h}^{H}\frac{I}{\mathcal{M}\Delta^j}$. Since each non-exhausting pull moves $\frac{\mathcal{M}}{2}$ elements one level up, and each node with height h can have one final pull exhausting all children, we get that the final number of pulls to height h is $P_h \leq \Big(D+\mathcal{M}\sum_{j=h}^{H}\frac{I}{\mathcal{M}\Delta^j}\Big) / \frac{\mathcal{M}}{2}+\frac{I}{\mathcal{M}\Delta^h} = \mathcal{O}\big(\frac{I}{\mathcal{M}\Delta^h}+\frac{D}{\mathcal{M}}\big)$, as $\Delta \geq 2$.

The total number of pulls is then $\sum_{h=1}^{H} P_h = \mathcal{O}\big(\frac{I}{\mathcal{M}\Delta}+\frac{DH}{\mathcal{M}}\big)$. Excluding the cost of accessing the leaves, we may treat each pull as being from semi-sorted lists. Applying Lemma 5, the total number of comparisons is $\mathcal{O}\big(\big(\frac{I}{\mathcal{M}\Delta}+\frac{D}{\mathcal{M}}\log_\Delta\frac{I}{\mathcal{M}}\big)\mathcal{M}\lg\Delta\big) = \mathcal{O}\Big(I\frac{\lg\Delta}{\Delta}+D\lg\frac{I}{\mathcal{M}}\Big)$ and the number of I/Os is $\mathcal{O}\big(\big(\frac{I}{\mathcal{M}\Delta}+\frac{DH}{\mathcal{M}}\big)\frac{\mathcal{M}}{B}\big) = \mathcal{O}\big(\frac{I}{\mathcal{M}}+\frac{DH}{B}\big)$ I/Os. ◀

▶ **Lemma 9** (Total leaf access cost). *The total cost of constructing the lazy semi-sorted leaves and accessing them is $\mathcal{O}(I+D\lg\Delta)$ comparisons and $\mathcal{O}\big(\frac{I}{B}+\frac{D}{B}\big)$ I/Os.*

Proof. As described in the proof of Lemma 8, the number of pulls from the leaves is $P_1 = \mathcal{O}\big(\frac{I}{\mathcal{M}\Delta}+\frac{D}{\mathcal{M}}\big)$. Virtual pulls may, however, also access elements in the leaves. Since each transfer to the internal part causes one virtual pull from $\mathcal{C}_0$, the number of virtual pulls from the leaves is bounded by $\frac{2D}{\mathcal{M}}$ by Lemma 7. The total number of pulls and virtual pulls from the leaves is $\mathcal{O}\big(\frac{I}{\mathcal{M}\Delta}+\frac{D}{\mathcal{M}}\big)$.

By Lemma 6, the cost for the initial construction of the at most $\frac{I}{\mathcal{M}}$ leaves as lazy semi-sorted lists is $\mathcal{O}(I)$ comparisons and $\mathcal{O}\left(\frac{I}{B}\right)$ I/Os. Since, by Lemma 5, each pull and virtual pull performs $\mathcal{O}(\mathcal{M}\lg\Delta)$ comparisons and accesses $\mathcal{O}\left(\frac{\mathcal{M}}{B}\right)$ blocks, we have, by Lemma 6, that at the leaves, the total number of comparisons is $\mathcal{O}\left(I+\left(\frac{I}{\mathcal{M}\Delta}+\frac{D}{\mathcal{M}}\right)\mathcal{M}\lg\Delta\right)=\mathcal{O}(I+D\lg\Delta)$ comparisons and the total number of I/Os is $\mathcal{O}\left(\frac{I}{B}+\left(\frac{I}{\mathcal{M}\Delta}+\frac{D}{\mathcal{M}}\right)\frac{\mathcal{M}}{B}\right)=\mathcal{O}\left(\frac{I}{B}+\frac{D}{B}\right)$ I/Os. ◀

▶ **Lemma 10** (Cost of transfers excluding pulls and leaf accessing). *The total cost of transfers between the internal and external part is $\mathcal{O}\left(D\lg\frac{I}{\mathcal{M}}\right)$ comparisons and $\mathcal{O}\left(\frac{I}{B}+\frac{D}{B}\log_\Delta\frac{I}{\mathcal{M}}\right)$ I/Os, excluding the cost of pulling to restore the invariants and to access the leaves.*

Proof. By Lemma 7, there are at most $\frac{I}{\mathcal{M}}$ transfers to the external part, each moving $\mathcal{M}$ elements from the input-buffer to a new leaf, with no comparisons and $\mathcal{O}\left(\frac{\mathcal{M}}{B}\right)$ I/Os.

By Lemma 7, there are at most $\frac{2D}{\mathcal{M}}$ transfers to the internal part. Each transfer involves finding the $\mathcal{M}$ smallest elements among the elements in the buffers of all roots in all collections. First, for each collection, we perform a virtual pull from the semi-sorted roots. We exclude the access cost for the leaves and treat them as semi-sorted. Applying Lemma 5 for each of the $\mathcal{O}(H)=\mathcal{O}\left(\log_\Delta\frac{I}{\mathcal{M}}\right)$ collections leads to a cost of $\mathcal{O}\left(\mathcal{M}\lg\Delta\log_\Delta\frac{I}{\mathcal{M}}\right)=\mathcal{O}\left(\mathcal{M}\lg\frac{I}{\mathcal{M}}\right)$ comparisons and $\mathcal{O}\left(\frac{\mathcal{M}}{B}\log_\Delta\frac{I}{\mathcal{M}}\right)$ I/Os. Then among these $\mathcal{O}\left(\mathcal{M}\log_\Delta\frac{I}{\mathcal{M}}\right)$ elements, we find the $\frac{\mathcal{M}}{2}$ smallest elements by applying Corollary 3. These elements are then transferred to the insert-buffer and removed from their respective roots. This may trigger pulls to restore the invariants, the cost of which we exclude here and is accounted for in Lemma 8. These operations do not asymptotically increase the total cost, which is $\mathcal{O}\left(D\lg\frac{I}{\mathcal{M}}\right)$ comparisons and $\mathcal{O}\left(\frac{D}{B}\log_\Delta\frac{I}{\mathcal{M}}\right)$ I/Os. Finally, a transfer to the internal part causes $\mathcal{O}(\mathcal{M})$ comparisons to build the new min-buffer and insert-buffer, which sums up to $\mathcal{O}(D)$ comparisons for all transfers to the internal part. ◀

Proof of Lemma 2. Summarizing the number of comparisons and I/Os performed, the total number of comparisons is

$$\mathcal{O}\Bigg(\underbrace{(I+D\lg\mathcal{M})}_{\text{internal memory}}+\underbrace{\left(I\frac{\lg\Delta}{\Delta}+D\lg\frac{I}{\mathcal{M}}\right)}_{\text{pulls}}+\underbrace{(I+D\lg\Delta)}_{\text{leaves}}+\underbrace{\left(D\lg\frac{I}{\mathcal{M}}\right)}_{\text{transfers}}\Bigg)=\mathcal{O}(I+D\lg I)\ ,$$

and the total number of I/Os is

$$\mathcal{O}\Bigg(\underbrace{\left(\frac{I}{\mathcal{M}}+\frac{DH}{B}\right)}_{\text{pulls}}+\underbrace{\left(\frac{I}{B}+\frac{D}{B}\right)}_{\text{leaves}}+\underbrace{\left(\frac{I}{B}+\frac{D}{B}\log_\Delta\frac{I}{\mathcal{M}}\right)}_{\text{transfers}}\Bigg)=\mathcal{O}\left(\frac{I}{B}+\frac{D}{B}\log_\Delta\frac{I}{\mathcal{M}}\right)\ .$$

Lemma 2 follows from $\mathcal{M}=\Theta(M)$ and $\Delta=\frac{\mathcal{M}}{B}$. ◀

8 Dependence on current priority queue size

In Sections 3–7, I denotes the total number of insertions performed, allowing the current size N to be arbitrarily smaller than I. Since the comparison and I/O bounds of Lemma 2 are dependent on I, the bounds are not a function of the current number of elements N in the priority queue. To achieve this, we apply the standard technique of *global rebuilding* [28, Chapter 5].

Proof of Theorem 1. Let N_0 denote the size of the priority queue at the last global rebuilding of the priority queue, and let I_0 and D_0 denote the number of insertions and deletions since the last global rebuilding, respectively. We rebuild the priority queue whenever $I_0+D_0>\frac{N_0}{2}$.

Rebuilding a priority queue containing N elements is performed by scanning over the structure, collecting the N elements into a list, and then repeatedly inserting these into a new empty priority queue using Insert. By Lemma 2, the resulting priority queue is constructed using $\mathcal{O}(N)$ comparisons and $\mathcal{O}\left(\frac{N}{B}\right)$ I/Os (the previous structure can be scanned in $\mathcal{O}\left(\frac{N}{B}\right)$ I/Os, since we argue below that $N = \Theta(N_0)$ and at most $2\frac{N_0+I_0}{\mathcal{M}} = \mathcal{O}\left(\frac{N}{\mathcal{M}}\right)$ buffers have been created in external memory since the last global rebuilding).

Together with Lemma 2, the previous rebuilding and the subsequent I_0 insertions and D_0 deletions, causing a total of $N_0 + I_0$ insertions and D_0 deletions, imply a total comparison cost of $\mathcal{O}(N_0 + I_0 + D_0 \lg(N_0 + I_0)) = \mathcal{O}(N_0 + D_0 \lg N_0)$ and a total I/O cost of $\mathcal{O}\left(\frac{N_0}{B} + \frac{1}{B}\left((N_0 + I_0) + D_0 \log_{M/B} \frac{N_0+I_0}{B}\right)\right) = \mathcal{O}\left(\frac{N_0}{B} + \frac{D_0}{B} \log_{M/B} \frac{N_0}{B}\right)$, since $I_0 \leq \frac{N_0}{2}$. We charge the linear cost to the at least $\frac{N_0}{3}$ priority queue operations between the last two global rebuildings, and the remaining cost to the deletions since the last global rebuilding. While no global rebuilding is triggered, we have $\frac{1}{2}N_0 \leq N \leq \frac{3}{2}N_0$, i.e., we have $N = \Theta(N_0)$ between two global rebuildings. Theorem 1 follows. ◀

9 Conclusion and open problems

The external-memory priority queue presented in this paper supports insertions with asymptotically fewer comparisons and I/Os than previous results. This is particularly beneficial in settings such as event-driven simulations or other incremental computations where execution may terminate before all elements have been extracted from the queue and the cost of insertions dominates. It should be noted that the amortized internal computation time required by our construction is asymptotically the same as the number of comparisons performed (which also holds for the black boxes used by the construction [5, 24, 31]).

The presented data structure is inherently cache-aware. This raises a natural question: does there exist a cache-oblivious priority queue that achieves the same amortized bounds on comparisons and I/Os?

The structure is also highly amortized (e.g., due to global rebuilding, transfers between internal and external memory, recursive pulling, lazy semi-sorted buffers), and a single Insert or DeleteMin operation might require $\Theta(N/B)$ I/Os in the worst-case. Another natural question, is whether it is possible to achieve a solution with worst-case guarantees, e.g., such that a window of B priority queue operations requires worst-case $\mathcal{O}(B \lg N)$ comparisons and $\mathcal{O}\left(\log_{M/B} \frac{N}{B}\right)$ I/Os, while maintaining the improved amortized bounds for Insert.

References

1 Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.

2 David P. Dobkin andJ. Ian Munro. Optimal time minimal space selection algorithms. *Journal of the ACM*, 28(3):454–461, 1981. doi:10.1145/322261.322264.

3 Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. doi:10.1007/S00453-003-1021-X.

4 Lars Arge, Michael A. Bender, Erik D. Demaine, Bryan Holland-Minkley, and J. Ian Munro. An optimal cache-oblivious priority queue and its application to graph algorithms. *SIAM Journal of Computing*, 36(6):1672–1695, 2007. doi:10.1137/S0097539703428324.

5 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

6 Klaus Brengel, Andreas Crauser, Paolo Ferragina, and Ulrich Meyer. An experimental study of priority queues in external memory. In Jeffrey Scott Vitter and Christos D. Zaroliagis, editors, *Algorithm Engineering, 3rd International Workshop, WAE '99, London, UK, July 19-21, 1999, Proceedings*, volume 1668 of *Lecture Notes in Computer Science*, pages 346–360. Springer, 1999. doi:10.1007/3-540-48318-7_27.

7 Gerth Stølting Brodal. A survey on priority queues. In Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 150–163. Springer, 2013. doi:10.1007/978-3-642-40273-9_11.

8 Gerth Stølting Brodal and Rolf Fagerberg. Cache oblivious distribution sweeping. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 426–438. Springer, 2002. doi:10.1007/3-540-45465-9_37.

9 Gerth Stølting Brodal and Rolf Fagerberg. Funnel heap — A cache oblivious priority queue. In Prosenjit Bose and Pat Morin, editors, *Algorithms and Computation, 13th International Symposium, ISAAC 2002 Vancouver, BC, Canada, November 21-23, 2002, Proceedings*, volume 2518 of *Lecture Notes in Computer Science*, pages 219–228. Springer, 2002. doi:10.1007/3-540-36136-7_20.

10 Gerth Stølting Brodal and Rolf Fagerberg. On the limits of cache-obliviousness. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 307–315. ACM, 2003. doi:10.1145/780542.780589.

11 Gerth Stølting Brodal, Rolf Fagerberg, Ulrich Meyer, and Norbert Zeh. Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths. In Torben Hagerup and Jyrki Katajainen, editors, *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Humlebaek, Denmark, July 8-10, 2004, Proceedings*, volume 3111 of *Lecture Notes in Computer Science*, pages 480–492. Springer, 2004. doi:10.1007/978-3-540-27810-8_41.

12 Gerth Stølting Brodal and Jyrki Katajainen. Worst-case external-memory priority queues. In Stefan Arnborg and Lars Ivansson, editors, *Algorithm Theory - SWAT '98, 6th Scandinavian Workshop on Algorithm Theory, Stockholm, Sweden, July, 8-10, 1998, Proceedings*, volume 1432 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 1998. doi:10.1007/BFB0054359.

13 Mark R. Brown. Implementation and analysis of binomial queue algorithms. *SIAM Journal of Computing*, 7(3):298–319, 1978. doi:10.1137/0207026.

14 Svante Carlsson, J. Ian Munro, and Patricio V. Poblete. An implicit binomial queue with constant insertion time. In Rolf G. Karlsson and Andrzej Lingas, editors, *SWAT 88, 1st Scandinavian Workshop on Algorithm Theory, Halmstad, Sweden, July 5-8, 1988, Proceedings*, volume 318 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1988. doi:10.1007/3-540-19487-8_1.

15 Rezaul Alam Chowdhury and Vijaya Ramachandran. Cache-oblivious shortest paths in graphs using buffer heap. In Phillip B. Gibbons and Micah Adler, editors, *SPAA 2004: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 27-30, 2004, Barcelona, Spain*, pages 245–254. ACM, 2004. doi:10.1145/1007912.1007949.

16 James R. Driscoll, Harold N. Gabow, Ruth Shrairman, and Robert Endre Tarjan. Relaxed heaps: An alternative to fibonacci heaps with applications to parallel computation. *Communications of the ACM*, 31(11):1343–1354, 1988. doi:10.1145/50087.50096.

17 Kasper Eenberg, Kasper Green Larsen, and Huacheng Yu. Decreasekeys are expensive for external memory priority queues. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1081–1093. ACM, 2017. doi:10.1145/3055399.3055437.

18 R. Fadel, K. V. Jakobsen, Jyrki Katajainen, and Jukka Teuhola. Heaps and heapsort on secondary storage. *Theoretical Computer Science*, 220(2):345–362, 1999. doi:10.1016/S0304-3975(99)00006-7.

19 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

20 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 285–298. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814600.

21 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. *ACM Transactions on Algorithms*, 8(1):4:1–4:22, 2012. doi:10.1145/2071379.2071383.

22 Bernhard Haeupler, Richard Hladík, Václav Rozhon, Robert E. Tarjan, and Jakub Tetek. Universal optimality of Dijkstra via beyond-worst-case heaps. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2099–2130. IEEE, 2024. doi:10.1109/FOCS61266.2024.00125.

23 Bernhard Haeupler, Richard Hladík, Václav Rozhon, Robert E. Tarjan, and Jakub Tetek. Bidirectional Dijkstra's algorithm is instance-optimal. In Ioana Oriana Bercea and Rasmus Pagh, editors, *2025 Symposium on Simplicity in Algorithms, SOSA 2025, New Orleans, LA, USA, January 13-15, 2025*, pages 202–215. SIAM, 2025. doi:10.1137/1.9781611978315.16.

24 Nicholas J. A. Harvey and Kevin C. Zatloukal. The post-order heap. In *Proceedings Third International Conference on Fun with Algorithms (FUN 2004), Elba, Italy, May 2004*, 2004. URL: http://people.csail.mit.edu/nickh/Publications/PostOrderHeap/FUN04-PostOrderHeap.pdf.

25 John Iacono, Riko Jacob, and Konstantinos Tsakalidis. External memory priority queues with decrease-key and applications to graph algorithms. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 60:1–60:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ESA.2019.60.

26 Shunhua Jiang and Kasper Green Larsen. A faster external memory priority queue with decreasekeys. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1331–1343. SIAM, 2019. doi:10.1137/1.9781611975482.81.

27 Vijay Kumar and Eric J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proceedings of the Eighth IEEE Symposium on Parallel and Distributed Processing, SPDP 1996, New Orleans, Louisiana, USA, October 23-26, 1996*, pages 169–176. IEEE Computer Society, 1996. doi:10.1109/SPDP.1996.570330.

28 Mark H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983. doi:10.1007/BFB0014927.

29 Nir Shavit and Itay Lotan. Skiplist-based concurrent priority queues. In *Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000*, pages 263–268. IEEE, 2000. doi:10.1109/IPDPS.2000.845994.

30 Mikkel Thorup. On RAM priority queues. *SIAM Journal on Computing*, 30(1):86–109, 2000. doi:10.1137/S0097539795288246.

31 Jean Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 21(4):309–315, 1978. doi:10.1145/359460.359478.

32 John William Joseph Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964. doi:10.1145/512274.512284.

On the Complexity of Knapsack Under Explorable Uncertainty: Hardness and Algorithms

Jens Schlöter
Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands

Abstract

In the *knapsack problem under explorable uncertainty*, we are given a knapsack instance with uncertain item profits. Instead of having access to the precise profits, we are only given *uncertainty intervals* that are guaranteed to contain the corresponding profits. The actual item profit can be obtained via a *query*. The goal of the problem is to adaptively query item profits until the revealed information suffices to compute an optimal (or approximate) solution to the underlying knapsack instance. Since queries are costly, the objective is to minimize the number of queries.

In the offline variant of this problem, we assume knowledge of the precise profits and the task is to compute a query set of minimum cardinality that a third party without access to the profits could use to identify an optimal (or approximate) knapsack solution. We show that this offline variant is complete for the second-level of the polynomial hierarchy, i.e., Σ_2^p-complete, and cannot be approximated within a non-trivial factor unless $\Sigma_2^p = \Delta_2^p$. Motivated by these strong hardness results, we consider a "resource-augmented" variant of the problem where the requirements on the query set computed by an algorithm are less strict than the requirements on the optimal solution we compare against. More precisely, a query set computed by the algorithm must reveal sufficient information to identify an approximate knapsack solution, while the optimal query set we compare against has to reveal sufficient information to identify an optimal solution. We show that this resource-augmented setting allows interesting non-trivial algorithmic results.

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

Keywords and phrases Explorable uncertainty, knapsack, queries, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.6

Related Version *Full Version*: `https://arxiv.org/abs/2507.02657` [29]

Funding This work was partially supported by the research project *Optimization for and with Machine Learning (OPTIMAL)*, funded by the Dutch Research Council (NWO), grant number OCENW.GROOT.2019.015.

Acknowledgements Parts of the work on this project have been done while the author was affiliated with the University of Bremen. The author thanks Nicole Megow for initial discussions.

1 Introduction

The field of *explorable uncertainty* considers optimization problems with uncertainty in the numeric input parameters. Initially, the precise values of the uncertain parameters are unknown. Instead, for each uncertain parameter, we are given an *uncertainty interval* that contains the precise value of that parameter. Each uncertain parameter can be queried to reveal its precise value. The goal is to adaptively query uncertain parameters until we have sufficient information to solve the underlying optimization problem.

In this paper, we consider *knapsack under explorable uncertainty* (KnapExp) with uncertain item profits. That is, we are given a set of items $\mathcal{I}$ and a knapsack capacity $B \in \mathbb{N}$. Each item $i \in \mathcal{I}$ has a known weight $w_i \in \mathbb{N}$ and an uncertain profit $p_i \in \mathbb{R}$ that is initially hidden within the known uncertainty interval I_i, i.e., $p_i \in I_i$. A *query* of an item i reveals the profit p_i. Our goal is to compute a set $P \subseteq \mathcal{I}$ of items with $w(P) := \sum_{i \in P} w_i \leq B$ that

© Jens Schlöter;
licensed under Creative Commons License CC-BY 4.0

maximizes the profit $p(P) := \sum_{i \in P} p_i$. We refer to this problem as the *underlying knapsack problem*. Since the profits are initially hidden within their uncertainty intervals, we do not always have sufficient information to compute an optimal or even approximate solution for the underlying knapsack problem. Instead, an algorithm for KnapExp can adaptively query items to reveal their profits until the revealed information suffices to compute an optimal solution for the underlying knapsack instance. As queries are costly, the goal is to minimize the number of queries.

The *offline version*, sometimes also called *verification problem*, of knapsack under explorable uncertainty (OfflineKnapExp) assumes full initial access to the profits p_i and asks for a *query set* $Q \subseteq \mathcal{I}$ of minimum cardinality such that access to the profits of the items in Q and access to the uncertainty intervals of the items in $\mathcal{I} \setminus Q$ suffices to compute an optimal solution to the underlying knapsack instance, independent of what the precise profits of the items in $\mathcal{I} \setminus Q$ are. In this work, we mainly focus on studying the offline version of knapsack under explorable uncertainty. Most commonly, problems under explorable uncertainy are studied in an *adversarial online setting*, where the uncertain values are unknown, query outcomes are returned in a worst-case manner, and algorithms are compared against the optimal solution for the corresponding offline version by using competitive analysis. The complexity of the offline version is a natural barrier for efficiently solving the online version.

So far, most problems that have been studied under explorable uncertainty have an underlying problem that belongs to the complexity class P, i.e., can be solved in polynomial time. The seminal work by Kahan [20] on computing the minimum in a set of uncertain values was followed by works on computing the k-th smallest uncertain value [16, 20], computing a minimum spanning tree with uncertain edge weights [10, 12, 14, 24, 25, 28], sorting [13, 19], shortest path [15], finding the cheapest set in a given family of sets [11, 26], simple geometric problems [7], stable matchings [2], and other selection problems [3, 13]. If we remove the explorable uncertainty aspect, then all of these problems can be solved in polynomial time.

Even tough these underlying problems are in P, the offline versions of the corresponding problems under explorable uncertainty are often NP-hard. For instance, the offline version of identifying the set of maximal points under explorable uncertainty is NP-hard [8], the offline version of the selection problem in [3, 13] is NP-hard, and the offline version of the minimum-spanning tree problem under vertex uncertainty is NP-hard [10]. The offline version of selecting the cheapest set is NP-hard [11] and even hard to approximate within a factor of $o(\log m)$, where m is the number of sets [26]. Similarly, the offline version of stable matching under uncertainty is NP-hard to approximate [2]. For all of these problems, adding the layer of explorable uncertainty increases the complexity from polynomial-time solvable to NP-hard and leads to interesting algorithmic challenges even tough the underlying problems are easy. However, this observation also raises the following question:

> If the underlying problem is already NP-hard, does adding the layer of explorable uncertainty still increase the complexity?

As a first main result, we answer this question in the affirmative for the offline version of knapsack under explorable uncertainty. More precisely, we show that OfflineKnapExp is complete for the second level of the polynomial hierarchy, i.e., Σ_2^p-complete. We even show that, under a certain conjecture ($\Sigma_2^p \neq \Delta_2^p$), no $n^{1-\epsilon}$-approximation is possible for any $\epsilon > 0$, where n is the number of items. The latter can be seen as a natural next step from the inapproximability result given in [26]: They show that approximating the offline version of the cheapest set problem is hard to approximate within a factor of $o(\log m)$ by exploiting that it is equivalent to solving a covering integer linear program (ILP) with m constraints,

whereas we show our inapproximability result by exploiting that offline KnapExp can be represented as a covering ILP with an exponential number of constraints. Unfortunately, these extremely strong hardness results pose further challenges:

> If the hardness of the offline version prevents any non-trivial approximation, is there any hope for interesting algorithmic results in the offline, online or stochastic version?

Our approach for answering this question is to consider a form of *resource augmentation*. More precisely, we relax the requirements on a solution Q for OfflineKnapExp: Instead of requiring that querying Q reveals sufficient information to identify an optimal solution for the underlying knapsack problem, we only require sufficient information to identify an α-approximate solution. Unfortunately, we can show that, unless P=NP, there is no non-trivial approximation for this relaxed problem variant if we compare against an optimal solution for the relaxed problem variant. However, as a second main result, we show that non-trivial algorithmic results are possible if the requirements on the algorithm's solution are less strict than the requirements on the optimal solution we compare against; we make this more precise in the next section.

1.1 Problem Definition

An *instance* $\mathcal{K}$ of KnapExp and OfflineKnapExp is a quintuple $\mathcal{K} = (\mathcal{I}, B, w, p, \mathcal{A})$, where $\mathcal{I} = \{1, \dots, n\}$ is a set of n items, $B \in \mathbb{N}$ is the knapsack capacity, w is the weight vector with $w_i \in \mathbb{N}_{\leq B}$ for all items $i \in \mathcal{I}$, p is the profit vector with $p_i \in \mathbb{R}_{\geq 0}$ for all $i \in \mathcal{I}$, and $\mathcal{A} = \{I_1, \dots, I_n\}$ is the set of uncertainty intervals such that $p_i \in I_i$ for all $i \in \mathcal{I}$. The quadruple $(\mathcal{I}, B, w, p)$ characterizes the underlying knapsack problem.

As is common in the area of explorable uncertainty, we assume that each uncertainty interval I_i is either open or *trivial*. That is, we either have $I_i = (L_i, U_i)$ for a *lower limit* L_i and an *upper limit* U_i, or $I_i = \{p_i\}$. In the latter case, we call both the item i and the uncertainty interval I_i *trivial* and define $U_i = L_i = p_i$. All items that are not trivial are called *non-trivial*. We use $\mathcal{I}_T$ to refer to the set of trivial items. A *query* of an item i reveals the profit p_i and can be seen as replacing the uncertainty interval $I_i = (L_i, U_i)$ with $I_i = \{p_i\}$.

In OfflineKnapExp, all input parameters are known to the algorithm, while in KnapExp the profits p_i are initially uncertain. Both problems ask for a *feasible query set* $Q \subseteq \mathcal{I}$ of minimum cardinality. Intuitively, a query set $Q \subseteq \mathcal{I}$ is feasible if the revealed information suffices to identify an optimal solution to the underlying knapsack problem *and* to determine the profit of such a solution. We proceed by making this definition more formal.

Packings and Feasible Query Sets. To formally define feasible query sets, we first define *packings*. A subset of items $P \subseteq \mathcal{I}$ is a *packing* if $\sum_{i \in P} w_i \leq B$. That is, packings are feasible solutions to the underlying knapsack problem. For $P \subseteq \mathcal{I}$ let $p(P) = \sum_{i \in P} p_i$ denote the profit of P. We call a packing *optimal* if it maximizes the profit over all packings. We usually use P^* to refer to an optimal packing and $p^* := p(P^*)$ to refer to the *optimal profit*.

For each packing $P \subseteq \mathcal{I}$ define $U_P := \sum_{i \in P} U_i$, i.e., the term U_P describes an upper limit on the maximum possible profit the packing could potentially have. Note that U_p can be computed even without access to the profits. By querying items in P, the upper limit U_P decreases as we gain more information and can replace the non-trivial uncertainty interval $I_i = (L_i, U_i)$ with $I_i = \{p_i\}$ after we query i and learn the profit p_i. For $Q \subseteq \mathcal{I}$, we use $U_i(Q)$ to denote the upper limit of i after querying Q, i.e., $U_i(Q) = p_i$ if $i \in Q$ and $U_i(Q) = U_i$ otherwise. The upper limit $U_P(Q)$ of packing P after querying a set $Q \subseteq \mathcal{I}$, is

$$U_P(Q) := \sum_{i \in P} U_i(Q) = \sum_{i \in P \setminus Q} U_i + \sum_{i \in P \cap Q} p_i = \sum_{i \in P} U_i - \sum_{i \in P \cap Q} (U_i - p_i) = U_P - \sum_{i \in P \cap Q} (U_i - p_i).$$

▶ **Definition 1.** *A query set $Q \subseteq \mathcal{I}$ is* feasible *if the following two conditions hold:*
1. *There is a packing $P \subseteq Q \cup \mathcal{I}_T$ with $p(P) = p^*$.*
2. *$U_P(Q) \leq p^*$ holds for every packing $P \subseteq \mathcal{I}$.*

The first condition of Definition 1 ensures that querying Q reveals sufficient information to verify that there exists a packing with the optimal profit p^*, while the second condition ensures that querying Q reveals sufficient information to verify that no packing can possibly have a larger profit than p^*, no matter what the profits of items $i \in \mathcal{I} \setminus Q$ actually are.

Since any packing P^* with $p(P^*) = p^*$ can only satisfy $U_{P^*}(Q) \leq p^*$ if $P^* \subseteq Q \cup \mathcal{I}_T$, the second condition of the definition actually implies the first one. In particular, this means that a query set is feasible if and only if it satisfies the constraints of the following ILP. Note that a similar covering point of view was first observed in [26] for the cheapest set problem.

$$\begin{array}{lll} \min & \sum_{i\in\mathcal{I}} x_i & \\ \text{s.t.} & \sum_{i\in P} x_i \cdot (U_i - p_i) \geq U_P - p^* & \forall P \subseteq \mathcal{I}\colon \sum_{i\in P} w_i \leq B \\ & x_i \in \{0,1\} & \forall i \in \mathcal{I} \end{array} \tag{K-ILP}$$

The offline problem. In the offline problem OfflineKnapExp, we are given an instance $\mathcal{K} = (\mathcal{I}, B, w, p, \mathcal{A})$ with full knowledge of all parameters and our goal is to compute a feasible queryset of minimum cardinality, which is equivalent to solving (K-ILP) with full knowledge of all coefficients. We use Q^* to refer to an optimal solution of OfflineKnapExp.

The online problem. In the online problem KnapExp, we are also given an instance $\mathcal{K} = (\mathcal{I}, B, w, p, \mathcal{A})$ but the profits p_i are initially unknown. The goal is to iteratively and adaptively query items i to reveal their profit p_i until the set of queried items is a feasibile query set. This problem can be seen as solving (K-ILP) with uncertain coefficients $U_i - p_i$ and right-hand side values $U_P - p^*$: Querying an item i corresponds to irrevocably setting $x_i = 1$, incurs a cost that cannot be reverted, and reveals the coefficient $(U_i - p_i)$.

Relaxations. As OfflineKnapExp turns out to admit strong inapproximability results, we introduce the following relaxed notion of feasible query sets. We use (α, β)-OfflineKnapExp to refer to the offline problem of computing a (α, β)-feasible query set of minimum cardinality.

▶ **Definition 2** ((α, β)-feasibility). *Let $\alpha, \beta \geq 1$. We say that a query set $Q \subseteq \mathcal{I}$ is (α, β)-feasible if the following two conditions hold:*
1. *There is a packing P such that $P \subseteq Q \cup \mathcal{I}_T$ and $p(P) \geq \frac{1}{\alpha} \cdot p^*$.*
2. *$U_P(Q) \leq \beta \cdot p^*$ for every packing P.*

The first condition of the definition ensures that we can find an α-approximation for the underlying knapsack instance by using only queried and trivial items. The second condition ensures that, after querying Q, no feasible packing can have profit greater than $\beta \cdot p^*$, no matter what the profits of the items in $\mathcal{I} \setminus Q$ are. Thus, querying Q reveals sufficient information to verify that the set P of the first condition is a $\frac{1}{\alpha\beta}$-approximation for the underlying knapsack instance. We use $Q^*_{\alpha,\beta}$ to refer to a minimum-cardinality (α, β)-feasible query set. Note that $Q^* = Q^*_{1,1}$, Q^* is (α, β)-feasible for every $\alpha, \beta \geq 1$ and $|Q^*| = \max_{\alpha\geq 1, \beta \geq 1} |Q^*_{\alpha,\beta}|$.

In contrast to Definition 1, the second condition of Definition 2 does not imply the first one. As a consequence, each (α, β)-feasible query set is a feasible solution for a variant of (K-ILP) in which we replace p^* with $\beta \cdot p^*$, but the inverse is not necessarily true.

1.2 Our Results and Outline

In Section 2, we give several hardness results for OfflineKnapExp. First, we show that deciding whether $Q = \emptyset$ is an (α, β)-feasible query set is weakly NP-hard for any $\alpha, \beta \geq 1$, which immediately implies that it is weakly NP-hard to approximate (α, β)-OfflineKnapExp within any bounded multiplicative factor. This hardness result mainly exploits that the optimal solution p^* to the weakly NP-hard [21] underlying knapsack problem appears on the right-hand sides of (K-ILP). Then, we move on to show that OfflineKnapExp is Σ_2^p-complete, which intuitively means that the problem remains hard even if we are given an oracle for deciding problems in NP. Since such an oracle allows us to compute p^*, the reason for the Σ_2^p-hardness is not the appearance of p^* in the right-hand sides but the exponential number of constraints in (K-ILP). In fact, we prove this hardness result via reduction from *succinct set cover* [31, 32], which is a set cover variant where the elements and sets are only given implicitly, i.e., the number of set cover constraints is exponential in the encoding size of the problem. Exploiting a result by [30], our reduction also shows that there is no $n_0^{1-\epsilon}$-approximation for any $\epsilon > 0$ unless $\Sigma_2^p = \Delta_2^p$, where $n_0 := |\mathcal{I} \setminus \mathcal{I}_T|$.

In Section 3, we design algorithms for (α, β)-OfflineKnapExp for different values of α and β. Since the hardness results prevent any non-trivial results when comparing against $|Q^*_{\alpha,\beta}|$, we analyze our algorithms by comparing their solutions to $|Q^*|$ instead. This can be seen as a form of resource augmentation as the requirements on the algorithms's solution are less strict than the requirements on the optimal solution we compare against. To achieve our algorithmic results, we treat the two conditions for (α, β)-feasible query sets (cf. Definition 2) as separate subproblems: (i) Compute a query set Q_1 such that there exists a packing $P \subseteq Q_1 \cup \mathcal{I}_T$ with $p(P) \geq \frac{1}{\alpha} p^*$, and (ii) Compute a query set Q_2 such that $U_P(Q_2) \leq \beta p^*$ holds for all packings P. First, we show how to solve subproblem (i) in polynomial-time for $\alpha = \frac{1}{1-\epsilon}$ with a set Q_1 such that $|Q_1| \leq |Q^*|$. Our algorithm for this subproblem exploits existing results for the two-dimensional knapsack problem. For (ii), we first show how to solve the problem in pseudopolynomial time for $\beta = 2 + \epsilon$ with the guarantee $|Q_2| \leq |Q^*|$. The algorithm is based on solving an OfflineKnapExp variant that only considers packings that correspond to certain greedy solutions. We justify the pseudopolynomial running-time by showing weak NP-hardness for this OfflineKnapExp variant. By considering a relaxed version of that problem, we manage to solve problem (ii) for $\beta = 4 + \epsilon$ with $|Q_2| \leq |Q^*|$ in polynomial time. Combining the results for both subproblems yields a pseudopolynomial algorithm that computes a $(\frac{1}{1-\epsilon}, 2+2\epsilon)$-feasible query set Q and a polynomial time algorithm that computes a $(\frac{1}{1-\epsilon}, 4 + 4\epsilon)$-feasible query set Q. In both cases, $|Q| \leq 2 \cdot |Q^*|$.

1.3 Further Related Work

Meißner [27] gives an adversarial lower bound of n for KnapExp that holds even if the instance only has two different weights, preventing any non-trivial adversarial results. However, Megow and Schlöter [26] show that this lower bound does not hold in a stochastic setting where the profits p_i are drawn from their intervals I_i according to an unknown distribution that satisfies $\Pr[p_i \leq \frac{U_i + L_i}{2}] \leq \tau$ for a threshold parameter τ. However, their result only breaks that particular lower bound instance and does not imply general stochastic results for KnapExp.

Goerigk et al. [18] consider a knapsack problem under uncertainty in a different query setting. In their problem, the profits are known and the weights are uncertain. Furthermore, there is a budget on the queries that an algorithm is allowed to make. These differences lead to a problem fundamentally different from KnapExp.

Maehara and Yamaguchi [23] consider packing ILPs with uncertainty in the cost coefficients. The cost coefficients can be queried. The key difference to the setting of explorable uncertainty is that they are interested in bounding the absolute number of queries instead of comparing against the optimal feasible query set. We remark that this is an important distinction between explorable uncertainty and many other query models. For example, the same distinction applies to a line of research that studies queries that reveal the *existence* of entities instead of numeric values, e.g., the existence of edges in a graph, c.f. [4–6, 9, 17, 33]. For instance, Behnezhad et al. [4] considered vertex cover in a stochastic setting and showed that it can be approximated within a factor of $(2+\epsilon)$ with only a constant number of queried edges per vertex.

2 Hardness of Approximation

We start by showing our hardness results for (α, β)-OfflineKnapExp. Not surprisingly, the appearance of p^* in the right-hand sides of (K-ILP) suffices to render (α, β)-OfflineKnapExp weakly NP-hard. The following proposition shows that even deciding whether a given set Q is (α, β)-feasible is already weakly NP-hard (cf. the full version [29] for a formal proof).

▶ **Proposition 3.** *Deciding if $Q = \emptyset$ is (α, β)-feasible is weakly NP-hard for any $\alpha, \beta \geq 1$.*

This means that distinguishing between instances that can be solved without any query and instances that need at least one query is weakly NP-hard, which implies the following:

▶ **Corollary 4.** *It is weakly NP-hard to approximate (α, β)-OfflineKnapExp within any bounded multiplicative factor.*

Proposition 3 is also an indicator that (α, β)-OfflineKnapExp might not be in NP, as verifying whether a query set is (α, β)-feasible is already NP-hard. For $\alpha, \beta = 1$, we make this observation more formal by proving that OfflineKnapExp is complete for the second level of the polynomial hierarchy, i.e., Σ_2^p-complete. Intuitively, the class Σ_2^p contains problems that, given an oracle for deciding problems in NP, can be solved in non-deterministic polynomial time. Similarly, the class Δ_2^p contains problems that, given the same type of oracle, can be solved in deterministic polynomial time. Hardness and completeness for the class Σ_2^p are defined in the same way as for the class NP. For a more formal introduction, we refer to [1]. Under the conjecture that $\sum_2^p \neq$ NP, the $\sum_2^p$-completeness implies that OfflineKnapExp is not in NP, and under the conjecture $\sum_2^p \neq \Delta_2^p$ it cannot be solved optimally in polynomial time even when given an oracle for deciding problems in NP.

▶ **Theorem 5.** OfflineKnapExp *is Σ_2^p-complete.*

Proof. We show Σ_2^p-membership in the full version [29] and focus here on proving Σ_2^p-hardness. Our proof is via reduction from *succinct set cover*, which is known to be $\sum_2^p$-complete [31, 32]. In the same way as for NP-hardness proofs, we need to give a polynomial time reduction to the decision problem variant of OfflineKnapExp such that the constructed instance is a YES-instance if and only if the given succinct set cover instance is a YES-instance.

Succinct set cover. We are given n decision variables $x_1, \ldots, x_n$, m propositional logic formulas $\phi_1, \ldots, \phi_m$ over these variables and an integer parameter k. Each formula ϕ_j is in 3-DNF form[1] and we use S_j to denote the set of 0-1-vectors (variable assignments) that

[1] *Disjunctive normal form (DNF)* refers to a disjunction of conjunctions, i.e., $\phi_j = C_{j,1} \vee \ldots \vee C_{j,k_j}$, where each clause $C_{j,k}$ is a conjunction of literals. In 3-DNF, each $C_{j,k}$ contains exactly three literals. A formula in DNF is satisfied by a variable assignment if at least one clause is satisfied by the assignment.

satisfy ϕ_j. The formulas ϕ_j have to satisfy $\bigcup_{j\in\{1,\dots,m\}} S_j = \{0,1\}^n$, i.e., each variable assignment satisfies at least one formula ϕ_j. The goal is to find a subset $S \subseteq \{1,\dots,m\}$ such that $\bigcup_{j\in S} S_j = \{0,1\}^n$ and $|S| \leq k$. We assume that each variable occurs as a literal in at least one formula. If not, we can just remove the variable and obtain a smaller instance. Succinct set cover can be interpreted as a set cover variant where the elements and sets are only given *implicitly* and not as an explicit part of the input.

Main reduction idea. Before we give the technical details of the reduction, we first sketch the basic idea. In particular, we describe the properties that we want the constructed instance to have. In the technical part, we then describe how to actually achieve these properties. At its core, the reduction will use the knapsack weights to encode structural information of the input instance into the constructed instance. The idea to use numerical weights to encode constraints is quite natural in hardness proofs for weakly NP-hard problems, see e.g. the NP-hardness proof for the subset sum problem given in [22]. The usage of the knapsack weights in our reduction is on some level inspired by such classical reductions, but requires several new ideas to handle the implicit representation of the input problem and the covering nature of OfflineKnapExp.

First, we introduce a single trivial item i^* with $w_{i^*} = p_{i^*} = B$. This item alone fills up the complete knapsack capacity B, which we define later, and the instance will be constructed in such a way that $p^* = p_{i^*} = B$ is the maximum profit of any packing. Thus, only packings P with $U_P > p^*$ induce non-trivial constraints in (K-ILP). We design the instance such that $U_P \geq p^*$ only holds for packings that use the full capacity.

▶ **Property 1.** A packing P of the constructed instance satisfies $U_P \geq p^*$ only if $w(P) = B$.

Next, we want each packing P with $w(P) = B$ and $U_P > p^*$ to represent a distinct variable assignment in $\{0,1\}^n$. To this end, we introduce a set X of $2n$ items, two items v_i and $\bar{v}_i$ for each variable x_i with $i \in \{1,\dots,n\}$. Intuitively, v_i represents the positive literal x_i and $\bar{v}_i$ represents the negative literal $\neg x_i$. We say that a subset $X' \subseteq X$ *represents* a variable assignment if $|X' \cap \{v_i, \bar{v}_i\}| = 1$ for all $i \in \{1,\dots,n\}$. We design our instance such that the packings P with $w(P) = B$ and $U_P > p^*$ exactly correspond to the variable assignments in $\{0,1\}^n$. Note that this excludes the packing $P = \{i^*\}$ as this packing has $w(P) = B$ and $p(P) = p^*$.

▶ **Property 2.** If $w(P) = B$ and $U_P > p^*$, then $P \cap X$ represents a variable assignment. Each variable assignment is represented by at least one P with $w(P) = B$ and $U_P > p^*$.

If the first two properties hold, then all non-trivial constraints in the ILP (K-ILP) for the constructed instance correspond to a packing P with $w(P) = B$ and $U_P > p^*$ and, thus, to a variable assignment of the given succinct set cover instance. Furthermore, each variable assignment is represented by at least one active constraint. With the next property, we want to ensure that each possible query, i.e., each non-trivial item, corresponds to a succinct set cover formula ϕ_j. To this end, we introduce the set of items $Y = \{y_1,\dots,y_m\}$. These items will be the only non-trivial items in the constructed instance, so each possible query is to an element of Y. Next, we want to achieve that querying an item y_j suffices to satisfy all constraints of (K-ILP) for packings P with $w(P) = B$ and $U_P > p^*$ that represent a variable assignment which satisfies formula ϕ_j, and does not impact any other constraints. Formally, we would like to design our instance such that the following property holds.

▶ **Property 3.** For each packing P with $U_P > p^*$ and each $y_j \in Y$: $y_j \in P$ if and only if $X \cap P$ represents a variable assignment that satisfies ϕ_j. If $y_j \in P$, then $U_P - p^* \leq U_{y_j} - p_{y_j}$.

If we manage to define our reduction in such a way that the three properties are satisfied, it is not hard to show correctness (see the full version [29] for the second direction):

First Direction. If there is an index set S with $|S| \leq k$ that is a feasible solution to the succinct set cover instance, then each possible variable assignment must satisfy at least one formula ϕ_j with $j \in S$. We claim that $Q = \{y_j \mid j \in S\}$ is a feasible query set for the constructed OfflineKnapExp instance. To this end, consider an arbitrary packing P with $U_P > p^*$, which are the only packings that induce non-trivial constraints in the corresponding (K-ILP). By Property 1, we have $w(P) = B$. Property 2 implies that $X \cap P$ represents some variable assignment φ and Property 3 implies that $y_j \in P$ for all ϕ_j that are satisfied by φ. By assumption that S is a feasible succinct set cover solution, we get $Q \cap P \neq \emptyset$. Property 3 implies $U_P(Q) \leq U_P(\{y_j\}) \leq p^*$ for each $y_j \in Q \cap P$. Thus, Q satisfies the constraint of P in the (K-ILP) for the constructed instance. As this holds for all P with $U_P > p^*$, the set Q is a feasible solution for the constructed OfflineKnapExp instance.

Technical reduction. It remains to show how to actually construct an OfflineKnapExp instance that satisfies the three properties. Given an instance of succinct set cover, we construct an instance of OfflineKnapExp consisting of four sets X, Φ, A and L of items such that $\mathcal{I} = X \cup \Phi \cup A \cup L$. The set X is defined exactly as sketched above, the set Φ contains the set $Y = \{y_j, \dots, y_m\}$ as introduced above, and $L := \{i^*\}$ for the item i^* with $w_{i^*} = p_{i^*} = B$. All further items will be used to ensure the three properties.

Conceptionally, we construct *several* partial weights for each item i that will later be combined into a single weight. For each item i, we construct two weights w_{i,ϕ_j} and w_{i,ρ_j} for each formula ϕ_j, and a single weight $w_{i,x}$. Similarly, we break down the knapsack capacity into multiple partial knapsack capacities B_x, and B_{ϕ_j}, B_{ρ_j} for each ϕ_j. Intuitively, the full weight w_i of an item i will be the concatenation of the decimal representations of the partial weights, i.e., $w_i = w_{i,x} w_{i,\rho_m} \cdots w_{i,\rho_1} w_{i,\phi_m} \cdots w_{i,\phi_1}$, and B will be the concatenation of the partial capacities. We make this more precise in the full version [29] after defining all partial weights in such a way that the following property holds.

▶ **Property 4.** For each packing P with $P \neq \{i^*\}$, it holds $\sum_{i \in P} w_i = B$ if and only if $\sum_{i \in P} w_{i,x} = B_x$, and $\sum_{i \in P} w_{i,\phi_j} = B_{\phi_j}$ and $\sum_{i \in P} w_{i,\rho_j} = B_{\rho_j}$ for all $j \in \{1, \dots, m\}$.

For now, we operate under the assumption that Property 4 holds and focus on the partial weights and capacities, and proceed by defining the remaining parts of the construction.

Definition of the w_x-weights. As formulated in Property 2, we would like each packing P with $w(P) = B$ and $U_P > p^*$ to represent a variable assignment. To this end, we need such a packing to contain exactly one item of $\{v_i, \bar{v}_i\}$ for each $i \in \{1, \dots, n\}$. To enforce this, we use the partial w_x-weights and the partial capacity B_x. In particular, we define $w_{v_i,x} = w_{\bar{v}_i,x} = 10^i$ for each $i \in \{1, \dots, n\}$, and $B_x = \sum_{i=1}^{n} 10^i$. For all items $j \in \mathcal{I} \setminus X$, we define $w_{j,x} = 0$, which immediately implies the following property:

▶ **Property 5.** P satisfies $\sum_{i \in P} w_{i,x} = B_x$ iff $P \cap X$ represents a variable assignment.

Definition of the set A and the w_{ϕ_j}-weights. Define $A = \bigcup_{j \in \{1,\dots,m\}} A_{\phi_j}$ for sets A_{ϕ_j} to be defined below. For formula ϕ_j, let k_j denote the number of clauses in ϕ_j and let $C_{j,1}, \dots, C_{j,k_j}$ denote these clauses. For each $C_{j,k}$, we add four items $a_{j,k,0}, a_{j,k,1}, a_{j,k,2}, a_{j,k,3}$ to set A_{ϕ_j}. The idea is to define the partial w_{ϕ_j}-weights and the partial B_{ϕ_j} capacity in such a way that the following property holds.

▶ **Property 6.** For each ϕ_j, a packing P satisfies $\sum_{i\in P} w_{i,\phi_j} = B_{\phi_j}$ and $\sum_{i\in P} w_{i,x} = B_x$ iff $a_{j,k,0} \in P$ for each clause $C_{j,k}$ that is satisfied by the assignment represented by $X \cap P$.

To achieve the property, we first define the partial w_{ϕ_j}-weights for the items X. For a literal x_i, let $\mathcal{C}_{x_i,j} := \{k \mid x_i \text{ occurs in } C_{j,k}\}$ and define $\mathcal{C}_{\neg x_i,j}$ in the same way. We define the weights w_{v_i,ϕ_j} and $w_{\bar{v}_i,\phi_j}$ as $\sum_{k\in\mathcal{C}_{x_i,j}} 10^{k_j+k-1}$ and $\sum_{k\in\mathcal{C}_{\neg x_i,j}} 10^{k_j+k-1}$, respectively. Intuitively, digit $k_j + k$ of the sum $\sum_{h\in X\cap P} w_{h,\phi_j}$ of a packing P with $\sum_{i\in P} w_{i,x} = B_x$ indicates whether the assignment represented by $X \cap P$ satisfies clause $C_{j,k}$ or not: If the clause is satisfied, then $X \cap P$ contains the items that represent the three literals of $C_{j,k}$ and the digit has value 3. Otherwise, $X \cap P$ contains at most two of the items that represent the literals of $C_{j,k}$ and the digit has value at most 2.

Finally, for each for $i \in \{0, 1, 2, 3\}$, we define the w_{ϕ_j}-weight of item $a_{j,k,i}$ as $w_{\phi_j,a_{j,k,i}} = i\cdot 10^{k_j+k-1} + 10^{k-1}$. For all remaining items $i \in \mathcal{I}\setminus(X \cup A_{\phi_j})$, we define the partial w_{ϕ_j}-weight to be zero. Furthermore, we define the partial capacity $B_{\phi_j} = \sum_{k=0}^{k_j-1} 10^k + \sum_{k=k_j}^{2k_j-1} 3 \cdot 10^k$. We claim that these definitions enforce Property 6.

Intuitively, the fact that the k_j decimal digits of lowest magnitude in B_{ϕ_j} have value 1 forces a packing P with $\sum_{i\in P} w_{i,\phi_j} = B_{\phi_j}$ to contain exactly one item of $\{a_{j,k,0}, \ldots, a_{j,k,3}\}$ for each $k \in \{1, \ldots, k_j\}$ as each such item increases the corresponding digit $k - 1$ by one. Similarly, the value of each digit $k_j + k - 1$ in B_{ϕ_j} is three. Since the elements of X that occur in clause $C_{j,k}$ increase the value of digit $k_j + k - 1$ in $w_{\phi_j}(P)$ by one and item $a_{j,k,i}$ increases the digit by i, a packing P with value three in digit $k_j + k - 1$ of $w_{\phi_j}(P)$ can contain item $a_{j,k,0}$ if and only if $X \cap P$ contains the three items that correspond to the literals in $C_{j,k}$. This implies Property 6. We give a more formal argumentation in the full version [29].

Definition of set Φ and the w_{ρ_j}-weight. Define $\Phi = \bigcup_{j\in\{1,\ldots,m\}} \Phi_j$ such that $\Phi_j = \{y_j, u_j, f_{j,0}, \ldots, f_{j,k_j-1}\}$. Note that y_j is the item that has already been introduced for Property 3. As a step toward enforcing this property, we define the w_{ρ_j}-weights such that:

▶ **Property 7.** For each ϕ_j, a packing P with $\sum_{i\in P} w_{i,\rho_j} = B_{\rho_j}$ has $y_j \in P$ if and only if $a_{j,k,0} \in P$ for some clause $C_{j,k}$ in ϕ_j.

To enforce this property, we define the following partial capacity $B_{\rho_j} = k_j + 10^{k_j^2} + 10^{k_j^2+1}$. Next, we define the w_{ρ_j}-weight of the elements $a_{j,k,0}$, $k \in \{1, \ldots, k_j\}$, as $w_{a_{j,k,0},\rho_j} = 1$. Furthermore, we define $w_{u_j,\rho_j} = 10^{k_j^2+1} + 10^{k_j^2} + k_j$, $w_{y_j,\rho_j} = 10^{k_j^2+1}$ and $w_{f_{j,k},\rho_j} = 10^{k_j^2} + k$, for all $k \in \{0, \ldots, k_j - 1\}$. For all other items, define the w_{ρ_j}-weight to be zero. We show in the full version [29] that these definitions imply Property 7.

Definition of the uncertainty intervals and precise profits. To finish the reduction, we define the profits and uncertainty intervals of all introduced items:

- For the items y_j, $j \in \{1, \ldots, m\}$, we define the uncertainty interval $I_{y_j} = (w_{y_j} - 2, w_{y_j} + \epsilon)$ for a fixed $0 < \epsilon < \frac{1}{m}$. We define the profits as $p_{y_j} = w_{y_j} - 1$.
- For all items $i \in \mathcal{I} \setminus \{y_j \mid j \in \{1, \ldots, m\}\}$, we use trivial uncertainty intervals $I_i = \{w_i\}$.

Proof of the three main properties. With the full construction in place, we are ready to prove the three main properties from the beginning of the proof:

1. **Property 1:** By definition of the profits, we have $p(P) \leq w(P)$ for each packing P, which implies that the maximum profit is $p^* \leq B$. Since the packing $P = L = \{i^*\}$ has a profit of exactly B, we get $p^* = B$. On the other hand, the upper limit U_P of a packing P is $w(P) + \epsilon|P \cap \{y_j \mid j \in \{1, \ldots, m\}\}|$ as only the items in $\{y_j \mid j \in \{1, \ldots, m\}\}$ have

a non-trivial uncertainty interval with upper limits of $w_{y_j} + \epsilon$. By choice of ϵ, this gives $U_P = w(P) + \epsilon|P \cap \{y_j \mid j \in \{1, \dots, m\}\}| < w(P) + 1$. Since all weights are integer, this implies that $U_P \geq p^* = B$ only holds if $w(P) = B$.

2. **Property 2:** The property, in particular the first part, is essentially implied by Property 4 and Property 5. We give the formal argumentation in the full version [29].
3. **Property 3:** By Property 1 and definition of the uncertainty intervals, a packing P has $U_P > p^*$ if and only if $w(P) = B$ and $P \neq \{i^*\}$. By Property 4, the latter holds if and only if $w_x(P) = B_x$, $w_{\phi_j}(P) = B_{\phi_j}$ and $w_{\rho_j}(P) = B_{\rho_j}$ for all $j \in \{1, \dots, m\}$. Fix a packing P with $y_j \in P$ for some $j \in \{1, \dots, m\}$. By Property 7, $y_j \in P$ holds if and only if $a_{j,k,0} \in P$ for some clause $C_{j,k}$ in ϕ_j. By Property 6, $a_{j,k,0} \in P$ if and only if $C_{j,k}$ is satisfied by the assignment represented by $X \cap P$. This gives the first part of Property 3. For the final part, observe that $U_{y_j} - p_{y_j} > 1$. On the other hand, $U_p - p^* < 1$. Hence, the second part of Property 3 holds.

To finish the proof of the reduction, it remains to argue about the running time and space complexity of the reduction. We do so in the full version [29]. The main argument is that, while the numerical values of the constructed weights are exponential, the number of digits in their decimal representations (and, thus, their encoding size) is polynomial. ◀

The previous theorem proves $\sum_2^p$-hardness for OfflineKnapExp. Exploiting the inapproximability result on the succinct set cover problem given in [30, Theorem 7.2], we can show the following stronger statement by using the same reduction (cf. the full version [29] for a formal proof).

▶ **Theorem 6.** *Unless $\Sigma_2^p = \Delta_2^p$, there exists no $n_0^{1-\epsilon}$-approximation (given access to an oracle for problems in NP) for* OfflineKnapExp *for any $\epsilon > 0$, where $n_0 := |\mathcal{I} \setminus \mathcal{I}_T|$.*

▶ Remark 7. We remark that all results given in this section require large numerical input parameters. Hence, they do not prohibit the existence of pseudopolynomial algorithms.

3 Algorithmic Results

In this section, we give algorithms for (α, β)-OfflineKnapExp for different values of α and β. Motivated by the hardness results of the previous section, we show bounds on the size of the computed query sets in comparison to $|Q^*|$ instead of $|Q^*_{\alpha,\beta}|$. All our algorithms treat the two conditions on (α, β)-feasible query sets (cf. Definition 2) as two separate subproblems:

1. Compute a query set Q_1 such that there exists a packing $P \subseteq Q_1 \cup \mathcal{I}_T$ with $p(P) \geq \frac{1}{\alpha} p^*$.
2. Compute a query set Q_2 such that $U_P(Q_2) \leq \beta \cdot p^*$ for all packings P.

In the following, we show how to solve these two problems and give bounds on $|Q_1 \cup Q_2|$ in terms of $|Q^*|$.

3.1 The First Subproblem

The following lemma solves the first subproblem for $\alpha = \frac{1}{1-\epsilon}$ by computing a packing P with $p(P) \geq (1 - \epsilon) \cdot p^*$ and $|P \setminus \mathcal{I}_T| \leq |Q^*|$. The set $Q_1 = P \setminus \mathcal{I}_T$ satisfies the requirement of the subproblem and has $|Q_1| \leq |Q^*|$. In the full version [29], we prove the lemma by exploiting existing algorithms for two-dimensional knapsack.

▶ **Lemma 8.** *Fix an $\epsilon > 0$. Given an instance of* OfflineKnapExp*, there exists a polynomial time algorithm that computes a packing P with $p(P) \geq (1 - \epsilon) \cdot p^*$ and $|P \setminus \mathcal{I}_T| \leq |Q^*|$.*

3.2 The Second Subprobem: Prefix Problems

For the second subproblem, we consider special packings that correspond to greedy solutions for the underlying knapsack problem. For an item $i \in \mathcal{I}$, define the *density* $d_i = \frac{p_i}{w_i}$ and the *optimistic density* $\bar{d}_i = \frac{U_i}{w_i}$. For a query set $Q \subseteq \mathcal{I}$, define the optimistic density of an item i *after* querying Q as $\bar{d}_i(Q) = \bar{d}_i$ if $i \notin Q$ and $\bar{d}_i(Q) = d_i$ if $i \in Q$. We use $\bar{\prec}_Q$ to denote the *optimistic density order* after querying Q, that is, for $i, j \in \mathcal{I}$ we use $i \bar{\prec}_Q j$ to denote that $\bar{d}_i(Q) \geq \bar{d}_j(Q)$. We assume an arbitrary but fixed tiebreaking rule between the items to treat $\bar{\prec}_Q$ as a total order. This allows us to define *optimistic prefixes* and the *prefix problem*.

▶ **Definition 9** (Optimistic prefixes). *For a set $Q \subseteq \mathcal{I}$ and a parameter $0 \leq C \leq B$, define the* optimistic prefix $\bar{F}_C(Q)$ *to be the maximal prefix S of order $\bar{\prec}_Q$ such that $w(S) \leq C$. To shorten the notation, we define $\bar{F}(Q) := \bar{F}_B(Q)$*

From the analysis of the knapsack greedy algorithm, it is well-known that the following holds for all packings P:

$$U_P(Q) \leq U_{\bar{F}(Q)}(Q) + \max_{i \in \mathcal{I}} U_i(Q). \tag{1}$$

If we compute a set Q, such that $U_{\bar{F}(Q)} \leq \beta' p^*$ and $\max_{i \in \mathcal{I}} U_i(Q) \leq \beta' \cdot p^*$, then Q solves the second subproblem for $\beta = 2 \cdot \beta'$, which motivates the following problem.

▶ **Definition 10** (Prefix problem). *Given an instance of* OfflineKnapExp *and a threshold parameter $D \geq p^*$, where p^* is the optimal profit of the underlying knapsack instance, the* prefix problem *asks for the set $Q \subseteq \mathcal{I}$ of minimum cardinality such that $U_{\bar{F}(Q)}(Q) \leq D$.*

Unfortunately, the prefix problem preserves the hardness of knapsack, even if the given threshold is larger than p^* by a constant factor. We show this in the full version [29].

▶ **Theorem 11.** *The prefix problem is weakly NP-hard for every $D = c \cdot p^*$ with $c \geq 1$.*

On the positive side, the problem can be solved to optimality in pseudopolynomial time.

▶ **Theorem 12.** *The prefix problem can be solved in pseudopolynomial time.*

We give the full proof of Theorem 12 in the full version [29], but highlight the main ideas here. In the following, we use Q^*_F to refer to an optimal solution for the prefix problem.

Assume for now, that the algorithm knows the last item i_1 in the prefix $\bar{F}(Q^*_F)$ and the first item i_2 outside of $\bar{F}(Q^*_F)$ in the order $\bar{\prec}_{Q^*_F}$ (Figure 1 (a)) and, for the sake of simplicity, assume that $i_1, i_2 \notin Q^*_F$. We design our algorithm to reconstruct the optimal solution $\bar{F}(Q^*_F)$ (or a similar solution) using just the knowledge of i_1 and i_2.

If we look at the same two items i_1, i_2 in the initial optimistic density order $\bar{\prec}_\emptyset$, then there can be a subset of items S before i_1, a subset of items A between i_1 and i_2 and a subset of items R after i_2 (Figure 1 (b)). Based on i_1 and i_2 being next to each other in order $\bar{\prec}_{Q^*_F}$, we can immediately deduce that $A \subseteq Q^*_F$ as items in $A \setminus Q^*_F$ would still be between i_1 and i_2 in $\bar{\prec}_{Q^*_F}$. Thus, the algorithm can safely add A to its solution as the optimal solution does the same. Similarly, from i_2 being the first item outside of $\bar{F}(Q^*_F)$, it is clear that $R \cap Q^*_F = \emptyset$ as the items in R stay outside the prefix $\bar{F}(Q^*_F)$ whether they are queried or not. Hence, the algorithm can safely ignore such items.

In the order $\bar{\prec}_A$, i.e., in the order after adding A to the solution, the items i_1 and i_2 must already be next to each other. However, we still can have $i_1 \notin \bar{F}(A)$, that is, i_1 might not yet be part of the prefix (Figure 1 (c)). To fix this, the algorithms needs to query items from the

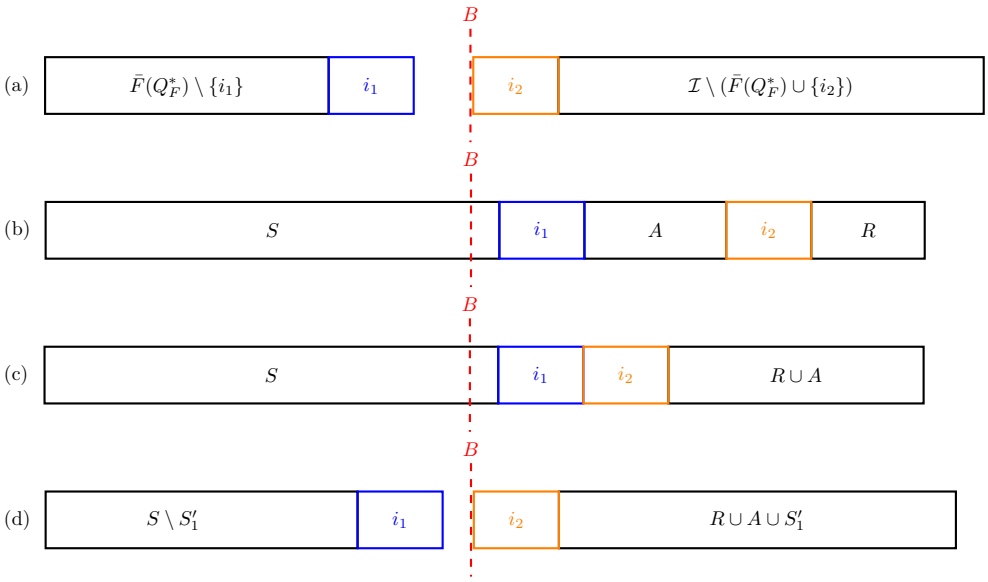


Figure 1 Illustration of the algorithmic ideas used to prove Theorem 12.

set $S_1 \subseteq S$, which contains items that are before i_1 in the order $\bar{\prec}_A$ but would move behind i_2 if they are added to the solution. To reconstruct the optimal solution, the algorithm has to query these items in such a way that i_1 enters the prefix but i_2 does not. On the one hand, the algorithm should select items $i \in S_1$ with high U_i as the items will leave the prefix and the goal of the prefix problem is to decrease the upper limit of the prefix below the threshold D. On the other hand, the algorithm needs to make sure not to query too many items in S_1, so the cardinality of the solution does not grow too large. If K is the minimum amount of weight that needs to be queried for i_1 to enter the prefix, D is the maximum amount of weight that can be queried before i_2 enters the prefix, and n_1 is the number of queries that the algorithm can afford, then the algorithm should select its queries from S_1 according to the following ILP, which we show to be solvable in pseudopolynomial time:

$$\begin{aligned} \max \quad & \textstyle\sum_{i \in S_1} x_i \cdot U_i \\ \text{s.t.} \quad & \textstyle\sum_{i \in S_1} x_i \cdot w_i \geq K \\ & \textstyle\sum_{i \in S_1} x_i \cdot w_i \leq H \\ & \textstyle\sum_{i \in S_1} x_i = n_1 \\ & x_i \in \{0,1\} \qquad \forall i \in S_1 \end{aligned} \tag{$\mathcal{P}_{S_1}$}$$

After adding the solution S_1' of $(\mathcal{P}_{S_1})$ to the solution, the prefix $\bar{F}(A \cup S_1')$ of the algorithm has reached roughly the same configuration as $\bar{F}(Q_F^*)$: i_1 is last in the prefix and i_2 is first outside the prefix (Figure 1 (d)). However, we still might have $U_{\bar{F}(A \cup S_1')}(A \cup S_1') > D$. To fix this, the algorithm has to query items of $S \setminus S_1'$ that stay in front of i_1 in the optimistic density order even after being queried. Since these items are part of the prefix $\bar{F}(A \cup S_1')$ *and* will stay part of the prefix even after being queried, the algorithm should greedily query such items i with large $U_i - p_i = U_{\bar{F}(A \cup S_1')}(A \cup S_1') - U_{\bar{F}(A \cup S_1' \cup \{i\})}(A \cup S_1' \cup \{i\})$ until the solution becomes feasible for the prefix problem instance. Our algorithm for Theorem 12 is based on exactly this approach. We show in the full version [29] how to formalize this and get rid of all assumptions that were used in the description above.

To achieve polynomial running time, we use the same approach but instead of $(\mathcal{P}_{S_1})$ we solve a certain LP-relaxation with the property that an optimal basic feasible solution has at most two fractional variables. Omitting the fractional items leads to the following corollary.

▶ **Corollary 13.** *Given an instance of the prefix problem with threshold parameter D, let Q_F^* denote an optimal solution to the instance. There exists a polynomial time algorithm that computes a set Q with $|Q| \leq |Q_F^*|$ such that $U_{\bar{F}(Q)}(Q) \leq D + 2 \cdot \max_{i \in \mathcal{I}} U_i$.*

3.3 Combining the Subproblems

By combining Lemma 8 and Theorem 12, we can show the following theorem. The idea is to use Lemma 8 to compute a packing P with $p(P) \geq (1-\epsilon')p^*$ for a carefully chosen $\epsilon' > 0$ and then use Theorem 12 with threshold $D = \frac{p(P)}{(1-\epsilon')}$ to compute a solution Q' to the prefix problem. Exploiting (1), we return the solution $Q = (P \setminus \mathcal{I}_T) \cup Q' \cup \{i \in \mathcal{I} \mid U_i > D\}$ and observe that Q satisfies the theorem. A formal proof is given in the full version [29].

▶ **Theorem 14.** *Fix $\epsilon > 0$. There exists a pseudopolynomial algorithm that given an instance of* OfflineKnapExp *computes a $(\frac{1}{1-\epsilon}, (1+\epsilon) \cdot 2)$-feasible query set Q with $|Q| \leq 2|Q^*|$.*

Replacing the usage of Theorem 12 with Corollary 13 in the approach above yields:

▶ **Theorem 15.** *Fix $\epsilon > 0$. There exists a polynomial time algorithm that given an instance of* OfflineKnapExp *computes a $(\frac{1}{1-\epsilon}, (1+\epsilon) \cdot 4)$-feasible query set Q with $|Q| \leq 2|Q^*|$.*

4 Conclusion

We hope that our results on OfflineKnapExp improve the understanding of NP-hard problems under explorable uncertainty. In particular, our algorithmic insights on the resource augmentation setting give hope for tackling such problems even if the corresponding offline versions have strong impossibility results. For knapsack specifically, studying a stochastic version of the prefix problem of Section 3, for example in the stochastic setting of [26], seems like a logical next step towards algorithmic results for the non-offline KnapExp. For OfflineKnapExp, our results of Section 3 show that non-trivial algorithmic results with theoretical guarantees are possible, opening the door for more research on finding the best-possible guarantees.

References

1. Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.
2. Evripidis Bampis, Konstantinos Dogeas, Thomas Erlebach, Nicole Megow, Jens Schlöter, and Amitabh Trehan. Competitive query minimization for stable matching with one-sided uncertainty. In *APPROX/RANDOM*, volume 317 of *LIPIcs*, pages 17:1–17:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.APPROX/RANDOM.2024.17.
3. Evripidis Bampis, Christoph Dürr, Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlöter. Orienting (hyper)graphs under explorable stochastic uncertainty. In *ESA*, volume 204 of *LIPIcs*, pages 10:1–10:18, 2021. doi:10.4230/LIPIcs.ESA.2021.10.
4. Soheil Behnezhad, Avrim Blum, and Mahsa Derakhshan. Stochastic vertex cover with few queries. In *SODA*, pages 1808–1846. SIAM, 2022. doi:10.1137/1.9781611977073.73.
5. Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Stochastic matching with few queries: (1-ϵ) approximation. In *STOC*, pages 1111–1124. ACM, 2020. doi:10.1145/3357713.3384340.
6. Avrim Blum, John P. Dickerson, Nika Haghtalab, Ariel D. Procaccia, Tuomas Sandholm, and Ankit Sharma. Ignorance is almost bliss: Near-optimal stochastic matching with few queries. *Oper. Res.*, 68(1):16–34, 2020. doi:10.1287/opre.2019.1856.

7 R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient update strategies for geometric computing with uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005. doi:10.1007/s00224-004-1180-4.

8 George Charalambous and Michael Hoffmann. Verification problem of maximal points under uncertainty. In *IWOCA 2013*, volume 8288 of *Lecture Notes in Computer Science*, pages 94–105. Springer, 2013. doi:10.1007/978-3-642-45278-9_9.

9 Shaddin Dughmi, Yusuf Hakan Kalayci, and Neel Patel. On sparsification of stochastic packing problems. In *ICALP*, volume 261 of *LIPIcs*, pages 51:1–51:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.51.

10 T. Erlebach and M. Hoffmann. Minimum spanning tree verification under uncertainty. In D. Kratsch and I. Todinca, editors, *WG 2014: International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 8747 of *Lecture Notes in Computer Science*, pages 164–175. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-319-12340-0_14.

11 T. Erlebach, M. Hoffmann, and F. Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Theoretical Computer Science*, 613:51–64, 2016. doi:10.1016/j.tcs.2015.11.025.

12 T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihal'ák, and R. Raman. Computing minimum spanning trees with uncertainty. In *STACS'08: 25th International Symposium on Theoretical Aspects of Computer Science*, pages 277–288, 2008. doi:10.48550/arXiv.0802.2855.

13 Thomas Erlebach, Murilo S. de Lima, Nicole Megow, and Jens Schlöter. Sorting and hypergraph orientation under uncertainty with predictions. In *IJCAI*, pages 5577–5585. ijcai.org, 2023. doi:10.24963/ijcai.2023/619.

14 Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlöter. Learning-augmented query policies for minimum spanning tree with uncertainty. In *ESA*, volume 244 of *LIPIcs*, pages 49:1–49:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.49.

15 T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62(1):1–18, 2007. doi:10.1016/j.jalgor.2004.07.005.

16 T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32(2):538–547, 2003. doi:10.1137/S0097539701395668.

17 Michel X. Goemans and Jan Vondrák. Covering minimum spanning trees of random subgraphs. *Random Struct. Algorithms*, 29(3):257–276, 2006. doi:10.1002/rsa.20115.

18 Marc Goerigk, Manoj Gupta, Jonas Ide, Anita Schöbel, and Sandeep Sen. The robust knapsack problem with queries. *Comput. Oper. Res.*, 55:12–22, 2015. doi:10.1016/j.cor.2014.09.010.

19 M. M. Halldórsson and M. S. de Lima. Query-competitive sorting with uncertainty. In *MFCS*, volume 138 of *LIPIcs*, pages 7:1–7:15, 2019. doi:10.4230/LIPIcs.MFCS.2019.7.

20 S. Kahan. A model for data in motion. In *STOC'91: 23rd Annual ACM Symposium on Theory of Computing*, pages 265–277, 1991. doi:10.1145/103418.103449.

21 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

22 Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., USA, 2005.

23 Takanori Maehara and Yutaro Yamaguchi. Stochastic packing integer programs with few queries. *Mathematical Programming*, 182(1):141–174, 2020. doi:10.1007/s10107-019-01388-x.

24 Corinna Mathwieser and Eranda Çela. Special cases of the minimum spanning tree problem under explorable edge and vertex uncertainty. *Networks*, 83(3):587–604, 2024. doi:10.1002/net.22204.

25 N. Megow, J. Meißner, and M. Skutella. Randomization helps computing a minimum spanning tree under uncertainty. *SIAM Journal on Computing*, 46(4):1217–1240, 2017. doi:10.1137/16M1088375.

26 Nicole Megow and Jens Schlöter. Set selection under explorable stochastic uncertainty via covering techniques. In *IPCO*, volume 13904 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2023. doi:10.1007/978-3-031-32726-1_23.

27 J. Meißner. *Uncertainty Exploration: Algorithms, Competitive Analysis, and Computational Experiments*. PhD thesis, Technischen Universität Berlin, 2018. doi:10.14279/depositonce-7327.

28 Arturo Merino and José A. Soto. The minimum cost query problem on matroids with uncertainty areas. In *ICALP*, volume 132 of *LIPIcs*, pages 83:1–83:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.83.

29 Jens Schlöter. On the complexity of knapsack under explorable uncertainty: Hardness and algorithms, 2025. doi:10.48550/arXiv.2507.02657.

30 Amnon Ta-Shma, Christopher Umans, and David Zuckerman. Loss-less condensers, unbalanced expanders, and extractors. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 143–152, 2001. doi:10.1145/380752.380790.

31 Christopher Umans. Hardness of approximating sigma$_2^{\mathrm{p}}$ minimization problems. In *FOCS*, pages 465–474. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814619.

32 Christopher Umans. The minimum equivalent dnf problem and shortest implicants. *Journal of Computer and System Sciences*, 63(4):597–611, 2001. doi:10.1006/jcss.2001.1775.

33 Jan Vondrák. Shortest-path metric approximation for random subgraphs. *Random Struct. Algorithms*, 30(1-2):95–104, 2007. doi:10.1002/rsa.20150.

Graph Modification of Bounded Size to Minor-Closed Classes as Fast as Vertex Deletion

Laure Morelle
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Ignasi Sau
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Dimitrios M. Thilikos
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Abstract

A *replacement action* is a function $\mathcal{L}$ that maps each graph H to a collection of graphs of size at most $|V(H)|$. Given a graph class $\mathcal{H}$, we consider a general family of graph modification problems, called $\mathcal{L}$-Replacement to $\mathcal{H}$, where the input is a graph G and the question is whether it is possible to replace some induced subgraph H_1 of G on at most k vertices by a graph H_2 in $\mathcal{L}(H_1)$ so that the resulting graph belongs to $\mathcal{H}$. $\mathcal{L}$-Replacement to $\mathcal{H}$ can simulate many graph modification problems including vertex deletion, edge deletion/addition/edition/contraction, vertex identification, subgraph complementation, independent set deletion, (induced) matching deletion/contraction, etc. We present two algorithms. The first one solves $\mathcal{L}$-Replacement to $\mathcal{H}$ in time $2^{\mathsf{poly}(k)} \cdot |V(G)|^2$ for every minor-closed graph class $\mathcal{H}$, where poly is a polynomial whose degree depends on $\mathcal{H}$, under a mild technical condition on $\mathcal{L}$. This generalizes the results of Morelle, Sau, Stamoulis, and Thilikos [ICALP 2020, ICALP 2023] for the particular case of Vertex Deletion to $\mathcal{H}$ within the same running time. Our second algorithm is an improvement of the first one when $\mathcal{H}$ is the class of graphs embeddable in a surface of Euler genus at most g and runs in time $2^{\mathcal{O}(k^9)} \cdot |V(G)|^2$, where the $\mathcal{O}(\cdot)$ notation depends on g. To the best of our knowledge, these are the first parameterized algorithms with a reasonable parametric dependence for such a general family of graph modification problems to minor-closed classes.

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

Keywords and phrases Graph modification problems, Parameterized complexity, Graph minors, Flat Wall theorem, Irrelevant vertex technique, Algorithmic meta-theorem, Parametric dependence, Dynamic programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.7

Related Version *Full Version*: http://arxiv.org/abs/2504.16803

Funding *Laure Morelle*: French project ELiT (ANR-20-CE48-0008-01)
Ignasi Sau: French project ELiT (ANR-20-CE48-0008-01)
Dimitrios M. Thilikos: ANR project GODASse ANR-24-CE48-4377, French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027), and Franco-Norwegian project PHC AURORA 2024-25 (Projet n° 51260WL)

1 Introduction

A graph modification problem is typically determined by a target graph class $\mathcal{H}$ and a prescribed set of allowed local modifications $\mathcal{M}$, such as vertex/edge removal or edge addition/contraction or combinations of them, and the question is, given a graph G and an integer k, whether it is possible to transform G to a graph in $\mathcal{H}$ by applying at most k modification operations from $\mathcal{M}$. Graph modification problems are fundamental in algorithmic graph theory, as can be seen from the span of applications in domains as diverse

© Laure Morelle, Ignasi Sau, and Dimitrios M. Thilikos;
licensed under Creative Commons License CC-BY 4.0

as computational biology, computer vision, machine learning, networking, or sociology; see [19] and the references therein. Unfortunately, most of these problems are NP-complete [32, 49], and this justifies, among other approaches, to study them from the parameterized complexity viewpoint (see the monographs [10, 12, 15, 40] for an introduction to the field), where the number k of allowed modifications is taken as the parameter.

In the recent years, there has been a very active line of research about algorithmic meta-theorems for graph modification problems where the target class $\mathcal{H}$ is *minor-closed*, that is, closed under vertex deletion, edge deletion, and edge contraction. By Robertson and Seymour's seminal result [43], a minor-closed graph class $\mathcal{H}$ has a finite number of minor-obstructions, that is, graphs that are not in $\mathcal{H}$ but whose all proper minors are. Combined with a minor containment algorithm [30] (see also [26, 42]) running in almost-linear time, this implies that checking membership in a minor-closed graph class can be done in almost-linear time. For some modification problems where the target class $\mathcal{H}$ is minor-closed, such as vertex deletion, edge deletion, or vertex identification, the graphs G such that (G, k) is a yes-instance of the problem for a fixed k form a minor-closed graph class, which immediately implies an FPT-algorithm in almost-linear time for these problems. However, not all graph modification problems define a minor-closed graph class. For instance, edge contraction to planar graphs does not. Indeed, consider the graph $K_{3,4}^{+}$ obtained from $K_{3,4}$ by adding one edge e on the side with three vertices. Contracting e gives the planar graph $K_{2,4}$, but $K_{3,4}$ cannot be made planar by contracting one edge. Hence, some other algorithmic meta-theorems for graph modification problems to minor-closed graph classes were later introduced. Some of them are ad-hoc meta-theorems such as the one of Fomin, Golovach, and Thilikos [17] that gives quadratic FPT-algorithms for graph modification problems where $\mathcal{H}$ is the class of planar graphs and the modification is any combination of edge addition and edge deletion. Much more generally, there has been a recent line of research on model-checking on minor-closed graph classes [16, 47], which in particular implies quadratic FPT-algorithms for an extremely wide family of graph modification problems where the target class $\mathcal{H}$ is minor-closed. Unfortunately, all these algorithmic meta-theorems have a major drawback: the parametric dependence on the "amount of modification" is humongous; in fact, even a rough upper bound is not known.

On the other hand, another line of research has focused on optimizing the parametric dependence for some particular graph modification problems when the parameter is the solution size. When the target class $\mathcal{H}$ is minor-closed, such study usually does not go much beyond $\mathcal{H}$ being the class of forests and the class of union of paths [7, 9, 22, 23, 33, 39, 48]. To the best of our knowledge, only the case of vertex deletion has been studied in a a series of papers [18, 24, 25, 28, 29, 37, 38, 44, 45], focusing on optimizing the running time (both the dependence on k and n). In particular, when the minor-obstructions of $\mathcal{H}$ are connected and that one of them is planar, the currently fastest algorithm runs in time $2^{\mathcal{O}(k)} \cdot n \log^2 n$ [18], when $\mathcal{H}$ excludes a planar graph, in time $2^{\mathcal{O}(k)} \cdot n^2$ [28], when $\mathcal{H}$ is the class of planar graphs, in time $2^{\mathcal{O}(k \log k)} \cdot n$ [24], when $\mathcal{H}$ is the class of graphs embeddable in a surface of bounded genus, in time $2^{\mathcal{O}(k^2 \log k)} \cdot n^{\mathcal{O}(1)}$ [29], and when $\mathcal{H}$ is any minor-closed graph class, in time $2^{\mathsf{poly}(k)} \cdot n^2$ [38].

This article places itself in-between these two lines of research: we consider generic "meta-modification" operations (of course, much less generic than those of the currently most general algorithmic meta-theorem in [47], but still quite versatile), and we manage to achieve the same (very reasonable) parametric dependence as the currently best one for vertex deletion [38] when the target $\mathcal{H}$ is any minor-closed graph class. We hope that our work will trigger further research about *efficient* algorithmic meta-theorems for graph modification problems to minor-closed graph classes.

Our results. We define a graph modification problem, called $\mathcal{L}$-Replacement to $\mathcal{H}$ ($\mathcal{L}$-R-$\mathcal{H}$ for short), which, depending on the choice of the function $\mathcal{L}$, called a *replacement action*, can simulate vertex deletion, edge deletion, edge completion, edge edition, edge contraction, vertex identification, independent set deletion, matching deletion, matching contraction, star deletion, and subgraph complementation, to name a few (see section 3 for an exposition of some problems encompassed by our result). When $\mathcal{H}$ is minor-closed, we solve $\mathcal{L}$-R-$\mathcal{H}$ in time $2^{\mathsf{poly}(k)} \cdot n^2$ (Theorem 2), where the degree of the polynomial poly depends on the maximum size $s_{\mathcal{H}}$ of the minor-obstructions of $\mathcal{H}$. This is the same running time as the one achieved by the currently best algorithm for vertex deletion [38] (the degree of k in poly is the same as in [38] up to an extra additive constant of one that is absolutely negligible compared to the total degree that depends (wildly) on $s_{\mathcal{H}}$). For the other graph modification problems encompassed within $\mathcal{L}$-R-$\mathcal{H}$, to the authors' knowledge, the only minor-closed classes for which a good parametric dependence was previously known, if any, were the class of forests and the class of union of paths [9, 22, 33, 39, 48].

As it is usually the case concerning meta-theorems, the degree d of the polynomial poly in Theorem 2 is unfortunately huge. While we did not compute its exact value, we know that $d \geq 2^{2^{s_{\mathcal{H}}^{24}}}$. Nevertheless, d can improved for some specific target classes $\mathcal{H}$. The *Euler genus* of a surface Σ that is obtained from the sphere by adding h handles and c crosscaps is defined to be $c + 2h$. In particular, when $\mathcal{H}$ is the class of graphs embeddable in a surface of Euler genus at most g, we provide another algorithm solving $\mathcal{L}$-R-$\mathcal{H}$ in time $2^{\mathcal{O}(k^9)} \cdot n^2$ (Theorem 3), where the $\mathcal{O}(\cdot)$ notation depends on g. Note that, as opposed to Theorem 2, in Theorem 3 the contribution of the genus (that is, of the target graph class $\mathcal{H}$) does *not* affect the degree of the parameter k in the exponent.

Organization. In section 2 we give basic definitions and conventions, and we formally define the problem and state our results. In section 3 we provide a non-exhaustive list of problems generated by different instantiations of the replacement action $\mathcal{L}$, and hence encompassed by our results. In section 4 we present an overview of our techniques. Due to space limitations, all proofs have been deferred to the full version of this article. In section 5 we present some directions for further research.

2 Definition of the problem and formal statement of the results

In this section we formally define the $\mathcal{L}$-R-$\mathcal{H}$ problem and state our results, already informally discussed in the introduction. We use standard graph-theoretic notation, and complete preliminaries about graphs (including tree decompositions and minors) can be found in the full version. We define here the non-standard notions that are needed in order to state our results.

Minor-closed graph classes. A graph class $\mathcal{H}$ is *minor-closed* if, for each graph G and each minor H of G, the fact that $G \in \mathcal{H}$ implies that $H \in \mathcal{H}$. Given a collection of graphs $\mathcal{F}$, we denote by $\mathsf{exc}(\mathcal{F})$ the class of graphs that do not contain a graph in $\mathcal{F}$ as a minor. Obviously, $\mathsf{exc}(\mathcal{F})$ is minor-closed. A *(minor-)obstruction* of a graph class $\mathcal{H}$ is a graph F that is not in $\mathcal{H}$, but whose minors are all in $\mathcal{H}$. The set of all the obstructions of $\mathcal{H}$ is denoted by $\mathsf{obs}(\mathcal{H})$. By the seminal work of Robertson and Seymour [43], if $\mathcal{H}$ is a minor-closed graph class, then $\mathsf{obs}(\mathcal{H})$ is finite. Note that, if $\mathcal{F} = \mathsf{obs}(\mathcal{H})$, then $\mathsf{exc}(\mathcal{F}) = \mathcal{H}$. The *detail* of a graph G is $\max\{|V(G)|, |E(G)|\}$.

Ordered graphs. For the definitions of the next two paragraphs to be correct, we actually need to consider ordered graphs instead of graphs (see the "Graph modifications" paragraph). An *ordered graph* is a graph G equipped with a strict total order on $V(G)$, denoted by $<_G$. In other words, there exists an indexation $v_1, \ldots, v_n$ of the vertices of $V(G)$ such that $v_1 <_G v_2 <_G \cdots <_G v_n$. A subgraph H of an ordered graph G naturally comes equipped with the strict order $<_H$ such that, for each distinct $u, v \in V(H)$, $u <_H v$ if and only if $u <_G v$.

Replacement actions. The *any-replacement action* is the function $\mathcal{M}$ that maps each ordered graph H_1 to the collection $\mathcal{M}(H_1)$ of all the pairs (H_2, ϕ), where H_2 is an ordered graph and $\phi : V(H_1) \to V(H_2) \cup \{\emptyset\}$ is a function such that:

- $|V(H_2)| \leq |V(H_1)|$,
- for each $v \in V(H_2)$, $\phi^{-1}(v) \neq \emptyset$, and
- $<_{H_2}$ is the strict total order such that, for each distinct $v_1, v_2 \in V(H_2)$, we have $v_1 <_{H_2} v_2$ if and only if $u_1 <_{H_1} u_2$ where, for $i \in [2]$, u_i is the smallest vertex (according to $<_G$) in $\phi^{-1}(v_i)$.

A *replacement action* (abbreviated as *R-action*) is any function $\mathcal{L}$ that maps an ordered graph (called a *pattern*) H_1 to a non-empty collection $\mathcal{L}(H_1) \subseteq \mathcal{M}(H_1)$ of its possible *pattern transformations*. See Figure 1 for an illustration. The vertices of H_1 mapped by ϕ to the empty set are said to be *deleted*, and two vertices of H_1 mapped by ϕ to the same vertex of H_2 are said to be *identified*. Given $S \subseteq V(H_1)$, we set $\phi^+(S) = \phi(S) \setminus \{\emptyset\}$. Note that, if $\phi(S) = \{\emptyset\}$, then $\phi^+(S) = \{\emptyset\} \setminus \{\emptyset\} = \emptyset$.

Graph modifications. Let $\mathcal{L}$ be an R-action, let G be an ordered graph, and $S \subseteq V(G)$. Let $(H_2, \phi) \in \mathcal{L}(G[S])$. We denote by $G^S_{(H_2,\phi)}$ the graph obtained from the disjoint union of $G - S$ and H_2 by adding an edge $u\phi(v)$ for each $u \in V(G) \setminus S$ and each $v \in \phi^{-1}(V(H_2))$ such that $uv \in E(G)$. We equip $G' := G^S_{(H_2,\phi)}$ with the strict total order $<_{G'}$ such that $v_1 <_{G'} v_2$ if and only if $u_1 <_G u_2$ where, for $i \in [2]$, $u_i := v_i$ if $v_i \in V(G) \setminus S$, and u_i is the smallest vertex in $\phi^{-1}(v_i)$ if $v_i \in V(H_2)$. We also set $\mathcal{L}_S(G) = \{G^S_{(H_2,\phi)} \mid (H_2, \phi) \in \mathcal{L}(G[S])\}$. See Figure 1 for an illustration.

Note that we consider ordered graphs merely so that the correspondence between the vertices in S and the vertices in $V(H_2)$ is well-defined. We actually omit the order from the statements, but it will be implicitly assumed that vertices have a label that allows us to keep track of them during the modification procedure.

Let $\mathcal{L}$ be an R-action and $\mathcal{H}$ be a graph class. We define the following problem.

$\mathcal{L}$-Replacement to $\mathcal{H}$ ($\mathcal{L}$-R-$\mathcal{H}$)

Input:	A graph G and $k \in \mathbb{N}$.
Question:	Is there a set $S \subseteq V(G)$ of size at most k such that $\mathcal{L}_S(G) \cap \mathcal{H} \neq \emptyset$?

Such a set S is called *solution* of $\mathcal{L}$-R-$\mathcal{H}$ for the instance (G, k).

We will use the following observation, which implies that a no-instance for Vertex Deletion to $\mathcal{H}$ is also a no-instance for $\mathcal{L}$-R-$\mathcal{H}$.

▶ **Observation 1.** *Let $\mathcal{H}$ be a hereditary graph class, let $\mathcal{L}$ be an R-action, let G be a graph, and let $S \subseteq V(G)$. If $\mathcal{L}_S(G) \cap \mathcal{H} \neq \emptyset$, then $G - S \in \mathcal{H}$.*

Proof. Indeed, suppose that there is $(H_2, \phi) \in \mathcal{L}(G[S])$ such that $G^S_{(H_2,\phi)} \in \mathcal{H}$. Then, because $\mathcal{H}$ is hereditary, $G^S_{(H_2,\phi)} - \phi^+(S) = G - S \in \mathcal{H}$. ◀

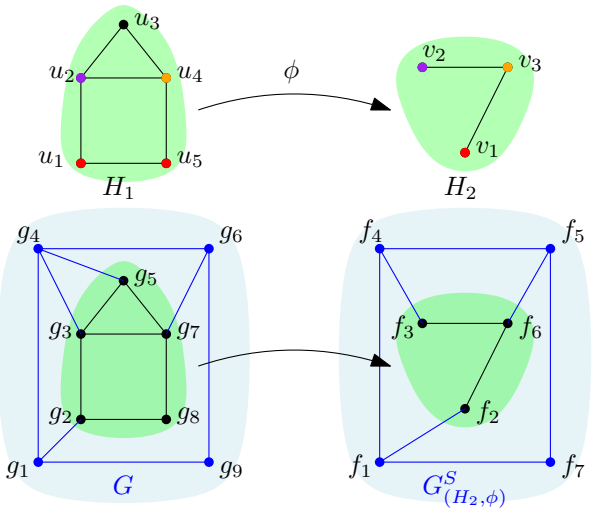


Figure 1 Example of $(H_2, \phi) \in \mathcal{L}(H_1)$ and of the graph modification $G^S_{(H,\phi)}$ where S is the set of black vertices of G. ϕ is represented by the colors, that is, $\phi(u_1) = \phi(u_5) = v_1$, $\phi(u_2) = \phi(v_2)$, $\phi(u_3) = \emptyset$, and $\phi(u_4) = v_3$. The order on the vertex sets of the depicted graphs is given by the corresponding labels.

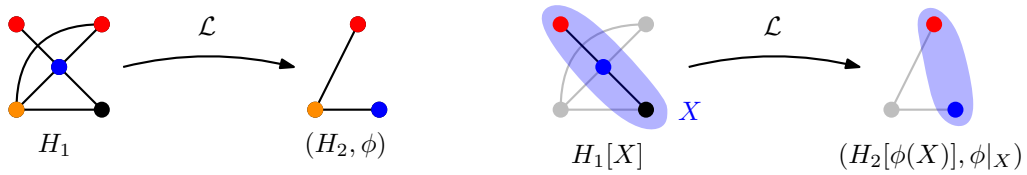


Figure 2 If $\mathcal{L}$ is hereditary, then a restriction of an allowed modification is also allowed.

Hereditary R-actions. An R-action is said to be *hereditary* if, for each ordered graph H_1, for each non-empty $X \subseteq V(H_1)$, and for each $(H_2, \phi) \in \mathcal{L}(H_1)$, we have $(H_2[\phi^+(X)], \phi|_X) \in \mathcal{L}(H_1[X])$. We say that $(H_2[\phi^+(X)], \phi|_X)$ is the *restriction* of (H_2, ϕ) to X. See Figure 2 for an illustration. Informally, an R-action is hereditary if, when a modification is allowed, then modifying "less" is allowed as well. For instance, if $\mathcal{L}$ allows us to delete *exactly* k vertices, then $\mathcal{L}$ also allows us to delete *at most* k vertices.

Some conventions. By convention, when there is no confusion, we set $n := |V(G)|$ and $m := |E(G)|$. In the rest of the paper, instead of considering a minor-closed graph class $\mathcal{H}$, we consider its obstruction set $\mathcal{F}$, and thus the minor-closed graph class $\mathsf{exc}(\mathcal{F})$. We define three constants depending on $\mathcal{F}$ that will be used throughout the paper whenever we consider such a collection $\mathcal{F}$. We define $a_{\mathcal{F}}$ as the minimum apex number of a graph in $\mathcal{F}$, we set $s_{\mathcal{F}} := \max\{|V(F)| \mid F \in \mathcal{F}\}$, and we define $\ell_{\mathcal{F}}$ to be the maximum detail of a graph in $\mathcal{F}$. Given a tuple $\mathbf{t} = (x_1, \ldots, x_\ell) \in \mathbb{N}^\ell$ and two functions $\chi, \psi : \mathbb{N} \to \mathbb{N}$, we write $\chi(n) = \mathcal{O}_{\mathbf{t}}(\psi(n))$ in order to denote that there exists a computable function $\phi : \mathbb{N}^\ell \to \mathbb{N}$ such that $\chi(n) = \mathcal{O}(\phi(\mathbf{t}) \cdot \psi(n))$. Notice that $s_{\mathcal{F}} \leq \ell_{\mathcal{F}} \leq s_{\mathcal{F}}(s_{\mathcal{F}} - 1)/2$, and thus $\mathcal{O}_{\ell_{\mathcal{F}}}(\cdot) = \mathcal{O}_{s_{\mathcal{F}}}(\cdot)$.

Our main result is the following.

▶ **Theorem 2.** *Let $\mathcal{F}$ be a finite collection of graphs and let $\mathcal{L}$ be a hereditary R-action. There is an algorithm that, given a graph G and $k \in \mathbb{N}$, runs in time $2^{\mathsf{poly}_{\mathcal{F}}(k)} \cdot n^2$ and either outputs a solution of $\mathcal{L}$-R-$\mathsf{exc}(\mathcal{F})$ for the instance (G, k) or reports a* no*-instance. Moreover, $\mathsf{poly}_{\mathcal{F}}$ is a polynomial whose degree depends on the maximum detail of a graph in $\mathcal{F}$.*

As mentioned in the introduction, the main result in [47] already implies that $\mathcal{L}$-R-$\mathcal{H}$ is solvable in time $f(k) \cdot n^2$ when $\mathcal{H}$ is minor-closed for some huge function f that is not even estimated. Our main contribution is an explicit and single-exponential dependence on k.

The degree of $\mathsf{poly}_{\mathcal{F}}(k)$ is quite big, but we can reduce it in some specific cases.

▶ **Theorem 3.** *Let $\mathcal{L}$ be a hereditary R-action and $\mathcal{H}$ be the class of graphs embeddable in a surface Σ of Euler genus at most g. There is an algorithm that, given a graph G and $k \in \mathbb{N}$, runs in time $2^{\mathcal{O}_g(k^9)} \cdot n^2$ and either outputs a solution of $\mathcal{L}$-R-$\mathcal{H}$ for the instance (G, k) or reports a* no*-instance.*

More generally, we study the annotated version of $\mathcal{L}$-R-$\mathcal{H}$. Let $\mathcal{L}$ be a hereditary R-action and $\mathcal{H}$ be a graph class. We define the following problem.

$\mathcal{L}$-Annotated Replacement to $\mathcal{H}$ ($\mathcal{L}$-AR-$\mathcal{H}$)

Input:	A graph G, a set of annotated vertices $S' \subseteq V(G)$, $(H_2', \phi') \in \mathcal{L}(G[S'])$, and $k \in \mathbb{N}$.
Question:	Is there a set $S \subseteq V(G)$ of size at most k and $(H_2, \phi) \in \mathcal{L}(G[S])$ such that (H_2', ϕ') is the restriction of (H_2, ϕ) to S' and $G^S_{(H_2,\phi)} \in \mathcal{H}$?

Obviously, we must have $S' \subseteq S$. Such a triple (S, H_2, ϕ) is called a *solution* of $\mathcal{L}$-AR-$\mathcal{H}$ for the instance (G, S', H_2', ϕ', k). An instance of $\mathcal{L}$-AR-$\mathcal{H}$ where $S' = \emptyset$ is an instance of $\mathcal{L}$-R-$\mathcal{H}$, so $\mathcal{L}$-AR-$\mathcal{H}$ generalizes $\mathcal{L}$-R-$\mathcal{H}$. Two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ are *equivalent* instances of $\mathcal{L}$-AR-$\mathcal{H}$ if $\mathcal{I}_1$ is a yes-instance of $\mathcal{L}$-AR-$\mathcal{H}$ if and only if $\mathcal{I}_2$ is a yes-instance of $\mathcal{L}$-AR-$\mathcal{H}$.

In fact, we prove stronger statements of Theorem 2 and Theorem 3 that apply to their respective annotated versions.

3 Problems generated by different instantiations of $\mathcal{L}$

Many graph modification problems correspond to $\mathcal{L}$-R-$\mathcal{H}$ for a specific R-action $\mathcal{L}$ and a specific target graph class $\mathcal{H}$. We give a few examples below. Let $\mathcal{H}$ be a minor-closed graph class. For instance, $\mathcal{H}$ could be the class of edgeless graphs, of forests, of graphs whose connected components have size at most k, of planar graphs, or of graphs embeddable in a surface Σ. Note that we do not mention Edge Addition to $\mathcal{H}$ (nor Edge Edition to $\mathcal{H}$) here, because when $\mathcal{H}$ is a minor-closed graph class, adding edges is "unnecessary", in the sense that the edge deletion variant has the same expressive power, and we can solve it. Note also that $\mathcal{L}$-R-$\mathcal{H}$, and thus in particular all problems of this section, was already known to be solvable in FPT-time (when $\mathcal{H}$ is minor-closed) by the result of [47]. However, as mentioned before, the parametric dependence is huge and not even explicit in [47].

Given a set A, we denote the identity function mapping each $a \in A$ to itself by id_A.

Vertex Deletion to $\mathcal{H}$

Input:	A graph G and $k \in \mathbb{N}$.
Question:	Is there a set $S \subseteq V(G)$ of size at most k such that $G - S \in \mathcal{H}$?

Vertex Deletion to $\mathcal{H}$ reduces to $\mathcal{L}_{\mathsf{vDel}}$-R-$\mathcal{H}$, where $\mathcal{L}_{\mathsf{vDel}}$ is the function that maps any graph H_1 to the singleton containing the empty graph and the constant function $\phi : V(H_1) \to \{\emptyset\}$. Vertex Deletion to $\mathcal{H}$ is already known [38] to be solvable within the same running time as the one of Theorem 2. Hence, the result of Theorem 2 is not an improvement for this specific problem, but it shows that our result is tight compared to the currently best known result for Vertex Deletion to $\mathcal{H}$.

Edge Deletion to $\mathcal{H}$

Input: A graph G and $k \in \mathbb{N}$.
Question: Is there a set $F \subseteq E(G)$ of size at most k such that $G - F \in \mathcal{H}$?

(G, k) is a yes-instance of Edge Deletion to $\mathcal{H}$ if and only if $(G, 2k)$ is a yes-instance of $\mathcal{L}_{\mathsf{eDel},k}$-R-$\mathcal{H}$, where $\mathcal{L}_{\mathsf{eDel},k}$ is the function that maps each graph H_1 to the set of pairs $(H_1 - F, \mathsf{id}_{V(H_1)})$ over all $F \subseteq E(G)$ of size at most k. Algorithms with a nice parametric dependence are only known for specific target classes $\mathcal{H}$. Namely, when $\mathcal{H}$ is the class of forests, Edge Deletion to $\mathcal{H}$ corresponds to Feedback Edge Set, which can be solved in constant time given that the size of a minimum feedback edge set is $m - n + 1$ (assuming the graph is connected). When $\mathcal{H}$ is the class of graphs that are a union of paths, then there is a linear kernel for the problem [34], as well as a FPT algorithm with parametric dependence on k at most 2^k [48]. We refer the reader to the survey of [9], as well as [13], for other results with explicit dependence on k when $\mathcal{H}$ is not a minor-closed graph class.

Given a graph G and a set of edges $F \subseteq E(G)$, we denote by G/F the graph obtained from G after contracting the edges in F.

Edge Contraction to $\mathcal{H}$

Input: A graph G and $k \in \mathbb{N}$.
Question: Is there a set $F \subseteq E(G)$ of size at most k such that $G/F \in \mathcal{H}$?

(G, k) is a yes-instance of Edge Contraction to $\mathcal{H}$ if and only if $(G, 2k)$ is a yes-instance of $\mathcal{L}_{\mathsf{Con},k}$-R-$\mathcal{H}$, where $\mathcal{L}_{\mathsf{Con},k}$ is the function that maps each graph H_1 to the set of pairs $(H_1/F, \phi)$ over all $F \subseteq E(G)$ of size at most k, where ϕ maps $v \in V(H_1)$ to the corresponding vertex of H_1/F. An explicit parametric dependence was given in [22] when $\mathcal{H}$ is a class of paths (running time $2^{k+o(k)} + n^{\mathcal{O}(1)}$) or the class of trees (running time $4.98^k \cdot n^{\mathcal{O}(1)}$). Though these classes are not minor-closed, we can easily extend these results to the case when $\mathcal{H}$ is the class of unions of paths or the class of forests (up to a 2^k factor). FPT-algorithms with an explicit parametric dependence were also studied when $\mathcal{H}$ is a collection of generalization and restriction of trees [2,3], or when $\mathcal{H}$ is the class of cactus graphs [31]. We refer the reader to [21] for more results when the target class is not minor-closed.

Vertex Identification to $\mathcal{H}$

Input: A graph G and $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq V(G)$ of size at most k and a partition $(X_1, \ldots, X_p)$ of S such that the graph obtained after identifying the vertices in X_i to a single vertex x_i, for $i \in [p]$, belongs to $\mathcal{H}$?

Vertex Identification to $\mathcal{H}$ reduces to $\mathcal{L}_{\mathsf{Id}}$-R-$\mathcal{H}$, where $\mathcal{L}_{\mathsf{Id}}$ is the function that maps each graph H_1 to the set of pairs (H_2, ϕ), where H_2 can be obtained from H_1 after identifying each X_i of a partition $(X_1, \ldots, X_p)$ of some set $S \subseteq V(H_1)$ to a single vertex x_i, and ϕ maps vertices of X_i to x_i and is the identity on $V(H_1) \setminus S$. Vertex Identification to $\mathcal{H}$ is known to admit a kernel of size $2k + 1$ when $\mathcal{H}$ is the class of forests [39]. To the authors' knowledge, this is the only known result for this problem.

Independent Set Deletion to $\mathcal{H}$

Input: A graph G and $k \in \mathbb{N}$.
Question: Is there an independent set $I \subseteq V(G)$ of size at most k such that $G - I \in \mathcal{H}$?

Independent Set Deletion to $\mathcal{H}$ reduces to $\mathcal{L}_{\mathsf{ISDel}}$-R-$\mathcal{H}$, where $\mathcal{L}_{\mathsf{ISDel}}$ is the function that maps any graph H_1 to the set of pairs $(H_1 - I, \phi)$ over all independent sets $I \subseteq V(H_1)$, where ϕ maps vertices of I to the empty set and is the identity on $V(H_1) \setminus I$.

When $\mathcal{H}$ is the class of forests, the problem is known to be solvable in time $3.62^k \cdot n^{\mathcal{O}(1)}$ [33]. Concerning other target classes that are not minor-closed, mainly bipartite graphs, let us mention [1, 6, 20].

To illustrate the versatility of $\mathcal{L}$-R-$\mathcal{H}$, let us present some other problems that can be defined by particular hereditary R-actions, though they do not seem to have been studied when parameterized by the solution size.

(Induced) Matching Deletion to $\mathcal{H}$

Input:	A graph G and $k \in \mathbb{N}$.
Question:	Is there an (induced) matching $M \subseteq E(G)$ of size at most k such that $G - M \in \mathcal{H}$?

(G, k) is a yes-instance of (Induced) Matching Deletion to $\mathcal{H}$ if and only if $(G, 2k)$ is a yes-instance of $\mathcal{L}_{\mathsf{mDel},k}$-R-$\mathcal{H}$, where $\mathcal{L}_{\mathsf{mDel},k}$ is defined similarly to $\mathcal{L}_{\mathsf{eDel},k}$ above, but for (induced) matchings. There are some results on Matching Deletion to $\mathcal{H}$ when $k = n$ and $\mathcal{H}$ is the class of forests [36, 41] or bipartite graphs (see [35] for a small survey).

(Induced) Matching Contraction to $\mathcal{H}$

Input:	A graph G and $k \in \mathbb{N}$.
Question:	Is there an (induced) matching $M \subseteq E(G)$ of size at most k such that $G/M \in \mathcal{H}$?

(G, k) is a yes-instance of (Induced) Matching Contraction to $\mathcal{H}$ if and only if $(G, 2k)$ is a yes-instance of $\mathcal{L}_{\mathsf{mCon},k}$-R-$\mathcal{H}$, where $\mathcal{L}_{\mathsf{mCon},k}$ is defined similarly to $\mathcal{L}_{\mathsf{Con},k}$ above, but for (induced) matchings.

Induced Star Deletion to $\mathcal{H}$

Input:	A graph G and $k \in \mathbb{N}$.
Question:	Is there a set $F \subseteq E(G)$ inducing a star $K_{1,k'}$ with $k' \leq k$ such that $G - F \in \mathcal{H}$?

(G, k) is a yes-instance of Star Deletion to $\mathcal{H}$ if and only if $(G, k+1)$ is a yes-instance of $\mathcal{L}_{\mathsf{StarDel},k}$-R-$\mathcal{H}$, where $\mathcal{L}_{\mathsf{StarDel},k}$ is the function that maps any graph H_1 to the set of pairs $(H_1 - F, \mathsf{id}_{V(H_1)})$ over all sets $F \subseteq E(G)$ inducing a subgraph of $K_{1,k}$.

Given a graph G, the *complement* of G, denoted by $\overline{G}$, is graph with vertex set $V(G)$ and edge set the edges that do not belong to $E(G)$.

Subgraph Complementation to $\mathcal{H}$

Input:	A graph G and $k \in \mathbb{N}$.
Question:	Is there a set $S \subseteq V(G)$ of size at most k such that the graph obtained after replacing $G[S]$ with its complement $\overline{G[S]}$ belongs to $\mathcal{H}$?

Subgraph Complementation to $\mathcal{H}$ reduces to $\mathcal{L}_{\mathsf{Comp}}$-R-$\mathcal{H}$, where $\mathcal{L}_{\mathsf{Comp}}$ is the function that maps any graph H_1 to the singleton containing the pair $(\overline{H_1}, \mathsf{id}_{V(H_1)})$. The problem was recently studied when $k = n$ for various target classes; we refer the reader to [4].

4 Overview of our techniques

To handle several modification problems at once, we adapt the vocabulary of Fomin, Golovach, and Thilikos [17], who introduced the notion of replacement action. Intuitively (see section 2 for the formal definition), a *replacement action* is a function $\mathcal{L}$ that maps a graph H_1 to a collection $\mathcal{L}(H_1)$ of pairs (H_2, ϕ) where H_2 is a graph with at most $|V(H_1)|$ vertices and ϕ maps each vertex of H_1 to either a vertex of H_2 or the empty set. Mapping a vertex of H_1 to the empty set corresponds to a deletion, while mapping several vertices to the same vertex of H_2 corresponds to an identification. Replacement actions were originally defined in [17] to solve a collection of graph modification problems where only edges are modified and where the target class is the class of planar graphs. Compared to [17], however, the size of H_2 may here be smaller than the size of H_1, which happens when deleting or identifying vertices, while in [17] it is required that $|V(H_1)| = |V(H_2)|$. Let us fix a replacement action $\mathcal{L}$ and a target graph class $\mathcal{H}$. Recall that the $\mathcal{L}$-Replacement to $\mathcal{H}$ ($\mathcal{L}$-R-$\mathcal{H}$) problem asks, given a graph G and $k \in \mathbb{N}$, whether there is an induced subgraph H_1 of size at most k in G and a pair $(H_2, \phi) \in \mathcal{L}(H_1)$ such that H_1 can be replaced by H_2 such that the resulting graph G' belongs to $\mathcal{H}$ (for $u \in V(G) \setminus V(H_1)$ and $v \in V(H_2)$, $uv \in E(G')$ if and only if there is $v' \in \phi^{-1}(v)$ such that $uv' \in E(G)$). For our techniques to work (see the "irrelevant vertex technique" paragraph below for more precision), we require our function $\mathcal{L}$ to be *hereditary*, which essentially means if H_2 is in $\mathcal{L}(H_1)$, then for any induced subgraph H_1' of H_1, the corresponding induced subgraph of H_2 is in $\mathcal{L}(H_1')$ (cf. section 2 for the formal definition and Figure 2 for an illustration). For instance, this implies that we can ask whether it is possible to do *at most* k edge editions to get a graph in $\mathcal{H}$, but we cannot ask whether it is possible to do *exactly* k edge editions to get a graph in $\mathcal{H}$.

High-level description of our algorithms. The techniques that we employ for our first algorithm (that is, when $\mathcal{H}$ is any minor-closed graph class) are strongly inspired by those used by Morelle, Sau, Stamoulis, Thilikos [38] for the particular case of vertex deletion (see also [44]), namely Vertex Deletion to $\mathcal{H}$, achieving the same running time. Nevertheless, in order to deal with our "meta-modification" operations, we need several new technical insights compared to the approach of [44], which we proceed to sketch. In a nutshell, the algorithm of [38] employs a win/win strategy that proceeds as follows:

- If the treewidth of the input graph is small (as a function of the parameter k), then solve the problem via a dynamic programming approach.
- If the treewidth of the input graph is big, then either
  - (*irrelevant vertex*) find a vertex v such that (G, k) and $(G - v, k)$ are equivalent instances, or
  - (*branching case*) find a set $A \subseteq V(G)$ of small size such that there exists $v \in A$ such that (G, k) and $(G - v, k - 1)$ are equivalent instances,

  and recurse.

Hence, we require three ingredients: one to solve the problem parameterized by treewidth, one to find an irrelevant vertex, and one to find an "obligatory set" A, all with a "reasonable" parametric dependence on k. Then, we need to construct an algorithm so that one of these three cases always applies and such that the overall running time is still within the desired bound, which is one of the most convoluted parts of the proof. In what follows we provide further insights about these steps, by first saying a few words about flat walls.

The need for annotation. Let S' be the set of vertices recursively guessed to be modified in the branching step. An advantage when the modification consists in vertex deletion is that we can simply recurse on $(G - S', k - |S'|)$. For the more general case of $\mathcal{L}$-R-$\mathcal{H}$, we cannot simply delete S', as the considered modification may be different from vertex deletion. We need 1) to guess how $G[S']$ is modified, that is, to guess $(H'_2, \phi') \in \mathcal{L}(G[S'])$ and 2) to remember S' and (H'_2, ϕ') in order to check that we eventually find a set $S \supseteq S'$ and an allowed modification $(H_2, \phi) \in \mathcal{L}(G[S])$ whose restriction to S' is (H'_2, ϕ') such that the modified graph is in $\mathcal{H}$. This is why we need to solve the *annotated version of the problem*, denoted by $\mathcal{L}$-AR-$\mathcal{H}$, where we add to the input a subset S' of vertices of G that are required to be part of H_1, as well as the modification (H'_2, ϕ') made on S'.

Dynamic programming algorithm in the case of bounded treewidth. Note that we cannot just use Courcelle's theorem [8], since we require a nice parametric dependence on k. Hence, we need to design our own dynamic programming algorithm to solve $\mathcal{L}$-AR-$\mathcal{H}$ parameterized by the treewidth and k. Essentially, the idea is to guess, in each bag $\beta(t)$ of the decomposition, the set S_t of vertices that are modified as well as how they are modified, and to reduce the size of the graph G_t induced by the bag t and its children using the representative-based technique of [5]. This technique is essentially based on the property that, given a graph G in a minor-closed graph class $\mathcal{H}$ with a boundary B, there is a graph R of *bounded size* with same boundary B, called the *representative of* G, such that, for any graph H glued on B to get $G \oplus H$ and $R \oplus H$, $G \oplus H \in \mathcal{H}$ if and only if $R \oplus H \in \mathcal{H}$. G_t does not belong to $\mathcal{H}$, so we cannot find a representative of G_t, but we find instead a representative of the graph $G'_t \in \mathcal{H}$ modified from G_t according to the guessed modification on S_t and the previously guessed modification on the children of t. Given that we may need to identify together vertices that are far apart in the tree decomposition, we need to remember throughout the algorithm the vertices that are guessed to be part of the solution. The fact that we keep information about these at most k vertices explains the dependence on k of the dynamic programming algorithm. More precisely, we prove the following result.

▶ **Theorem 4.** *Let $\mathcal{F}$ be a finite collection of graphs and $\mathcal{L}$ be an R-action. There is an algorithm that, given $k \in \mathbb{N}$, a graph G of treewidth at most w, a set $S' \subseteq V(G)$ of size at most k, and $(H'_2, \phi') \in \mathcal{L}(G[S'])$, in time $2^{\mathcal{O}_{\ell_{\mathcal{F}}}(k^2+(k+w)\log(k+w))} \cdot n$ either outputs a solution of $\mathcal{L}$-AR-exc($\mathcal{F}$) for the instance (G, S', H'_2, ϕ', k), or reports a* no*-instance.*

The above result parameterized by treewidth and k may be of independent interest, given that it implies an algorithm with a good parametric dependence on the treewidth and k for a number of graph modification problems. Note that the question of whether $\mathcal{L}$-R-$\mathcal{H}$ is FPT parameterized by only treewidth is open. Even Courcelle's theorem only implies a running time of $f(\mathsf{tw}, k) \cdot n$, given that the size of the CMSO formula expressing yes-instances of $\mathcal{L}$-R-$\mathcal{H}$ depends on k. Note that, in [38], the bounded treewidth part consists just in a black-box application of the algorithm of Baste, Sau, and Thilikos [5].

Flat walls. An essential tool of our approach is the notion of *flat wall*, originating in the work of Robertson and Seymour [42]. Informally speaking, a flat wall is a structure made up of (non-necessarily planar) pieces, called *flaps*, that are glued together in a bidimensional grid-like way defining the so-called *bricks* of the wall. While such a structure may not be planar, it enjoys topological properties similar to those of planar graphs, in the sense that two paths that are not routed entirely inside a flap cannot "cross", except at a constant-sized vertex set A whose vertices are called *apices*. Hence, flat walls are only "locally non-planar",

and after removing apices we can apply useful locality arguments, in the sense that two vertices that are in "distant" flaps should also be "distant" in the whole graph without the apices. In this article we apply some variants of one of the most celebrated results in the theory of Graph Minors by Robertson and Seymour [42,43], known as the *Flat Wall theorem* (see also [27,46] for recently proved variants), which informally states that graphs of large treewidth contain either a large clique minor or a large flat wall.

In order for our formal statements to be mathematically correct, we would need to introduce a number of notions originating in [46]. Unfortunately, in the attached full version several pages are required to provide all these technical notions. Due to space limitations, in this extended abstract we only provide intuitive descriptions of the main notions required to read the statement of the results, but we skip a number of technical terms (such as renditions, tilts, influence, regular flatness pairs, etc.) that are not the main focus of this sketch. All details can be found in the attached full version, and we refer the reader to [46] for a more detailed exposition of these definitions and the reasons for which they were introduced.

An *r-wall* is any graph W obtained from a so-called *elementary* r-wall $\bar{W}$ after subdividing edges: see Figure 3 for self-explanatory illustration of a 5-wall.

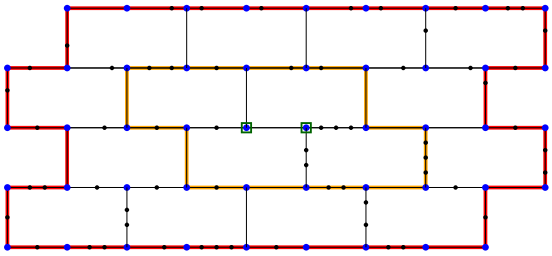


Figure 3 A 5-wall.

A flat wall is illustrated in Figure 4, where the flaps mentioned above correspond to the orange cells. The perimeter of a flat wall in a graph G separates $V(G)$ into two sets X and Y with Y containing the wall. The *compass* of a flat wall is $G[Y]$. For example, in Figure 4, X is the set of vertices in the green part, and Y the set of vertices in the orange part.

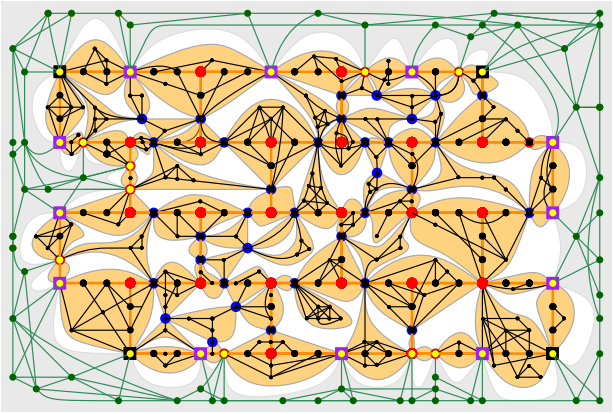


Figure 4 Illustration of flat wall, adapted from [5, Figure 4]). The edges of the subjacent wall are depicted in orange, defining the corresponding bricks.

In order to find an irrelevant vertex, we need to deal with *homogeneous* flat walls. Intuitively, homogeneous flat walls are flat walls that allow the routing of the same set of (topological) minors in the augmented flaps (i.e., the flaps together with the apex set)

"cropped" by each one of their bricks. Such a homogeneous wall can be detected in a big enough flat wall and this "homogeneity" property implies that some central part of a big enough homogeneous wall can be declared irrelevant.

Another very useful notion is that of a *canonical partition* of a graph G with respect to some wall W of G. Informally, this refers to a partition of the vertex set of G into bags that follow the grid-like structure of W; see Figure 5. Essentially, the goal is to be able to contract each of these bags to obtain a grid that is a minor of W, and thus of G. In particular, we prove (see Theorem 5 below) that if G contains as a minor a grid Γ along with a set A whose vertices have sufficiently many neighbors in the grid, then some vertex in A is obligatory. We use canonical partitions here to easily find such a structure given a wall of G.

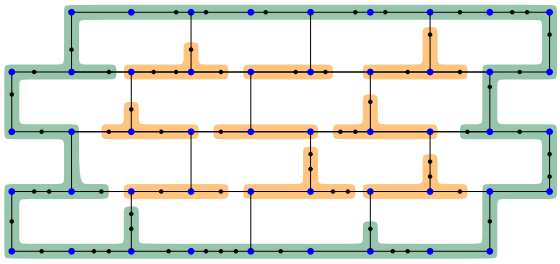


Figure 5 A 5-wall and its canonical partition $\mathcal{Q}$. The green bag is the external bag Q_{ext} and the orange bags are the internal bags of $\mathcal{Q}$. Contracting each internal bag of $\mathcal{Q}$ we obtain a (3×3)-grid.

Branching step. The branching case is not much different from what is done in [38] (originally from [45]): essentially, if there is a big enough wall W (cf. Figure 3) and a set A of vertices having many disjoint paths to W (cf. Figure 6), then some modification (H_A, ϕ_A) must happen in A and we can branch. Here, we however need to additionally prove that we must have $|\phi_A(A) \setminus \{\emptyset\}| < |A|$. We stress that it is important here to guess some modification in A that strictly decreases the size of A, so that, after applying this partial modification to G at the next step in the recursion, we will not find the exact same obligatory set A. Hence, in the algorithm with input (G, S', H_2', ϕ', k), at each step, either we find an irrelevant vertex and strictly decrease the size of G, or we branch and strictly increase the size of S'.

More precisely, the next result is the main technical ingredient in this part of the proof, essentially stating that a part of the solution S can be found in a set A of size $a_{\mathcal{F}}$ in which every vertex is adjacent to many vertices of a big enough wall. This is our "obligatory vertex" method. See Figure 6 for an illustration and Section 6 of the full version for the details.

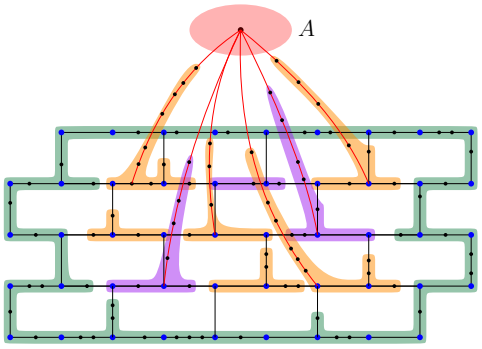


Figure 6 Illustration of Theorem 5.

▶ **Lemma 5.** *Let $\mathcal{F}$ be a finite collection of graphs and $\mathcal{L}$ be a hereditary R-action. There exist three functions $f_1, f_2, f_3 : \mathbb{N} \to \mathbb{N}$ such that the following holds. Let $k \in \mathbb{N}$. Let G be a graph, $S' \subseteq V(G)$ be a set of size at most k, and $(H_2', \phi') \in \mathcal{L}(G[S'])$. Suppose that $G' := G^{S'}_{(H_2', \phi')}$ contains a set $A \subseteq V(G')$ of size at least $a_{\mathcal{F}}$ and that there is a wall W in $G' - A$ of height $f_1(k)$. Suppose also that there is a W-canonical partition $\tilde{\mathcal{Q}}$ of $G' - A$ such that each vertex of A is adjacent to at least $f_2(k)$ many $f_3(k)$-internal bags of $\tilde{\mathcal{Q}}$. Then, for every solution (S, H_2, ϕ) of $\mathcal{L}$-AR-exc($\mathcal{F}$) for (G, S', H_2', ϕ'), it holds that $A' \neq \emptyset$, where $A' := (S \setminus S') \cap A$, and that $|\phi^+(A')| < |A'|$. Moreover $f_1(k) = \mathcal{O}_{s_{\mathcal{F}}}(k^2)$, $f_2(k) = \mathcal{O}_{s_{\mathcal{F}}}(k^3)$, and $f_3(k) = \mathcal{O}_{s_{\mathcal{F}}}(k^2)$.*

Finding an irrelevant vertex. As expected, we use the irrelevant vertex technique of Robertson and Seymour [42]. More specifically, we generalize the irrelevant vertex technique used in [38] (actually proved in [45]). This technique is based on the (intuitive but surprisingly hard to prove) fact that the central vertex of a homogeneous flat wall is always irrelevant. While our irrelevant vertex technique for $\mathcal{L}$-AR-$\mathcal{H}$ takes inspiration from [45], it is far more involved due to the annotation and the fact that we allow a wide variety of modifications. In particular, we need to redefine what it means to be homogeneous for a flat wall, to adapt it to our new setting. The previous definition was made to handle the case when we had to remove a small vertex set, called *apex set*, to find a flat wall, and more specifically to handle the fact that some vertices are possibly deleted from the apex set. Now, we also need to handle any other way the apex set may be modified, hence the new definition. The fact that we ask the replacement action $\mathcal{L}$ to be hereditary comes from the irrelevant vertex technique. Indeed, in order to prove that the central vertex v of a homogeneous flat wall W is irrelevant, we essentially prove that, for any solution (S, H_2, ϕ), we can delete a small part X of W containing v, and that the restriction of (S, H_2, ϕ) to $G - X$ is still a solution.

The following is the main technical result that we prove in this part, stating that an irrelevant vertex can be found in a big enough flat wall whose compass has bounded treewidth.

▶ **Theorem 6.** *Let $\mathcal{F}$ be a finite collection of graphs and $\mathcal{L}$ be a hereditary R-action. There exist a function $f_4 : \mathbb{N}^2 \to \mathbb{N}$, whose images are odd integers, and an algorithm with the following specifications:*

`Irrelevant-Vertex`$(G, S', H_2', \phi', k, A, a, W, \mathfrak{R}, t)$

Input: *Integers $k, a, t \in \mathbb{N}$, a graph G, a set $S' \subseteq V(G)$ of size at most k, $(H_2', \phi') \in \mathcal{L}(G[S'])$, a set $A \subseteq V(G')$ of size at most a, where $G' := G^{S'}_{(H_2, \phi)}$, and a regular flatness pair $(W, \mathfrak{R})$ of $G' - A$ of height at least $f_{??}(k, a)$ whose $\mathfrak{R}$-compass has treewidth at most t and does not intersect $\phi'(S')$.*

Output: *A vertex $v \in V(G) \setminus S'$ such that (G, S', H_2', ϕ', k) and $(G - v, S', H_2', \phi', k)$ are equivalent instances of $\mathcal{L}$-AR-exc($\mathcal{F}$). Moreover, $f_{??}(k, a) = \mathcal{O}_{a, \ell_{\mathcal{F}}}(k^c)$, where $c = \mathcal{O}_{a, \ell_{\mathcal{F}}}(1)$, and the algorithm runs in time $2^{\mathcal{O}_{a, \ell_{\mathcal{F}}}(k \log k + t \log t)} \cdot (n + m)$.*

We also prove the following result for the bounded genus case with a better dependence on k and a better running time. In this case, we do not ask for our flat wall to have bounded treewidth, but to have a planar embedding instead. Note that here, instead of a single vertex v, we might sometimes find an entire planar block of vertices V that is irrelevant.

▶ **Theorem 7.** *Let $\mathcal{L}$ be a hereditary R-action and $\mathcal{F}$ be the collection of obstructions of the graphs embeddable in a surface of genus at most g. There exist a function $f_5 : \mathbb{N} \to \mathbb{N}$, whose images are odd integers, and an algorithm with the following specifications:*

`Planar-Irrelevant-Vertex`$(G, S', H_2', \phi', k, W, \mathfrak{R})$

Input: *An integer $k \in \mathbb{N}$, a graph G, a set $S' \subseteq V(G)$ of size at most k, $(H_2', \phi') \in \mathcal{L}(G[S'])$, and a flatness pair $(W, \mathfrak{R} = (X, Y, P, C, \Gamma, \sigma, \pi))$ of $G^{S'}_{(H_2', \phi')}$ of height at least $f_{??}(k)$ whose $\mathfrak{R}$-compass does not intersect $\phi'(S')$ and is embeddable in a disk with $X \cap Y$ on the boundary.*

Output: *A non-empty set $Y \subseteq V(G) \setminus S'$ such that (G, S', H_2', ϕ', k) and $(G - Y, S', H_2', \phi', k)$ are equivalent instances of $\mathcal{L}$-AR-exc$(\mathcal{F})$.*

Moreover, $f_5(k) = \mathcal{O}(k)$ and the above algorithm runs in time $\mathcal{O}(n + m)$.

Piecing everything together. Finally, we combine the three ingredients discussed above to find an algorithm for $\mathcal{L}$-AR-$\mathcal{H}$. We essentially proceed as follows. Let (G, S', H_2', ϕ', k) be the instance we want to solve, and G' be obtained by doing the modification (H_2', ϕ') of S'. In the first steps, we either find that G has small treewidth, where we can use our dynamic programming algorithm to conclude, or that G' contains a wall W. Given W, we first try to find a flat wall W' inside, with all the necessary conditions to find an irrelevant vertex. If we manage to do so, we remove the irrelevant vertex and recurse. Otherwise, through a greedy procedure, we try to find an obligatory vertex set A with many disjoint paths to W in G'. If we find such a set, we branch and recurse. If not, we manage to argue that we must have a no-instance, and conclude.

The special case of bounded genus. Our second algorithm (Theorem 3), when $\mathcal{H}$ is a class of graphs embeddable in a surface of bounded Euler genus, uses two additional ideas to get an improved running time. The first one is that here, the obligatory set A is a singleton. Indeed, the size of A is the size of the minimum number of vertices one can remove from an obstruction of $\mathcal{H}$ to make it planar. It is well known that, when $\mathcal{H}$ is such a class, there is some integer t depending on the Euler genus such that $K_{3,t} \notin \mathcal{H}$, and thus, $|A| = 1$. In particular, this implies that we do not need to branch on A, but that we instead immediately find an obligatory vertex. The second idea is about homogeneous flat walls. In the running time $2^{\mathsf{poly}(k)} \cdot n^2$ of the first algorithm, the degree of poly essentially corresponds to the size of the required flat wall to find a big enough homogeneous flat wall, and hence an irrelevant vertex, inside of it. In the case where $\mathcal{H}$ is the class of graphs embeddable in a surface of Euler genus at most g, we prove that we can find a homogeneous flat wall inside a flat wall of smaller size, hence the improved running time. To do so, we prove that, after some preliminary processing, a flat wall that is furthermore embeddable in a disk with the perimeter on its boundary is already homogeneous. Hence, our second algorithm proceeds similarly to the first one, but if we find a flat wall W' in G', we divide W' into $k + 1$ disjoint smaller flat walls and check whether they belong to $\mathcal{H}$. By the pigeonhole principle, one of them, W_i, does not contain a modified vertex and must thus be in $\mathcal{H}$, otherwise we return a no-instance. Then, we argue, using a result from [11] to guarantee additional properties of the planar embedding that are needed for technical reasons, that we can find a smaller flat wall W_i' in W_i with a *planar embedding* (even if the genus of the target graph class is strictly positive). Hence, we find an irrelevant vertex in W_i' and conclude.

5 Conclusion

For a large family of graph modification problems involving a bounded number of vertices, if the target class $\mathcal{H}$ is minor-closed, we provided an algorithm solving the problem in time $2^{\mathsf{poly}(k)} \cdot n^2$. This is actually the same running time as the best known running time for Vertex Deletion to $\mathcal{H}$ [38]. For the other graph modification problems encompassed by our result, such as Edge Deletion to $\mathcal{H}$, Edge Contraction to $\mathcal{H}$, Vertex Identification to $\mathcal{H}$, or Independent Set Deletion to $\mathcal{H}$, the only minor-closed $\mathcal{H}$ for which an algorithm with an explicit parametric dependence in the solution size was known, to the authors' knowledge, were the class of forests and the class of union of paths. Other problems, such as Matching Deletion to $\mathcal{H}$, Matching Contraction to $\mathcal{H}$, Induced Star Deletion to $\mathcal{H}$, or Subgraph Complementation to $\mathcal{H}$, were not even considered yet from the parameterized complexity viewpoint, other than in the meta-theorem of [47].

The degree of $\mathsf{poly}(k)$ in the running time comes from the irrelevant vertex technique and is quite huge. In the bounded genus case, we reduce the running time to $2^{\mathcal{O}(k^9)} \cdot n^2$ thanks to some improvement on the irrelevant vertex technique. This does not match the parametric dependence in the running time of $2^{\mathcal{O}(k^2 \log k)} \cdot n^{\mathcal{O}(1)}$ for Vertex Deletion to $\mathcal{H}$ [29] for $\mathcal{H}$ of bounded genus, though we possibly have a better dependence on n. To the authors' knowledge, this is the first bounded genus result with an explicit parametric dependence in the solution size for the other graph modification problems encompassed by our result.

Improving more the parametric dependence in the general case would certainly require coming up with new techniques. On the other hand, given the recent results of [30] for minor containment, it is worth studying whether the quadratic dependence on n could be improved to an almost-linear dependence while maintaining a good dependence on k. Note that the approach of [30] heavily uses Courcelle's theorem [8], which would require to be translated to a plausibly very involved dynamic programming algorithm to keep a good parametric dependence on k.

Given that we require the replacement action $\mathcal{L}$ to be hereditary for our irrelevant vertex technique to work, we unfortunately restrict the graph modification problems that we solve. For instance, Planar Subgraph Isomorphism can be expressed as an $\mathcal{L}$-R-Planar problem for a specific $\mathcal{L}$, which is not hereditary. Hence, we do not encompass this problem in our general algorithm, while such an algorithm is provided in [17], where the constraint about $\mathcal{L}$ being hereditary is not required. While most of the "reasonable" modification problems correspond to a hereditary replacement action, it is worth investigating whether our result can be extended to non-hereditary replacement actions.

Here, we only consider modifications that affect a bounded number of vertices of the input graph. This is necessary as we want k to decrease by one each time we find an obligatory vertex (or, more precisely, as we want the size of the increasingly guessed partial solution to be bounded by k), so that the depth of the branching tree is bounded. Some relevant graph modification problems, however, such as Elimination Distance to $\mathcal{H}$ [38] or $\mathcal{H}$-Treewidth [14] (where we want to delete a vertex set X whose "torso" has bounded treedepth or treewidth, respectively, such that $G - X \in \mathcal{H}$), consider a modification that affects a set of vertices that may have *unbounded* size. In this case, the branching method does not seem applicable. However, the irrelevant vertex technique still works, and provided that we have a dynamic programming for graphs of bounded treewidth, an algorithm can still be designed in some cases, but with a worse parametric dependence on k. This is what is done, for instance, in [38] for Elimination Distance to $\mathcal{H}$. Therefore, we could consider extending the results of this paper to (some kinds of) modifications involving sets of vertices or edges of unbounded size.

References

1 Akanksha Agrawal, Pallavi Jain, Lawqueen Kanesh, Pranabendu Misra, and Saket Saurabh. Exploring the kernelization borders for hitting cycles. In *Proc. of the 13th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 115 of *LIPIcs*, pages 14:1–14:14, 2018. doi:10.4230/LIPICS.IPEC.2018.14.

2 Akanksha Agrawal, Lawqueen Kanesh, Saket Saurabh, and Prafullkumar Tale. Paths to trees and cacti. *Theoretical Computer Science*, 860:98–116, 2021. doi:10.1016/J.TCS.2021.01.033.

3 Akanksha Agrawal, Saket Saurabh, and Prafullkumar Tale. On the parameterized complexity of contraction to generalization of trees. *Theory of Computing Systems*, 63(3):587–614, 2019. doi:10.1007/S00224-018-9892-Z.

4 Dhanyamol Antony, Sagartanu Pal, and R. B. Sandeep. Algorithms for subgraph complementation to some classes of graphs. *Information Processing Letters*, 188:106530, 2025. doi:10.1016/J.IPL.2024.106530.

5 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting Minors on Bounded Treewidth Graphs. IV. An Optimal Algorithm. *SIAM Journal on Computing*, 52(4):865–912, 2023. doi:10.1137/21M140482X.

6 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Independent feedback vertex set for p_5-free graphs. *Algorithmica*, 81(4):1342–1369, 2019. doi:10.1007/S00453-018-0474-X.

7 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

8 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

9 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr Golovach. A survey of parameterized algorithms and the complexity of edge modification. *Computer Science Review*, 48:100556, 2023. doi:10.1016/j.cosrev.2023.100556.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. The bidimensional theory of bounded-genus graphs. *SIAM Journal on Discrete Mathematics*, 20(2):357–371, 2006. doi:10.1137/040616929.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

13 Maël Dumas, Anthony Perez, Mathis Rocton, and Ioan Todinca. Polynomial kernels for edge modification problems towards block and strictly chordal graphs. *CoRR*, abs/2201.13140, 2022. arXiv:2201.13140.

14 Eduard Eiben, Robert Ganian, Thekla Hamm, and O-joung Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. *Journal of Computer and System Sciences*, 121:57–75, 2021. doi:10.1016/j.jcss.2021.04.005.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

16 Fedor V. Fomin, Petr A. Golovach, Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. Compound logics for modification problems. In *Proc. of the 50th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 61:1–61:21, 2023. doi:10.4230/LIPICS.ICALP.2023.61.

17 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Modification to planarity is fixed parameter tractable. In *Proc. of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 126 of *LIPIcs*, pages 28:1–28:17, 2019. doi:10.4230/LIPIcs.STACS.2019.28.

18 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proc. of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 470–479, 2012. doi:10.1109/FOCS.2012.62.

19 Fedor V. Fomin, Saket Saurabh, and Neeldhara Misra. Graph modification problems: A modern perspective. In *Proc. of the 9th International Workshop on Frontiers in Algorithmics (FAW)*, volume 9130 of *LNCS*, pages 3–6. Springer, 2015. doi:10.1007/978-3-319-19647-3_1.

20 Hanna Furmanczyk, Marek Kubale, and Stanislaw P. Radziszowski. On bipartization of cubic graphs by removal of an independent set. *Discrete Applied Mathematics*, 209:115–121, 2016. doi:10.1016/J.DAM.2015.10.036.

21 Petr A. Golovach, Pim van 't Hof, and Daniël Paulusma. Obtaining planarity by contracting few edges. *Theoretical Computer Science*, 476:38–46, 2013. doi:10.1016/J.TCS.2012.12.041.

22 Pinar Heggernes, Pim van 't Hof, Benjamin Lévêque, Daniel Lokshtanov, and Christophe Paul. Contracting graphs to paths and trees. *Algorithmica*, 68(1):109–132, 2014. doi:10.1007/S00453-012-9670-2.

23 Yoichi Iwata and Yusuke Kobayashi. Improved analysis of highest-degree branching for feedback vertex set. *Algorithmica*, 83(8):2503–2520, 2021. doi:10.1007/S00453-021-00815-W.

24 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *Proc. of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1802–1811, 2014. doi:10.1137/1.9781611973402.130.

25 Ken-ichi Kawarabayashi. Planarity allowing few error vertices in linear time. In *Proc. of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 639–648, 2009. doi:10.1109/FOCS.2009.45.

26 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012. doi:10.1016/j.jctb.2011.07.004.

27 Ken-ichi Kawarabayashi, Robin Thomas, and Paul Wollan. A new proof of the flat wall theorem. *Journal of Combinatorial Theory, Series B*, 129:204–238, 2018. doi:10.1016/j.jctb.2017.09.006.

28 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms*, 12(2):21:1–21:41, 2016. doi:10.1145/2797140.

29 Tomasz Kociumaka and Marcin Pilipczuk. Deleting Vertices to Graphs of Bounded Genus. *Algorithmica*, 81(9):3655–3691, 2019. doi:10.1007/s00453-019-00592-7.

30 Tuukka Korhonen, Michal Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time. In *Proc. of the 65th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 53–61, 2024. doi:10.1109/FOCS61266.2024.00014.

31 R. Krithika, Pranabendu Misra, and Prafullkumar Tale. A single exponential-time FPT algorithm for cactus contraction. *Theoretical Computer Science*, 954:113803, 2023. doi:10.1016/J.TCS.2023.113803.

32 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

33 Shaohua Li and Marcin Pilipczuk. An improved FPT algorithm for independent feedback vertex set. *Theory of Computing Systems*, 64(8):1317–1330, 2020. doi:10.1007/S00224-020-09973-W.

34 Wenjun Li, Qilong Feng, Jianer Chen, and Shuai Hu. Improved kernel results for some FPT problems based on simple observations. *Theoretical Computer Science*, 657:20–27, 2017. doi:10.1016/J.TCS.2016.06.012.

35 Carlos V. G. C. Lima, Dieter Rautenbach, Uéverton S. Souza, and Jayme Luiz Szwarcfiter. On the computational complexity of the bipartizing matching problem. *Annals of Operations Research*, 316(2):1235–1256, 2022. doi:10.1007/S10479-021-03966-9.

36 Carlos Vinícius G. C. Lima, Dieter Rautenbach, Uéverton S. Souza, and Jayme Luiz Szwarcfiter. Decycling with a matching. *Information Processing Letters*, 124:26–29, 2017. doi:10.1016/J.IPL.2017.04.003.

37 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/s00453-010-9484-z.

38 Laure Morelle, Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. Faster parameterized algorithms for modification problems to minor-closed classes. *TheoretiCS*, 3, 2024. doi:10.46298/THEORETICS.24.19.

39 Laure Morelle, Ignasi Sau, and Dimitrios M. Thilikos. Vertex identification to a forest. *CoRR*, 2024. doi:10.48550/arXiv.2409.08883.

40 Rolf Niedermeier. *Invitation to fixed parameter algorithms*, volume 31. Oxford University Press, 2006. doi:10.1093/ACPROF:OSO/9780198566076.001.0001.

41 Fábio Protti and Uéverton S. Souza. Decycling a graph by the removal of a matching: new algorithmic and structural aspects in some classes of graphs. *Discrete Mathematics & Theoretical Computer Science*, 20(2), 2018. doi:10.23638/DMTCS-20-2-15.

42 Neil Robertson and Paul D. Seymour. Graph Minors. XIII. The Disjoint Paths Problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

43 Neil Robertson and Paul D. Seymour. Graph Minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

44 Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. k-apices of minor-closed graph classes. II. Parameterized Algorithms. *ACM Transactions on Algorithms*, 18(3):21:1–21:30, 2022. doi:10.1145/3519028.

45 Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. k-apices of minor-closed graph classes. I. Bounding the obstructions. *Journal of Combinatorial Theory, Series B*, 161:180–227, 2023. doi:10.1016/J.JCTB.2023.02.012.

46 Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. A more accurate view of the Flat Wall Theorem. *Journal of Graph Theory*, 107(2):263–297, 2024. doi:10.1002/jgt.23121.

47 Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. Parameterizing the quantification of CMSO: model checking on minor-closed graph classes. In *Proc. of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3728–3742, 2025. doi:10.1137/1.9781611978322.124.

48 Dekel Tsur. Faster deterministic algorithm for co-path set. *Information Processing Letters*, 180:106335, 2023. doi:10.1016/J.IPL.2022.106335.

49 Mihalis Yannakakis. Node- and Edge-Deletion NP-Complete Problems. In *Proc. of the 10th Annual ACM Symposium on Theory of Computing*, pages 253–264, 1978. doi:10.1145/800133.804355.

Fast Computation of k-Runs, Parameterized Squares, and Other Generalised Squares

Yuto Nakashima
Department of Informatics, Kyushu University, Fukuoka, Japan

Jakub Radoszewski
Institute of Informatics, University of Warsaw, Poland

Tomasz Waleń
Institute of Informatics, University of Warsaw, Poland

Abstract

A k-mismatch square is a string of the form XY where X and Y are two equal-length strings that have at most k mismatches. Kolpakov and Kucherov [*Theor. Comput. Sci.*, 2003] defined two notions of k-mismatch repeats, called k-repetitions and k-runs, each representing a sequence of consecutive k-mismatch squares of equal length. They proposed algorithms for computing k-repetitions and k-runs working in $\mathcal{O}(nk \log k + \mathsf{output})$ time for a string of length n over an integer alphabet, where output is the number of the reported repeats. We show that $\mathsf{output} = \mathcal{O}(nk \log k)$, both in case of k-repetitions and k-runs, which implies that the complexity of their algorithms is actually $\mathcal{O}(nk \log k)$. We apply this result to computing parameterized squares.

A parameterized square is a string of the form XY such that X and Y parameterized-match, i.e., there exists a bijection f on the alphabet such that $f(X) = Y$. Two parameterized squares XY and $X'Y'$ are equivalent if they parameterized match. Recently Hamai et al. [SPIRE 2024] showed that a string of length n over an alphabet of size σ contains less than $n\sigma$ non-equivalent parameterized squares, improving an earlier bound by Kociumaka et al. [*Theor. Comput. Sci.*, 2016]. We apply our bound for k-mismatch repeats to propose an algorithm that reports all non-equivalent parameterized squares in $\mathcal{O}(n\sigma \log \sigma)$ time. We also show that the number of non-equivalent parameterized squares can be computed in $\mathcal{O}(n \log n)$ time. This last algorithm applies to squares under any substring compatible equivalence relation and also to counting squares that are distinct as strings. In particular, this improves upon the $\mathcal{O}(n\sigma)$-time algorithm of Gawrychowski et al. [CPM 2023] for counting order-preserving squares that are distinct as strings if $\sigma = \omega(\log n)$.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithm, k-mismatch square, parameterized square, order-preserving square, maximum gapped repeat

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.8

Related Version *Full Version*: `https://arxiv.org/abs/2509.02179`

Funding *Yuto Nakashima*: JSPS KAKENHI Grant Number JP25K00136.
Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2022/46/E/ST6/00463.

1 Introduction

A string of the form XX, for any string X, is called a *square* (or a *tandem repeat*). Squares are a classic notion in combinatorics on words (see, e.g., the early works of Thue [53] on avoidability of square substrings), text algorithms (starting from an algorithm for computing square substrings by Main and Lorentz [40]), and bioinformatics (see Gusfield's book [22]). A string of length n contains at most n distinct square substrings [8] and all of them can be computed in $\mathcal{O}(n)$ time [4, 9, 14, 23] or even $\mathcal{O}(n/\log_\sigma n)$ time [10] assuming that the string is over an integer alphabet of size σ. A maximal sequence of consecutive squares in a string


© Yuto Nakashima, Jakub Radoszewski, and Tomasz Waleń;
licensed under Creative Commons License CC-BY 4.0



is called a run (or a generalised run; cf. Section 2); see [33]. A string of length n contains at most n runs [3] and they can all be computed in $\mathcal{O}(n)$ time [3, 17]. Our work is devoted to efficient algorithms for computing known generalizations of squares and runs: k-mismatch squares represented as k-runs or k-repetitions, parameterized squares, and generalised squares that include parameterized squares, order-preserving squares, and Cartesian-tree squares.

We assume that positions in a string X are numbered from 1 to $|X|$, so that $X[i]$ is the ith character of X. For integers i, j such that $1 \le i \le j \le |X|$, by $X[i\,..\,j] = X[i\,..\,j+1)$ we denote the substring composed of characters $X[i], X[i+1], \ldots, X[j]$. We use similar notation for integer intervals: $[i\,..\,j] = [i\,..\,j+1) = \{i, i+1, \ldots, j\}$.

We say that a length-n string is over an integer alphabet if its letters belong to $[0\,..\,n^{\mathcal{O}(1)}]$. We use the word-RAM model of computation.

1.1 k-Mismatch Squares and k-Runs

For two equal-length strings X and Y, their Hamming distance is defined as $d_H(X,Y) = |\{i \in [1\,..\,|X|] : X[i] \neq Y[i]\}|$. A *$k$-mismatch square* (also known under the name of k-mismatch tandem repeat) is a string XY such that $|X| = |Y|$ and $d_H(X,Y) \le k$. Let T be a string of length n. A *k-run of period ℓ* in T (cf. [38]) is a maximal substring $T[a\,..\,b)$ such that $T[i\,..\,i+2\ell)$ is a k-mismatch square for every $i \in [a\,..\,b-2\ell]$; see Figure 1. Maximality means that the k-run is extended to the right and left as much as possible provided that the definition is still satisfied.

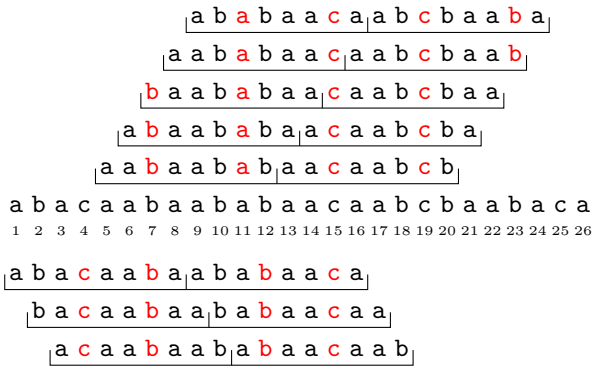


Figure 1 String $T[1\,..\,26]$ contains two 2-runs with period 8, $T[1\,..\,18]$ and $T[5\,..\,24]$. The two 2-runs represent three 2-mismatch squares (below) and five 2-mismatch squares (above), respectively; mismatches are shown in red. Strings $T[x\,..\,x+16)$ for $x \in \{4, 10, 11\}$ are not 2-mismatch squares.

Landau, Schmidt and Sokol [39] showed an algorithm for computing k-runs in a string of length n that works in $\mathcal{O}(nk\log(n/k))$ time; hence, the total number of k-runs reported is $\mathcal{O}(nk\log(n/k))$. Kolpakov and Kucherov [34, 35] showed that all k-runs (called there runs of k-mismatch tandem repeats) in a string of length n can be computed in $\mathcal{O}(nk\log k + \mathsf{output})$ time, where output is the number of k-runs. Many other algorithms for computing approximate tandem repeats under various metrics, in the context of computational biology and with the aid of statistical methods and heuristics, were proposed [6, 7, 15, 16, 28, 31, 36, 43, 45, 46, 47, 48, 49, 50, 51, 52, 55, 56]. In Section 2 we show the following theorem.

▶ **Theorem 1.** *A string of length n contains $\mathcal{O}(nk\log k)$ k-runs.*

Theorem 1 provides the first $\mathcal{O}(n)$ upper bound on the number of k-runs for $k = \mathcal{O}(1)$ and implies that the algorithm of Kolpakov and Kucherov computing k-runs actually works in $\mathcal{O}(nk\log k)$ time. Actually, we show a stronger condition that a string T of length n

contains $\mathcal{O}(nk \log k)$ *uniform* k-runs. Intuitively, a uniform k-run is a maximal sequence of consecutive k-mismatch squares of the same length in which the mismatches of all squares are at the same positions. (See Section 2 for a formal definition.) Our bound on the number of uniform k-runs implies the bound for k-runs as well as an $\mathcal{O}(nk \log k)$ upper bound on the number of k-repetitions as defined in [34, 35] (called there k-mismatch globally defined repetitions).

To prove Theorem 1, we explore a combinatorial relation between k-runs and maximum gapped repeats [32] and apply the optimal $\mathcal{O}(n\alpha)$ bound on the number of maximal α-gapped repeats in a length-n string [20, 27].

1.2 Parameterized Squares

For a string X, by $\mathsf{Alph}(X)$ we denote the set of characters of X. Two strings X, Y *parameterized match* if $|X| = |Y|$ and there is a bijection $f : \mathsf{Alph}(X) \mapsto \mathsf{Alph}(Y)$ such that $f(X) = Y$ (i.e., $|X| = |Y|$ and $f(X[i]) = Y[i]$ for all $i \in [1\,..\,|X|]$). A *parameterized square* (*p-square*, in short) is a string XY such that X parameterized matches Y (see Figure 2). Two p-squares XY and $X'Y'$ are called *equivalent* if they parameterized match.

Parameterized matching was introduced by Baker [2] motivated by applications in code refactoring and plagiarism detection. The notion of p-squares was introduced by Kociumaka, Radoszewski, Rytter, and Waleń [30] who showed that a string of length n over alphabet of size σ contains at most $2\sigma!n$ non-equivalent p-squares. They also considered avoidability of parameterized cubes. The bounds from [30] were recently improved by Hamai, Taketsugu, Nakashima, Inenaga, and Bannai [24]; they showed that a length-n string over alphabet of size σ contains less than σn non-equivalent p-squares. This bound automatically implies that such a string contains at most $\sigma! \cdot \sigma \cdot n$ p-squares that are distinct as strings (improving the $2(\sigma!)^2 n$ upper bound from [30]). We show that all non-equivalent p-squares can be computed efficiently; our approach also extends to p-squares that are distinct as strings. See Section 4.

▶ **Theorem 2.** *All non-equivalent p-square substrings in a string of length n over alphabet of size σ can be computed in $\mathcal{O}(n\sigma \log \sigma)$ time.*

All p-square substrings that are distinct as strings can be computed in $\mathcal{O}(n\sigma \log \sigma + \mathsf{output})$ time, where output is the number of p-squares reported.

By the above discussion, all p-square substrings that are distinct as strings can be computed in $\mathcal{O}(n(\sigma+1)!)$ time. The key ingredient in the algorithm behind Theorem 2 is a relation between p-squares and σ-mismatch squares (and uniform σ-runs).

1.3 Generalised Squares under Substring Consistent Equivalence Relations

We then show an alternative algorithm for counting p-squares whose complexity does not depend on the alphabet size σ. The algorithm is stated for squares under any *substring consistent equivalence relation* (*SCER* in short). An SCER is a relation $\approx$ on strings such that $X \approx Y$ implies that (1) $|X| = |Y|$ and (2) $X[i\,..\,j] \approx Y[i\,..\,j]$ for all $1 \le i \le j \le |X|$; see [42]. Known examples of SCERs include parameterized matching, order-preserving matching [29, 37], Cartesian tree matching [44], and palindrome pattern matching [26]. Two strings X and Y *order-preserving match* if $|X| = |Y|$, sets $\mathsf{Alph}(X)$ and $\mathsf{Alph}(Y)$ are totally ordered, and there is a bijection $f : \mathsf{Alph}(X) \mapsto \mathsf{Alph}(Y)$ that is increasing (i.e., if $x < y$ then $f(x) < f(y)$) such that $f(X) = Y$. Strings X, Y *Cartesian-tree match* if they have the same shape of a Cartesian tree (cf. [54]). Finally, strings X and Y *palindrome match* if for all $1 \le i \le j \le |X|$, $X[i\,..\,j]$ is a palindrome if and only if $Y[i\,..\,j]$ is a palindrome.

An $\approx$-*square* is a string XY such that $X \approx Y$ under SCER $\approx$. Thus p-squares and order-preserving squares [30] (op-squares, in short) as well as Cartesian-tree squares [44] (CT-squares) and squares in the sense of palindrome matching (palindrome-squares) are $\approx$-squares for the respective SCERs $\approx$. See Figure 2 for an example.

$$S = \underbrace{1\ 3\ \overbrace{2\ 2\ 4\ 3}^{Y}}_{X}\ \underbrace{4\ 4\ 1\ 2\ \overbrace{3\ 2\ 3}^{Y}}_{Z}$$

Figure 2 For string $S = 1322434412323$, $X = 132\ 243$ is an op-square (hence, automatically, a p-square and a CT-square), $Y = 2243\ 4412$ is a p-square, and $Z = 412\ 323$ is a CT-square. Among the three substrings, only Z is *not* a palindrome-square, as its second half is a palindrome whereas its first half is not.

In a prefix consistent equivalence relation (PCER), condition (2) of an SCER only needs to hold for $i = 1$. An SCER is a PCER. We say that $\mathcal{E} : \Sigma^* \to \mathbb{Z}$ is an *encoding function* if

$$X \approx Y \quad \Leftrightarrow \quad (\,|X| = |Y| \ \wedge\ \forall_{i \in [1\,..\,|X|]}\, \mathcal{E}(X[1\,..\,i]) = \mathcal{E}(Y[1\,..\,i])\,). \tag{1}$$

▶ **Observation 3.** *For every PCER $\approx$ on strings over integer alphabet there exists an encoding function.*

Proof. Let $\mathcal{E}(X)$ be defined as the number of the equivalence class under $\approx$ of X among all strings of length $|X|$. Let us verify that $\mathcal{E}$ satisfies equivalence (1). ($\Rightarrow$) If $X \approx Y$, then, by definition, $|X| = |Y|$ and $X[1\,..\,i] \approx Y[1\,..\,i]$ for all $i \in [1\,..\,|X|]$. Then indeed $\mathcal{E}(X[1\,..\,i]) = \mathcal{E}(Y[1\,..\,i])$. ($\Leftarrow$) Taking $i = |X| = |Y|$, we obtain $\mathcal{E}(X) = \mathcal{E}(Y)$. As $|X| = |Y|$, we have $X \approx Y$. ◀

The encoding function defined in the proof of Observation 3 could be hard to compute efficiently for a particular PCER $\approx$. It is also stronger, as it satisfies $X \approx Y \ \Leftrightarrow\ (\,|X| = |Y| \ \wedge\ \mathcal{E}(X) = \mathcal{E}(Y))$. Luckily, for the aforementioned known SCERs efficient encoding functions are known, as shown in the following Example 4.

For an encoding function $\mathcal{E}$, let $\pi^{\mathcal{E}}(n)$, $\rho^{\mathcal{E}}(n)$ be integer sequences such that for a string T of length n over integer alphabet, after $\pi^{\mathcal{E}}(n)$ preprocessing time, $\mathcal{E}(T[i\,..\,j])$ for any i, j can be computed in $\rho^{\mathcal{E}}(n)$ time.

▶ **Example 4.** For parameterized matching, there exists a classic *prev*-encoding, see [2], that is an encoding function:

$$\mathit{prev}(X) = \begin{cases} |X| - i & X[i] = X[|X|] \text{ and } X[j] \neq X[|X|] \text{ for all } j \in [i+1\,..\,|X|) \\ 0 & \text{otherwise} \end{cases}$$

The *prev*-encodings of all prefixes of a string T can be computed by sorting all pairs $(T[i], i)$. If T is over an integer alphabet, this is performed in $\pi^{prev}(n) = \mathcal{O}(n)$ time. Then $\mathit{prev}(T[i\,..\,j])$ can be computed from $\mathit{prev}(T[1\,..\,j])$ in $\rho^{prev}(n) = \mathcal{O}(1)$ time.

For order-preserving matching, one can use an encoding as pairs $(\alpha(X), \beta(X))$. Here

$$\alpha(X) \text{ is the largest } j < |X| \text{ such that } X[j] = \max\{X[k] \ :\ k \in [1\,..\,|X|),\ X[k] \leq X[|X|]\},$$

and if there is no such j, then $\alpha(X) = 0$. Similarly,

$$\beta(X) \text{ is the largest } j < |X| \text{ such that } X[j] = \min\{X[k] \ :\ k \in [1\,..\,|X|),\ X[k] \geq X[|X|]\},$$

and $\beta(X) = 0$ if no such j exists; see [13, 29, 37]. As we require the encoding function to return integers, we can set $\mathcal{E}(X) = \alpha(X) \cdot |X| + \beta(X)$, since $\alpha(X), \beta(X) \in [0\,..\,|X|)$. Then [13, Lemma 4] implies that $\mathcal{E}$ is an encoding function for order-preserving matching and [13, Lemma 24] gives $\pi^{\mathcal{E}}(n) = \mathcal{O}(n\sqrt{\log n})$, $\rho^{\mathcal{E}}(n) = \mathcal{O}(\log n / \log\log n)$.

For Cartesian-tree matching, [44, Theorem 1] shows that the following *parent-distance* representation:

$$PD(X) = \begin{cases} |X| - \max_{1 \le j < |X|}\{j \,:\, X[j] \le X[|X|]\} & \text{if such } j \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

is an encoding function. Encodings of all prefixes of a string T can be computed in $\pi^{PD}(n) = \mathcal{O}(n)$ time using a folklore nearest smaller value algorithm. In [44, Section 5.2] it is noted that $PD(T[i\,..\,j])$ can be computed from $PD(T[1\,..\,j])$ in $\rho^{PD}(n) = \mathcal{O}(1)$ time.

By X^R we denote the reverse of string X. In [26, Lemma 2] it is shown that the length of the longest suffix palindrome of X, formally:

$$Lpal(X) = \max\{|X| - k + 1 \,:\, X[k\,..\,|X|] = X[k\,..\,|X|]^R\},$$

is an encoding function for palindrome matching.

Computing encodings of substrings $Lpal(T[i\,..\,j])$ can be reduced in linear time to 2D range successor queries as follows.

In the 2D range successor problem we are given n points in an $n \times n$ grid and we are to answer queries that, given a rectangle in the grid, report a point in the rectangle with the smallest first coordinate, if any. Such queries can be answered in $\mathcal{O}(\log\log n)$ time after $\mathcal{O}(n\sqrt{\log n})$ preprocessing [18].

In the reduction, we compute all maximal palindromes in T in $\mathcal{O}(n)$ time using Manacher's algorithm [41]. Let us show how we deal with odd-length palindromes; the approach for even-length palindromes is analogous. For a maximal palindrome $T[c - r\,..\,c + r]$ with $c \in [1\,..\,n]$ and $r \ge 0$, we create a point $(c, c + r)$. To compute $Lpal(T[i\,..\,j])$, it suffices to find the point in the rectangle $[\lceil (i + j)/2 \rceil\,..\,j] \times [j\,..\,\infty)$ with the smallest first coordinate. If x is the sought coordinate, the longest odd-length suffix palindrome of $T[i\,..\,j]$ has length $2(j - x) + 1$.

By [18], we get $\pi^{Lpal}(n) = \mathcal{O}(n\sqrt{\log n})$ and $\rho^{Lpal}(n) = \mathcal{O}(\log\log n)$.

In Section 5 we show the next theorem on efficient counting of $\approx$-squares.

▶ **Theorem 5.** *Let $\approx$ be an SCER and $\mathcal{E}$ be its encoding function. The number of non-equivalent $\approx$-square substrings in a length-n string can be computed in $\mathcal{O}(\pi^{\mathcal{E}}(n) + n\rho^{\mathcal{E}}(n) + n\log n)$ time.*

The number of $\approx$-square substrings in a length-n string that are distinct as strings can be computed in the same time complexity.

Consequently, in a length-n string, the numbers of non-equivalent p-squares, op-squares, CT-squares, and palindrome-squares, as well as the numbers of the respective squares that are distinct as strings can be computed in $\mathcal{O}(n\log n)$ time, as the respective SCERs enjoy encoding functions with $\pi^{\mathcal{E}}(n) = \mathcal{O}(n\log n)$ and $\rho^{\mathcal{E}}(n) = \mathcal{O}(\log n)$ as shown in Example 4. In particular, the algorithm of Theorem 5 in the case that $\sigma = \omega(\log n)$ improves upon the $\mathcal{O}(n\sigma)$-time algorithm of Gawrychowski, Ghazawi, and Landau [19] for reporting (thus, counting) op-squares that are distinct as strings. Moreover, the obtained complexity for counting non-equivalent p-squares (p-squares that are distinct as strings, respectively) is better than the one in Theorem 2 if $\sigma = \omega(\log n / \log\log n)$ ($\sigma = \Omega(\log\log n)$, respectively).

Theorem 5 involves constructing a $\approx$-suffix tree data structure (see Section 3 for a definition).

2 Upper Bound on the Number of k-Runs

This section introduces an upper bound on the number of k-runs (proof of Theorem 1) which implies their efficient computation by the algorithm of Kolpakov and Kucherov [34, 35]. A position i in string T is called an *ℓ-mismatching position* if $i \in [1\,..\,n-\ell]$ and $T[i] \neq T[i+\ell]$. In particular, $T[i\,..\,i+2\ell)$ is a k-mismatch square if and only if $T[i\,..\,i+\ell)$ contains at most k ℓ-mismatching positions. Let us formally define a uniform k-run.

▶ **Definition 6.** *A substring $T[a\,..\,b)$ is called a* uniform k-run of period ℓ *if $b-a \geq 2\ell$ and the sets $\{j \in [i\,..\,i+\ell) : T[j] \neq T[j+\ell]\}$ of ℓ-mismatching positions for all $i \in [a\,..\,b-2\ell]$ have cardinality at most k and are all the same. We are interested in maximal uniform k-runs.*

A *gapped repeat* in a string T is a fragment of the form UVU, for $|U| > 0$. Fragments denoted by U are called *arms* of the gapped repeat (left and right arm). The *period* of the gapped repeat is defined as $|UV|$. An α-gapped repeat (for $\alpha \geq 1$) in T is a gapped repeat UVU such that $|UV| \leq \alpha|U|$. An (α-)gapped repeat is called *maximal* if its arms cannot be extended simultaneously with the same character to the right or to the left. A maximal (α-)gapped repeat will be called an (α-)MGR, for short. Gawrychowski, I, Inenaga, Köppl, and Manea [20] showed that, for a real $\alpha \geq 1$, a string of length n contains $\mathcal{O}(n\alpha)$ α-MGRs.

A *generalised run* is a pair $(T[x\,..\,y), p)$ such that p satisfies $2p \leq y-x$ and (1) $x = 1$ or $T[x-1\,..\,y)$ does not have period p, and (2) $y = |T|+1$ or $T[x\,..\,y]$ does not have period p. That is, a generalised run is a 0-run. A run is a generalised run in which p is the smallest period of $T[x\,..\,y)$. A string of length n contains less than n runs and $\mathcal{O}(n)$ generalised runs [3].

▶ **Definition 7.** *We say that an MGR $UVU = T[x\,..\,y)$ with period $\ell = |UV|$ in T* induces *a uniform k-run $T[a\,..\,b)$ of period ℓ if $[x\,..\,y-\ell) \cap [a\,..\,b-\ell) \neq \emptyset$.*

We further say that a generalised run $(T[x\,..\,y), \ell)$ induces *a uniform k-run $T[a\,..\,b)$ of period ℓ if $[x\,..\,y-\ell) \cap [a\,..\,b-\ell) \neq \emptyset$.*

Let us emphasize that the uniform k-run does not need to be a substring of the MGR or generalised run. Intuitively, an MGR induces a uniform k-run if the left half of at least one k-mismatch square of the k-run contains a position of the left arm of the MGR; see Figure 3. If an MGR or a generalised run induces a uniform k-run, we say that the k-run is *induced* by the MGR or run, respectively. We also say that the MGR or generalised run induces all k-mismatch squares in the uniform k-run that it induces.

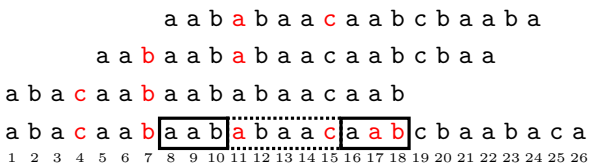


Figure 3 String $T[1\,..\,26]$ from Figure 1 with 8-mismatching positions shown in red. The string contains three uniform 2-runs with period 8, $T[1\,..\,18]$, $T[5\,..\,22]$, and $T[8\,..\,24]$ (in Figure 1, the last two form a single 2-run). They are all induced by a 3-MGR UVU for $U = T[8\,..\,10] = T[16\,..\,18]$ and $V = T[11\,..\,15]$. In the proof of Lemma 8 for this MGR, we have $L = \{7, 4, 0\}$ and $R = \{11, 15, 17\}$.

▶ **Lemma 8.** *An MGR induces at most $2k+1$ uniform k-runs.*

Proof. Consider an MGR $T[x\,..\,y)$ with period ℓ. Let R be the set of $k+1$ smallest ℓ-mismatching positions in $[y-\ell\,..\,y)$ in T; if there are no such $k+1$ positions, we select all ℓ-mismatching positions in $[y-\ell\,..\,y)$ in T to the set R. Similarly, let L be the set of $k+1$ largest ℓ-mismatching positions in $[x-\ell+1\,..\,x)$ in T; if there are no $k+1$ such positions, we select all ℓ-mismatching positions in $[x-\ell+1\,..\,x)$. If $|R|\leq k$, we insert y to R. If $|L|\leq k$, we insert $x-\ell$ to L. This way, if a k-mismatch square $T[i\,..\,i+2\ell)$ is induced by the MGR $T[x\,..\,y)$, then $i\in I$ for $I=[\min L+1\,..\,\max R-\ell]$ (as otherwise the left half of the square would contain more than k ℓ-mismatching positions or would not touch $T[x\,..\,y-\ell)$).

Let us consider a window of length ℓ sliding left-to-right over T with the left end point of the window located in interval I. There are at most $2k$ moments when the number of ℓ-mismatching positions in the window changes. Between them, we obtain uniform k-runs if the number of ℓ-mismatching positions in the window is at most k. ◀

An analogous lemma for generalised runs is obtained with the same proof (replacing "MGR" by "generalised run"); see Figure 4.



Figure 4 String $T[1\,..\,26]$ with its 6-mismatching positions shown in red. A generalised run $T[6\,..\,21]$ with period 6 (rectangle) induces five uniform 2-runs with period 6 shown above. In this example, the generalised run corresponds to a run with period 3.

▶ **Lemma 9.** *A generalised run induces at most $2k+1$ uniform k-runs.*

A uniform k-run can be induced by several MGRs and generalised runs.

▶ **Example 10.** The middle uniform 2-run $T[5\,..\,22]$ in Figure 3 is induced also by two other MGRs with period 8, $T[5\,..\,14]$ (with arms $T[5\,..\,6]=T[13\,..\,14]=\mathtt{aa}$) and $T[12\,..\,22]$ (with arms $T[12\,..\,14]=T[20\,..\,22]=\mathtt{baa}$).

For a gapped repeat UVU, the fraction $\frac{|UV|}{|U|}$ is called the *gap ratio*. That is, the gap ratio is the minimum α such that UVU is an α-gapped repeat. We assign to each gapped repeat UVU in T a *weight* equal to the reciprocal of its gap ratio, that is, $\frac{|U|}{|UV|}$.

▶ **Lemma 11.** *If $\ell\geq 4k$ and $T[a\,..\,b)$ is a uniform k-run of period ℓ, then it is induced by a generalised run or by some number of $(2k+2)$-MGRs of total weight at least $\frac{1}{4}$.*

Proof. Let $p_1,\ldots,p_t$ be all ℓ-mismatching positions among $[a\,..\,b-\ell)$, listed in an increasing order. We define sentinel p_0 being the maximum ℓ-mismatching position that is smaller than a; if there is no such position, we set $p_0=0$. Similarly, we define sentinel p_{t+1} as the minimum ℓ-mismatching position that is at least $b-\ell$; if no such position exists, we set $p_{t+1}=n+1-\ell$.

Let us fix $s\in[0\,..\,t]$. We have $U_s:=T[p_s+1\,..\,p_{s+1})=T[p_s+\ell+1\,..\,p_{s+1}+\ell)$, so $W_s:=T[p_s+1\,..\,p_{s+1}+\ell)$ has period ℓ. Moreover, we have (1) $p_s=0$ or $T[p_s]\neq T[p_s+\ell]$ and (2) $p_{s+1}+\ell=n+1$ or $T[p_{s+1}]\neq T[p_{s+1}+\ell]$. If $0<|U_s|<\ell$, then W_s is an MGR with arms equal to U_s. If $|U_s|\geq\ell$, W_s is a generalised run. The MGR or generalised run induces $T[a\,..\,b)$ unless $p_{s+1}=a$ or $p_s=b-\ell-1$.

Assume that the uniform k-run $T[a\,..\,b)$ is not induced by a generalised run. If a string W_s, for $s \in [0\,..\,t]$, is not a $(2k+2)$-MGR of period ℓ, the length of its left arm U_s is smaller than $\frac{\ell}{2k+2}$. Hence, for strings W_s that are not $(2k+2)$-MGRs, the total length of their left arms U_s is at most $\frac{\ell}{2}$. The remaining $(2k+2)$-MGRs among W_s have total length of left arms at least $(b-a-\ell)-\frac{\ell}{2}-k \geq \frac{\ell}{2}-k$ (accounting for the at most k ℓ-mismatching positions $p_1,\ldots,p_t$) which is at least $\frac{\ell}{4}$ by the assumption of the lemma. This means that the total weight of $(2k+2)$-MGRs that induce the k-run, defined as the sum of the lengths of their left arms divided by their periods, all equal to ℓ, is at least $\frac{1}{4}$. ◀

We show a theorem that implies Theorem 1.

▶ **Theorem 12.** *A string of length n contains $\mathcal{O}(nk\log k)$ uniform k-runs.*

Proof. Let T be a string of length n. Let us consider a bipartite graph $G=(V_1\cup V_2,E)$ such that vertices in V_1 are uniform k-runs in T, vertices in V_2 are $(2k+2)$-MGRs and generalised runs in T, and there is an edge $uw\in E$, for $u\in V_1$ and $w\in V_2$, if k-run u is induced by the MGR or generalised run w. Our goal is to bound $|V_1|$ from above.

Let us remove from G all vertices in V_1 that are uniform k-runs with period at most $4k$. There are at most $4kn$ such k-runs (as there are at most $4kn$ substrings of T of length at most $4k$). Let us also remove from G all vertices in V_2 that are generalised runs and all vertices in V_1 that are adjacent to at least one of the removed vertices in V_2. By [3], this way at most $1.5n$ vertices are removed from V_2, so by Lemma 9, $\mathcal{O}(nk)$ vertices are removed from V_1. Let $G'=(V_1'\cup V_2',E')$ be the remaining graph. We assume that each edge has a weight equal to the weight of the MGR in V_2' it is incident to. In order to bound $|V_1'|$, we will consider the *total weight of edges* in E'.

For each $\alpha\in[2\,..\,2k+2]$, let x_α be the number of α-MGRs in T that are not $(\alpha-1)$-MGRs. Each such MGR has weight at most $\frac{1}{\alpha-1}$ and $2k+1\leq 3k$ edges incident to it (cf. Lemma 8). Thus the sum of weights of edges in E' is bounded from above by

$$\sum_{\alpha=2}^{2k+2}\frac{3kx_\alpha}{\alpha-1} \tag{2}$$

The improved upper bound on the number of α-MGRs of I and Köppl [27] implies that

$$\sum_{i=2}^{\alpha} x_i < 13n\alpha \quad \text{for every } \alpha\in[2\,..\,2k+2] \tag{3}$$

We show how to bound the number (2) in general.

▷ Claim 13. For every sequence $(x_2,\ldots,x_{2k+2})$ of non-negative integers that satisfies the condition (3), the value (2) is $\mathcal{O}(nk\log k)$.

Proof. As long as there exists $\alpha\in[3\,..\,2k+2]$ such that $x_\alpha>13n$, we select $i\in[2\,..\,\alpha)$ such that $x_i<13n$ or $i=2$ and $x_i<26n$, decrement x_α and increment x_i. Such i exists by (3) and the pigeonhole principle. The operation does not change any lhs in (3) and increases (2).

In the end, we have $x_2\leq 26n$ and $x_\alpha\leq 13n$ for all $\alpha\in[3\,..\,2k+2]$. Hence, (2) is bounded as:

$$\sum_{\alpha=2}^{2k+2}\frac{3kx_\alpha}{\alpha-1} \;\leq\; 78k+39nk\sum_{\alpha=2}^{2k+1}\frac{1}{\alpha} \;\leq\; 78k+39nk(\ln(2k+1)+1) \;=\; \mathcal{O}(nk\log k). \qquad \triangleleft$$

By the claim, the sum of weights of edges in G' is $\mathcal{O}(nk \log k)$. By Lemma 11, for each uniform k-run with period $\ell \geq 4k$ that is not induced by a generalised run, the total weight of edges incident to it is at least $\frac{1}{4}$. This concludes that $|V_1'| = \mathcal{O}(nk \log k)$, so the total number of uniform k-runs (across all periods $\ell \in [1\,..\,\lfloor n/2 \rfloor]$) is $|V_1| = \mathcal{O}(nk \log k)$. ◀

A k-run of period ℓ contains a prefix being a uniform k-run of period ℓ, constructed by taking k-mismatch squares of length 2ℓ at subsequent positions as long as their sets of ℓ-mismatching positions are the same. By maximality, no two k-runs with equal period start at the same position, so this way each k-run produces a different uniform k-run. Thus the number of k-runs in T does not exceed the number of uniform k-runs, which proves Theorem 1. By the same argument, each k-repetition implies a unique uniform k-run and so the number of k-repetitions is $\mathcal{O}(nk \log k)$. (We refer the reader to the definition of k-repetition in [34, 35, 38] as this notion is not used below.)

3 ≈-Suffix Trees and Their Applications

We introduce useful tools for the next two sections. Cole and Hariharan [11] defined a *quasi-suffix collection* as a collection of strings $S_1, S_2, \ldots, S_n$ that satisfies the following conditions:

1. $|S_1| = n$ and $|S_i| = |S_{i-1}| - 1$.
2. No S_i is a prefix of another S_j.
3. Suppose strings S_i and S_j have a common prefix of length $\ell > 0$. Then S_{i+1} and S_{j+1} have a common prefix of length at least $\ell - 1$.

A quasi-suffix collection is specified implicitly by a character oracle that given i, j returns $S_i[j]$. They obtain the following result.

▶ **Theorem 14** ([11]). *The compacted trie of a quasi-suffix collection of n strings can be constructed in $\mathcal{O}(n)$ expected time assuming that the character oracle works in $\mathcal{O}(1)$ time.*

For a string T of length n, the *≈-suffix tree* of T is a compacted trie of strings $\mathit{Code}(T[i\,..\,n])\#$, $i \in [1\,..\,n+1]$, where $\mathit{Code}(X) = \mathcal{E}(X[1])\mathcal{E}(X[1\,..\,2]) \cdots \mathcal{E}(X)$ for a string X and $\#$ is an end-marker that is not an encoding of any string. We extend the sequences π, ρ so that for any string $T \in [0\,..\,\sigma)^n$, after $\pi^{\mathcal{E}}(n, \sigma)$ preprocessing time, $\mathcal{E}(T[i\,..\,j])$ for any i, j can be computed in $\rho^{\mathcal{E}}(n, \sigma)$ time. By $\gamma^{\mathcal{E}}(n, \sigma)$ we denote an upper bound on the total number of distinct characters in the strings stored in the ≈-suffix tree of a string in $[0\,..\,\sigma)^n$. Theorem 14 implies the following corollary. (A similar result, but only for order-preserving equivalence, was shown in [19, Lemma 16].)

▶ **Corollary 15.** *Let ≈ be an SCER and $\mathcal{E}$ be its encoding function. The ≈-suffix tree of a string in $[0\,..\,\sigma)^n$ can be constructed in worst case time:*

$$\mathcal{O}(\pi^{\mathcal{E}}(n, \sigma) + n\rho^{\mathcal{E}}(n, \sigma) + \min\{n \log^2 \log n / \log \log \log n,\ n\gamma^{\mathcal{E}}(n, \sigma)\}).$$

Proof. Let us verify that strings $\mathit{Code}(T[i\,..\,n])\#$ satisfy the conditions 1-3 of a quasi-suffix collection. Condition 1 for $n+1$ is obvious. Condition 2 follows due to the end-marker. As for condition 3, assume $\mathsf{LCP}(S_i, S_j) = \ell > 0$ and let $i \neq j$ as otherwise the conclusion is trivial. Then $\ell < |S_i|, |S_j|$ because of the end-marker. By equivalence (1), we have $T[i\,..\,i+\ell) \approx T[j\,..\,j+\ell)$. Because ≈ is an SCER, we have $T[i+1\,..\,i+\ell) \approx T[j+1\,..\,j+\ell)$. Hence, $\mathsf{LCP}(S_{i+1}, S_{j+1}) \geq \ell - 1$ by equivalence (1).

An oracle for the quasi-suffix collection S_i answers queries in $\mathcal{O}(\rho^{\mathcal{E}}(n, \sigma))$ time after $\mathcal{O}(\pi^{\mathcal{E}}(n, \sigma))$ preprocessing. The only source of randomness in the algorithm behind Theorem 14 is the need to maintain, for each explicit node of the current tree, a dictionary

indexed by the next character on an outgoing edge. If we store the respective characters per each node in a dynamic predecessor data structure of Andersson and Thorup [1], the total space remains $\mathcal{O}(n)$ and each predecessor query is answered in $\mathcal{O}(\log^2 \log n / \log\log\log n)$ time in the worst case. Alternatively, one can store all the children of a node in a list, to achieve $\mathcal{O}(\gamma^{\mathcal{E}}(n,\sigma))$ space per an explicit node and $\mathcal{O}(\gamma^{\mathcal{E}}(n,\sigma))$ time for a predecessor query. The complexity follows. ◀

▶ Remark 16. For SCERs mentioned in Example 4, we obtain the following time complexities for constructing $\approx$-suffix trees in the respective settings: $\mathcal{O}(n \log^2 \log n / \log\log\log n)$ for parameterized matching and Cartesian tree matching, $\mathcal{O}(n\sqrt{\log n})$ for palindrome matching (that matches the complexity from [26]), and $\mathcal{O}(n \log n / \log\log n)$ for order-preserving matching (a faster, $\mathcal{O}(n\sqrt{\log n})$-time construction was proposed in [13]).

For two strings X and Y, by $\mathsf{LCP}^{\approx}(X,Y)$ we denote $\max\{\ell \geq 0 : X[1\,..\,\ell] \approx Y[1\,..\,\ell]\}$. As in the case of standard suffix trees, by equivalence (1), having an $\approx$-suffix tree of T and the data structure answering lowest common ancestor queries for nodes in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ preprocessing [25, 5], we can answer $\mathsf{LCP}^{\approx}$ queries about pairs of substrings of T in $\mathcal{O}(1)$ time.

The longest previous $\approx$-factor array $\mathsf{LPF}^{\approx}$ for a string T is an array such that

$$\mathsf{LPF}^{\approx}[i] = \max\{\ell \geq 0 : T[i\,..\,i+\ell) \approx T[j\,..\,j+\ell) \text{ for some } j \in [1\,..\,i)\}.$$

Computing this array can be stated in terms of the $\approx$-suffix tree: for a leaf with index i of the tree we are to find a leaf with index $j < i$ such that the lowest common ancestor of the two leaves is as low as possible. The actual value $\mathsf{LPF}^{\approx}[i]$ is then the (weighted) depth of this ancestor. In [13, Theorem 16] it was shown that this problem, stated for an arbitrary rooted (weighted) tree with n leaves, can be solved in $\mathcal{O}(n)$ time. Thus, the $\mathsf{LPF}^{\approx}$ array for a length-n string can be computed in $\mathcal{O}(n)$ time if the $\approx$-suffix tree is available.

4 Counting p-Squares in $\mathcal{O}(n\sigma \log \sigma)$ Time

In this section we consider $T \in [0\,..\,\sigma)^n$ and $\approx$ denotes the relation of parameterized matching. We use the following function $\mathbf{E}$ for $\approx$ (see Example 17):

$$\mathbf{E}(X) = |\mathsf{Alph}(U)| \text{ where } U \text{ is the longest suffix of } X[1\,..\,|X|) \text{ without letter } X[|X|].$$

▶ **Example 17.** $\mathbf{E}(X)$ for $X = [1,2,1,1,2,3,2,1,4]$ is as follows:

i	1	2	3	4	5	6	7	8	9
$X[i]$	1	2	1	1	2	3	2	1	4
$\mathbf{E}(X[1\,..\,i])$	0	1	1	0	1	2	1	2	3

A similar function was used in the context of parameterized matching in [30]. Lemma 18 below shows that $\mathbf{E}$ is an encoding function for parameterized matching and Lemma 19 shows that it can be computed efficiently.

The ($\Leftarrow$) part of the proof of Lemma 18 is similar to the proof of [30, Lemma 5.4]. A proof of the lemma can be found in the full version.

▶ **Lemma 18.** *$\mathbf{E}$ is an encoding function for parameterized matching.*

▶ **Lemma 19.** *We have $\pi^{\mathbf{E}}(n,\sigma) = \mathcal{O}(n\sigma)$, $\rho^{\mathbf{E}}(n,\sigma) = \mathcal{O}(\sigma)$, and $\gamma^{\mathbf{E}}(n,\sigma) = \sigma + 1$.*

Proof. Let $T \in [0\,..\,\sigma)^n$. By definition, $\mathbf{E}(T[i\,..\,j]) \in [0\,..\,\sigma)$ for any indices i, j. Hence, $\gamma^{\mathbf{E}}(n,\sigma) = \sigma + 1$ (including the sentinel). In order to compute encodings of substrings of T, we can store for every $c \in [0\,..\,\sigma)$, the numbers of characters c in respective prefixes of T. Moreover, for every index i, we store the position $prev[i] \in [0\,..\,i)$ of the last occurrence of character $T[i]$ in $T[1\,..\,i)$; $prev[i] = 0$ if there is no such occurrence. The array $prev$ can be computed from left to right in $\mathcal{O}(n+\sigma)$ time by storing an array of size σ of rightmost occurrences of each character. Then indeed $\pi^{\mathbf{E}}(n,\sigma) = \mathcal{O}(n\sigma)$. When computing $\mathbf{E}(T[i\,..\,j])$, we can compare the counts of every character c at prefixes $T[1\,..\,j)$ and $T[1\,..\,i')$ where $i' = \max(prev[j]+1, i)$. Hence, $\rho^{\mathbf{E}}(n,\sigma) = \mathcal{O}(\sigma)$. ◀

We define strings $\overrightarrow{T}$, $\overleftarrow{T}$ of length n such that

$$\overrightarrow{T}[i] = \mathbf{E}(T[1\,..\,i]), \quad \overleftarrow{T}[i] = \mathbf{E}((T[i\,..\,n])^R).$$

Our goal is to reduce reporting non-equivalent p-squares to computing uniform k-runs. We use two auxiliary lemmas.

▶ **Lemma 20.** *If $T[i\,..\,i+2\ell)$ is a p-square, then $\overrightarrow{T}[i\,..\,i+2\ell)$ and $\overleftarrow{T}[i\,..\,i+2\ell)$ are σ-mismatch squares.*

Proof. We show a proof that $\overrightarrow{T}[i\,..\,i+2\ell)$ is a σ-mismatch square; the proof that $\overleftarrow{T}[i\,..\,i+2\ell)$ is a σ-mismatch square is symmetric.

Let $i_1, \ldots, i_t \in [0\,..\,\ell)$, for $t \in [1\,..\,\sigma]$, be the positions of leftmost occurrences of characters from $[0\,..\,\sigma)$ in $T[i\,..\,i+\ell)$; if some character is not present in $T[i\,..\,i+\ell)$, it is not included in the sequence. Let $j \in [0\,..\,\ell) \setminus \{i_1, \ldots, i_t\}$. Let $j' \in [0\,..\,j)$ be the maximum index such that $T[i+j'] = T[i+j]$, so $T[i+\ell+j'] = T[i+\ell+j]$ and $T[i+\ell+j''] \neq T[i+\ell+j]$ for all $j'' \in [j'+1\,..\,j)$ because $T[i\,..\,i+\ell) \approx T[i+\ell\,..\,i+2\ell)$. Then

$$\overrightarrow{T}[i+j] \;=\; |\mathsf{Alph}(T[i+j'+1\,..\,i+j))| \;=\; |\mathsf{Alph}(T[i+\ell+j'+1\,..\,i+\ell+j))| \;=\; \overrightarrow{T}[i+\ell+j]$$

by the fact that $\approx$ is an SCER. Hence, $\overrightarrow{T}[i\,..\,i+\ell)$ and $\overrightarrow{T}[i+\ell\,..\,i+2\ell)$ have at most $t \le \sigma$ mismatches, as required. ◀

▶ **Lemma 21.** *If $T[i\,..\,i+2\ell)$ is a p-square and $\overrightarrow{T}[i+\ell] = \overrightarrow{T}[i+2\ell]$, then $T[i+1\,..\,i+2\ell]$ is a p-square. Similarly, if $T[i\,..\,i+2\ell)$ is a p-square and $\overleftarrow{T}[i-1] = \overleftarrow{T}[i+\ell-1]$, then $T[i-1\,..\,i+2\ell-1)$ is a p-square.*

Proof. Again, we only prove the first part of the lemma. We have $T[i\,..\,i+\ell) \approx T[i+\ell\,..\,i+2\ell)$, so $T[i+1\,..\,i+\ell) \approx T[i+1+\ell\,..\,i+2\ell)$ since $\approx$ is an SCER. If

$$\overrightarrow{T}[i+2\ell] = \overrightarrow{T}[i+\ell] < |\mathsf{Alph}(T[i+1\,..\,i+\ell))| = |\mathsf{Alph}(T[i+1+\ell\,..\,i+2\ell))|,$$

then $\overrightarrow{T}[i+\ell] = \mathbf{E}(T[i+1\,..\,i+\ell])$, $\overrightarrow{T}[i+2\ell] = \mathbf{E}(T[i+1+\ell\,..\,i+2\ell])$, and $T[i+1\,..\,i+\ell] \approx T[i+1+\ell\,..\,i+2\ell]$ by the fact that $\mathbf{E}$ is an encoding function for parameterized matching (Lemma 18). Otherwise,

$$T[i+\ell] \notin \mathsf{Alph}(T[i+1\,..\,i+\ell)) \quad \text{and} \quad T[i+2\ell] \notin \mathsf{Alph}(T[i+1+\ell\,..\,i+2\ell)),$$

so we immediately obtain $T[i+1\,..\,i+\ell] \approx T[i+1+\ell\,..\,i+2\ell]$. In both cases, $T[i+1\,..\,i+2\ell]$ is a p-square. ◀

If $T[a\,..\,b)$ is a uniform k-run with period ℓ, then there is no ℓ-mismatching position in $T[a+\ell\,..\,b-\ell)$, i.e., $T[a+\ell\,..\,b-\ell) = T[a+2\ell\,..\,b)$. Hence, if $\overrightarrow{T}[a\,..\,b)$ is a uniform k-run with period ℓ and $T[a\,..\,a+2\ell)$ is a p-square, then by Lemma 21, each string $T[i\,..\,i+2\ell)$ for $i \in [a\,..\,b-2\ell]$ is a p-square. With this intuition, we are ready to obtain the reduction.

▶ **Lemma 22.** *Let $T \in [0\,..\,\sigma)^n$. Reporting non-equivalent p-squares in T reduces in $\mathcal{O}(n\sigma+r)$ time to computing uniform σ-runs in $\overrightarrow{T}$ and $\overleftarrow{T}$, where r is the number of these σ-runs.*

Reporting p-squares in T that are distinct as strings reduces to the same problem in $\mathcal{O}(n\sigma + r + \mathsf{output})$ time, where output is the number of p-squares reported.

Proof. For a uniform k-run $T[a\,..\,b)$ of period ℓ, we define its *interval* as $[a\,..\,b-2\ell]$. For each $\ell \in [1\,..\,\lfloor n/2 \rfloor]$, let $\overrightarrow{\mathcal{P}}_\ell$ and $\overleftarrow{\mathcal{P}}_\ell$ be sets of intervals of uniform σ-runs of period ℓ in $\overrightarrow{T}$ and $(\overleftarrow{T})^R$, respectively. Let $\overleftarrow{\mathcal{P}}'_\ell = \{[n-b-2\ell\,..\,n-a-2\ell] \,:\, [a\,..\,b] \in \overleftarrow{\mathcal{P}}_\ell\}$. Finally, let $\mathcal{P}_\ell = \left(\bigcup \overrightarrow{\mathcal{P}}_\ell\right) \cap \left(\bigcup \overleftarrow{\mathcal{P}}'_\ell\right)$.

▷ Claim 23. If $T[i\,..\,i+2\ell)$ is a p-square in T, then $i \in \mathcal{P}_\ell$.

Proof. By Lemma 20, if $T[i\,..\,i+2\ell)$ is a p-square in T, then $\overrightarrow{T}[i\,..\,i+2\ell)$ and $\overleftarrow{T}[i\,..\,i+2\ell) = (\overleftarrow{T})^R[n-i-2\ell\,..\,n-i)$ are σ-mismatch squares. Therefore, there exist intervals $[a\,..\,b] \in \overrightarrow{\mathcal{P}}_\ell$ and $[a'\,..\,b'] \in \overleftarrow{\mathcal{P}}_\ell$ such that $i \in [a\,..\,b]$ and $n-i-2\ell \in [a'\,..\,b']$. In particular, $i \in \bigcup \overrightarrow{\mathcal{P}}_\ell$. We have $[n-b'-2\ell\,..\,n-a'-2\ell] \in \overleftarrow{\mathcal{P}}'_\ell$ and $i \in [n-b'-2\ell\,..\,n-a'-2\ell]$, so $i \in \bigcup \overleftarrow{\mathcal{P}}'_\ell$. ◁

We store $\mathcal{P}_\ell$ as a union of non-empty intervals of the form $\{I \cap J \,:\, I \in \overrightarrow{\mathcal{P}}_\ell,\, J \in \overleftarrow{\mathcal{P}}'_\ell\}$. Let this representation be denoted as $\mathcal{R}_\ell$.

All the representations $\mathcal{R}_\ell$ can be computed from $\overrightarrow{\mathcal{P}}_\ell$ and $\overleftarrow{\mathcal{P}}'_\ell$ in $\mathcal{O}(n + \sum_\ell(|\overrightarrow{\mathcal{P}}_\ell| + |\overleftarrow{\mathcal{P}}_\ell|))$ total time. Let us bucket sort all endpoints of intervals in $\overrightarrow{\mathcal{P}}_\ell$ and $\overleftarrow{\mathcal{P}}'_\ell$, for each ℓ. When processing the endpoints for a given ℓ, we keep track of the number of intervals containing a given position. This counter never exceeds 2 as intervals in each of $\overrightarrow{\mathcal{P}}_\ell$ and $\overleftarrow{\mathcal{P}}'_\ell$ are pairwise disjoint. Whenever the counter reaches 2, for some endpoint a, it will drop at the next endpoint encountered, say b. Then $[a\,..\,b)$ is inserted into $\mathcal{R}_\ell$.

The next claim shows that, for each interval in $\mathcal{R}_\ell$, all positions correspond to p-squares of length 2ℓ or none of them does.

▷ Claim 24. Let $[a\,..\,b] \in \mathcal{R}_\ell$. For each $i \in [a\,..\,b]$, $T[i\,..\,i+2\ell)$ is a p-square if and only if $T[a\,..\,a+2\ell)$ is a p-square.

Proof. Let us fix $i \in [a\,..\,b]$. Let $[a'\,..\,b'] \in \overrightarrow{\mathcal{P}}_\ell$ and $[a''\,..\,b''] \in \overleftarrow{\mathcal{P}}'_\ell$ be intervals such that $[a\,..\,b] = [a'\,..\,b'] \cap [a''\,..\,b'']$.

Assume first that $T[a\,..\,a+2\ell)$ is a p-square. As $\overrightarrow{\mathcal{P}}_\ell$ was computed from uniform σ-runs, we have $\overrightarrow{T}[a'+2\ell\,..\,b'+2\ell) = \overrightarrow{T}[a'+\ell\,..\,b'+\ell)$, so $\overrightarrow{T}[a+2\ell\,..\,i+2\ell) = \overrightarrow{T}[a+\ell\,..\,i+\ell)$. By Lemma 21 applied for each subsequent position in $[a\,..\,i)$, $T[i\,..\,i+2\ell)$ is a p-square.

Now assume that $T[i\,..\,i+2\ell)$ is a p-square. As $[a''\,..\,b''] \in \overleftarrow{\mathcal{P}}'_\ell$, $[n-b''-2\ell\,..\,n-a''-2\ell] \in \overleftarrow{\mathcal{P}}_\ell$. By definition, we have $(\overleftarrow{T})^R[n-b''\,..\,n-a'') = (\overleftarrow{T})^R[n-b''-\ell\,..\,n-a''-\ell)$, i.e., $\overleftarrow{T}[a''\,..\,b'') = \overleftarrow{T}[a''+\ell\,..\,b''+\ell)$. In particular, $\overleftarrow{T}[a\,..\,i) = \overleftarrow{T}[a+\ell\,..\,i+\ell)$. By Lemma 21 applied for each position in $[a+1\,..\,i]$ in decreasing order, $T[a\,..\,a+2\ell)$ is a p-square. ◁

By Corollary 15 and Lemmas 18 and 19, after $\mathcal{O}(n\sigma)$ preprocessing we can check if a given even-length substring of T is a p-square in $\mathcal{O}(1)$ time using an $\mathsf{LCP}^{\approx}$ query. For each $\ell \in [1\,..\,\lfloor n/2 \rfloor]$ and each interval $[a\,..\,b] \in \mathcal{R}_\ell$, we check if $T[a\,..\,a+2\ell)$ is a p-square. By Claim 24, if so, we obtain an interval of occurrences of p-squares, and if not, then none of the positions in $[a\,..\,b]$ is the start of a p-square of length 2ℓ in T. The total number of intervals in $\mathcal{R}_\ell$ is linear in the number of uniform σ-runs in $\overrightarrow{T}$ and $\overleftarrow{T}$, so we obtain $\mathcal{O}(r)$ intervals of occurrences of p-squares. By Claim 23, we do not miss any p-squares.

The final step of reporting non-equivalent p-squares mimics an analogous algorithm for standard squares of Bannai, Inenaga, and Köppl [4]. We report non-equivalent p-squares at their leftmost occurrence in T and use the $\mathsf{LPF}^{\approx}$ array to identify them in intervals of

occurrences. More precisely, we compute the $\mathsf{LPF}^{\approx}$ array in $\mathcal{O}(n\sigma)$ time and construct in $\mathcal{O}(n)$ time a data structure for answering Range Minimum Queries (RmQs) over $\mathsf{LPF}^{\approx}$ in $\mathcal{O}(1)$ time per query; see [5, 25]. For each of the $\mathcal{O}(r)$ intervals $[a\,..\,b]$ of starting positions of p-squares of length 2ℓ, we want to list all positions $i \in [a\,..\,b]$ such that $\mathsf{LPF}^{\approx}[i] < 2\ell$ and report corresponding leftmost occurrences of p-squares $T[i\,..\,i+2\ell)$. We ask an RmQ on $\mathsf{LPF}^{\approx}$ on the interval $[a\,..\,b]$; let the minimum be attained for $i \in [a\,..\,b]$. If $\mathsf{LPF}^{\approx}[i] \geq 2\ell$, the algorithm is finished. Otherwise, we report $T[i\,..\,i+2\ell)$ and run the algorithm recursively on intervals $[a\,..\,i-1]$ and $[i+1\,..\,b]$. The total time complexity is proportional to the number of reported p-squares plus 1. By [24], T contains $\mathcal{O}(n\sigma)$ non-equivalent p-squares. This completes an $\mathcal{O}(n\sigma + r)$-time reduction to computing uniform σ-runs.

When reporting all p-squares that are distinct as strings, it suffices to replace the $\mathsf{LPF}^{\approx}$ array with the standard $\mathsf{LPF} = \mathsf{LPF}^{=}$ array. Such an array can be computed in $\mathcal{O}(n)$ time after the letters of the string have been sorted [12]. Then the computations take $\mathcal{O}(n\sigma + r + \mathsf{output})$ time. ◀

By Theorem 12, string $\overrightarrow{T}$ (and $\overleftarrow{T}$) contains $\mathcal{O}(n\sigma \log \sigma)$ uniform σ-runs. They can be computed in $\mathcal{O}(n\sigma \log \sigma)$ time; we use the algorithm of Kolpakov and Kucherov [34, 35] to compute all σ-runs in $\overrightarrow{T}$ and then partition each σ-run to uniform σ-runs using kangaroo jumps. That is, let $\overrightarrow{T}[a\,..\,b)$ be a σ-run with period. We compute two values:

$$d_1 = 1 + \mathsf{LCP}(\overrightarrow{T}[a\,..\,b-2\ell), \overrightarrow{T}[a+\ell\,..\,b-\ell))$$
$$d_2 = 1 + \mathsf{LCP}(\overrightarrow{T}[a+\ell\,..\,b-\ell), \overrightarrow{T}[a+2\ell\,..\,b))$$

that, intuitively, find the first ℓ-mismatching position in the σ-run, if any (d_1), and the first ℓ-mismatching position after position $a+\ell-1$, if any (d_2). Let $d = \min(d_1, d_2)$. We then report a uniform σ-run $\overrightarrow{T}[a\,..\,a+d+2\ell)$ and continue processing the (non-maximal) σ-run $\overrightarrow{T}[a+d\,..\,b)$ until its length drops below 2ℓ. The LCP-queries in $(\overrightarrow{T})^R$ are answered in $\mathcal{O}(1)$ time [5, 25]. Thus Lemma 22 implies Theorem 2.

5 Counting Generalised Squares in $\mathcal{O}(n \log n)$ Time

In this section we show an algorithm that counts non-equivalent $\approx$-squares, for an SCER $\approx$ with encoding function $\mathcal{E}$, in $\mathcal{O}(\pi^{\mathcal{E}}(n) + n\rho^{\mathcal{E}}(n) + n \log n)$ time.

For a string T of length n and positive integer $p \leq n/2$, we denote

$$\mathsf{Squares}_p^{\approx} = \{i \in [1\,..\,n-2p+1] : T[i\,..\,i+2p) \text{ is an } \approx\text{-square}\}.$$

An *interval representation* of a set X of integers is $X = [i_1\,..\,j_1] \cup [i_2\,..\,j_2] \cup \cdots \cup [i_t\,..\,j_t]$, where $j_1 + 1 < i_2, \ldots, j_{t-1} + 1 < i_t$; t is called the *size* of the representation.

Counterparts of the following lemma corresponding to order-preserving squares and Cartesian-tree squares were shown in [21] and [44], respectively. Our proof generalises these proofs; it can be found in the full version.

▶ **Lemma 25.** *Let $\approx$ be an SCER and $\mathcal{E}$ be its encoding function. Given a string T of length n, the interval representations of the sets $\mathsf{Squares}_p^{\approx}$ for all $1 \leq p \leq n/2$ have total size $\mathcal{O}(n \log n)$ and can be computed in $\mathcal{O}(\pi^{\mathcal{E}}(n) + n\rho^{\mathcal{E}}(n) + n \log n)$ time.*

We count non-equivalent $\approx$-squares using the $\mathsf{LPF}^{\approx}$ array. We would like to count an $\approx$-square at the position of its leftmost occurrence. For an integer array $A[1\,..\,n]$, we denote a range count query for $i, j \in [1\,..\,n]$, $x \in \mathbb{Z}$ as $\mathsf{RangeCount}_A(i, j, x) = |\{k \in [i\,..\,j] : A[k] < x\}|$.

Then our problem reduces to computing the sum

$$\sum_{p=1}^{\lfloor n/2 \rfloor} \sum_{[i..j] \in \mathsf{Squares}_p} \mathsf{RangeCount}_{\mathsf{LPF}^{\approx}}(i, j, 2p). \tag{4}$$

We say that an array $A[1..n]$ is a *linear oscillation array* if $\sum_{i=1}^{n-1} |A[i+1] - A[i]| = \mathcal{O}(n)$. The following fact is folklore for the LPF array; we prove it for $\mathsf{LPF}^{\approx}$ in the full version.

▶ **Lemma 26.** *For any SCER $\approx$, $\mathsf{LPF}^{\approx}$ is a linear oscillation array.*

We will now show that the sum from Equation (4) can be computed in $\mathcal{O}(n \log n)$ time. We use a very simple abstract problem.

Counting Problem
Input: A set $Y \subseteq [1..n]$, initially empty, and an integer ℓ, initially $\ell = 0$.
Operation: One of: (1) insert an element $x \in [1..n]$ to Z; (2) delete an element $x \in Y$ from Y; (3) increment ℓ by 1; (4) decrement ℓ by 1. After each operation, output $|Y \cap [\ell+1..n]|$.

The Counting Problem can be solved with a basic indicator array for Y.

▶ **Lemma 27.** *After $\mathcal{O}(n)$-time preprocessing, each operation in the Counting Problem can be answered in $\mathcal{O}(1)$ time.*

Proof. We store an indicator array of bits $C[1..n]$ such that $C[i] = 1$ if and only if $i \in Y$. Initially, $C \equiv 0$. We also store a value $a = |Y \cap [\ell + 1..n]|$; initially $a = 0$. After each operation, the current value of a is returned.

When we insert an element $x \in [1..n]$ to Y, we set $C[x]$ to 1 and increment a if $x > \ell$. Symmetrically, when we delete $x \in Y$ from Y, we set $C[x]$ to 0 and decrement a if $x < \ell$. When we increment ℓ, we subtract $C[\ell]$ from a. When we decrement ℓ, we add $C[\ell]$ to a. ◀

We are ready to obtain the final main result.

Proof of Theorem 5. First we show how to compute the number of non-equivalent $\approx$-squares in a length-n string. The $\mathsf{LPF}^{\approx}$ array can be computed in $\mathcal{O}(\pi^{\mathcal{E}}(n) + n\rho^{\mathcal{E}}(n) + n \log^2 \log n / \log \log \log n)$ time (Corollary 15). We compute the interval representations of the sets $\mathsf{Squares}_p^{\approx}$ using Lemma 25 in $\mathcal{O}(\pi^{\mathcal{E}}(n) + n\rho^{\mathcal{E}}(n) + n \log n)$ time. The interval representations have total size $\mathcal{O}(n \log n)$. We need to compute the sum of results of $\mathcal{O}(n \log n)$ $\mathsf{RangeCount}$ queries as in Equation (4). We will do it using the Counting Problem.

Let us bucket sort all start and end points of intervals across all sets $\mathsf{Squares}_p^{\approx}$ in $\mathcal{O}(n \log n)$ total time. We create an instance of the Counting Problem. For each position k from 1 to n, we proceed as follows. First, for each interval $[k..j] \in \mathsf{Squares}_p^{\approx}$, for any p, we insert $2p$ to the set Y. Then, we change the current value ℓ in the problem to $\mathsf{LPF}^{\approx}[k]$ by performing increments or decrements, as appropriate. Afterwards, we add the returned value of the Counting Problem to the final result. Finally, for each interval $[i..k] \in \mathsf{Squares}_p^{\approx}$, for any p, we delete $2p$ from the set Y.

For each $p \in [1..\lfloor n/2 \rfloor]$, the intervals in $\mathsf{Squares}_p$ are pairwise disjoint. By Lemma 25, $\mathcal{O}(n \log n)$ insertions and deletions are performed in the Counting Problem. The total number of increments and decrements is bounded by $\mathcal{O}(n)$ by Lemma 26 (and the fact that the values in this array are in $[0..n)$). In total, the sum (4) is computed in $\mathcal{O}(n \log n)$ time. We obtain the first part of Theorem 5.

As before, if one wishes to compute all $\approx$-squares that are distinct as substrings, it suffices to replace the $\mathsf{LPF}^{\approx}$ array in the algorithm by the standard $\mathsf{LPF} = \mathsf{LPF}^{=}$ array. We obtain the second part of Theorem 5. ◀

References

1. Arne Andersson and Mikkel Thorup. Dynamic ordered sets with exponential search trees. *Journal of the ACM*, 54(3):13, 2007. doi:10.1145/1236457.1236460.
2. Brenda S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52(1):28–42, 1996. doi:10.1006/JCSS.1996.0003.
3. Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "Runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
4. Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.CPM.2017.22.
5. Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.
6. Gary Benson. An algorithm for finding tandem repeats of unspecified pattern size. In Sorin Istrail, Pavel A. Pevzner, and Michael S. Waterman, editors, *Proceedings of the Second Annual International Conference on Research in Computational Molecular Biology, RECOMB 1998, New York, NY, USA, March 22-25, 1998*, pages 20–29. ACM, 1998. doi:10.1145/279069.279079.
7. Gary Benson. Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research*, 27(2):573–580, 1999. doi:10.1093/nar/27.2.573.
8. Srečko Brlek and Shuo Li. On the number of squares in a finite word. *Combinatorial Theory*, 5(1), 2025. doi:10.5070/C65165014.
9. Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Efficient enumeration of distinct factors using package representations. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval – 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 247–261. Springer, 2020. doi:10.1007/978-3-030-59212-7_18.
10. Panagiotis Charalampopoulos, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Counting distinct square substrings in sublinear time. In Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science, MFCS 2025, August 25–29, 2025, Warsaw, Poland*, LIPIcs, 2025. doi:10.4230/LIPICS.MFCS.2025.36.
11. Richard Cole and Ramesh Hariharan. Faster suffix tree construction with missing suffix links. *SIAM Journal on Computing*, 33(1):26–42, 2003. doi:10.1137/S0097539701424465.
12. Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Information Processing Letters*, 106(2):75–80, 2008. doi:10.1016/J.IPL.2007.10.006.
13. Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Order-preserving indexing. *Theoretical Computer Science*, 638:122–135, 2016. doi:10.1016/J.TCS.2015.06.050.
14. Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014. doi:10.1016/J.TCS.2013.11.018.
15. Olivier Delgrange and Eric Rivals. STAR: an algorithm to search for tandem approximate repeats. *Bioinformatics*, 20:2812–2820, 2004. doi:10.1093/bioinformatics/bth335.

16 Nevzat Onur Domaniç and Franco P. Preparata. A novel approach to the detection of genomic approximate tandem repeats in the Levenshtein metric. *Journal of Computational Biology*, 14(7):873–891, 2007. PMID: 17803368. doi:10.1089/cmb.2007.0018.

17 Jonas Ellert and Johannes Fischer. Linear time runs over general ordered alphabets. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 63:1–63:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.63.

18 Younan Gao, Meng He, and Yakov Nekrich. Fast preprocessing for optimal orthogonal range reporting and range successor with applications to text indexing. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 54:1–54:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.54.

19 Pawel Gawrychowski, Samah Ghazawi, and Gad M. Landau. Order-preserving squares in strings. In Laurent Bulteau and Zsuzsanna Lipták, editors, *34th Annual Symposium on Combinatorial Pattern Matching, CPM 2023, June 26-28, 2023, Marne-la-Vallée, France*, volume 259 of *LIPIcs*, pages 13:1–13:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CPM.2023.13.

20 Pawel Gawrychowski, Tomohiro I, Shunsuke Inenaga, Dominik Köppl, and Florin Manea. Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes – finding all maximal α-gapped repeats and palindromes in optimal worst case time on integer alphabets. *Theory of Computing Systems*, 62(1):162–191, 2018. doi:10.1007/S00224-017-9794-5.

21 Garance Gourdel, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Arseny M. Shur, and Tomasz Waleń. String periods in the order-preserving model. *Information and Computation*, 270, 2020. doi:10.1016/J.IC.2019.104463.

22 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

23 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *Journal of Computer and System Sciences*, 69(4):525–546, 2004. doi:10.1016/J.JCSS.2004.03.004.

24 Rikuya Hamai, Kazushi Taketsugu, Yuto Nakashima, Shunsuke Inenaga, and Hideo Bannai. On the number of non-equivalent parameterized squares in a string. In Zsuzsanna Lipták, Edleno Silva de Moura, Karina Figueroa, and Ricardo Baeza-Yates, editors, *String Processing and Information Retrieval – 31st International Symposium, SPIRE 2024, Puerto Vallarta, Mexico, September 23-25, 2024, Proceedings*, volume 14899 of *Lecture Notes in Computer Science*, pages 174–183. Springer, 2024. doi:10.1007/978-3-031-72200-4_13.

25 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. doi:10.1137/0213024.

26 Tomohiro I, Shunsuke Inenaga, and Masayuki Takeda. Palindrome pattern matching. *Theoretical Computer Science*, 483:162–170, 2013. doi:10.1016/J.TCS.2012.01.047.

27 Tomohiro I and Dominik Köppl. Improved upper bounds on all maximal α-gapped repeats and palindromes. *Theoretical Computer Science*, 753:1–15, 2019. doi:10.1016/J.TCS.2018.06.033.

28 Haim Kaplan, Ely Porat, and Nira Shafrir. Finding the position of the k-mismatch and approximate tandem repeats. In Lars Arge and Rusins Freivalds, editors, *Algorithm Theory – SWAT 2006, 10th ScandinavianWorkshop on Algorithm Theory, Riga, Latvia, July 6-8, 2006, Proceedings*, volume 4059 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2006. doi:10.1007/11785293_11.

29 Jinil Kim, Peter Eades, Rudolf Fleischer, Seok-Hee Hong, Costas S. Iliopoulos, Kunsoo Park, Simon J. Puglisi, and Takeshi Tokuyama. Order-preserving matching. *Theoretical Computer Science*, 525:68–79, 2014. doi:10.1016/J.TCS.2013.10.006.

30 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Maximum number of distinct and nonequivalent nonstandard squares in a word. *Theoretical Computer Science*, 648:84–95, 2016. doi:10.1016/J.TCS.2016.08.010.

31 Roman Kolpakov, Ghizlane Bana, and Gregory Kucherov. mreps: Efficient and flexible detection of tandem repeats in DNA. *Nucleic Acids Research*, 31:3672–3678, 2003. doi:10.1093/nar/gkg617.

32 Roman Kolpakov, Mikhail Podolskiy, Mikhail Posypkin, and Nickolay Khrapov. Searching of gapped repeats and subrepetitions in a word. *Journal of Discrete Algorithms*, 46-47:1–15, 2017. doi:10.1016/J.JDA.2017.10.004.

33 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

34 Roman M. Kolpakov and Gregory Kucherov. Finding approximate repetitions under Hamming distance. In Friedhelm Meyer auf der Heide, editor, *Algorithms – ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2001. doi:10.1007/3-540-44676-1_14.

35 Roman M. Kolpakov and Gregory Kucherov. Finding approximate repetitions under Hamming distance. *Theoretical Computer Science*, 303(1):135–156, 2003. doi:10.1016/S0304-3975(02)00448-6.

36 Arun Krishnan and Francis Tang. Exhaustive whole-genome tandem repeats search. *Bioinformatics*, 20(16):2702–2710, May 2004. doi:10.1093/bioinformatics/bth311.

37 Marcin Kubica, Tomasz Kulczyński, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear time algorithm for consecutive permutation pattern matching. *Information Processing Letters*, 113(12):430–433, 2013. doi:10.1016/J.IPL.2013.03.015.

38 Gregory Kucherov and Dina Sokol. Approximate tandem repeats. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1–6. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. doi:10.1007/978-3-642-27848-8_24-2.

39 Gad M. Landau, Jeanette P. Schmidt, and Dina Sokol. An algorithm for approximate tandem repeats. *Journal of Computational Biology*, 8(1):1–18, 2001. doi:10.1089/106652701300099038.

40 Michael G. Main and Richard J. Lorentz. An O(n log n) algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984. doi:10.1016/0196-6774(84)90021-X.

41 Glenn K. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *Journal of the ACM*, 22(3):346–351, 1975. doi:10.1145/321892.321896.

42 Yoshiaki Matsuoka, Takahiro Aoki, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Generalized pattern matching and periodicity under substring consistent equivalence relations. *Theoretical Computer Science*, 656:225–233, 2016. doi:10.1016/J.TCS.2016.02.017.

43 Angelika Merkel and Neal Gemmell. Detecting short tandem repeats from genome data: opening the software black box. *Briefings in Bioinformatics*, 9:355–366, 2008. doi:10.1093/bib/bbn028.

44 Sung Gwan Park, Magsarjav Bataa, Amihood Amir, Gad M. Landau, and Kunsoo Park. Finding patterns and periods in Cartesian tree matching. *Theoretical Computer Science*, 845:181–197, 2020. doi:10.1016/J.TCS.2020.09.014.

45 Marco Pellegrini, M. Elena Renda, and Alessio Vecchio. TRStalker: an efficient heuristic for finding fuzzy tandem repeats. *Bioinformatics*, 26(12):i358–i366, June 2010. doi:10.1093/bioinformatics/btq209.

46 Jeff Reneker, Chi-Ren Shyu, Peiyu Zeng, Joseph C. Polacco, and Walter Gassmann. ACMES: fast multiple-genome searches for short repeat sequences with concurrent cross-species information retrieval. *Nucleic Acids Research*, 32:W649–W653, 2004. doi:10.1093/nar/gkh455.

47 E. Rivals, J. Delahaye, M. Dauchet, and O. Delgrange. A first step toward chromosome analysis by compression algorithms. In *Proceedings First International Symposium on Intelligence in Neural and Biological Systems. INBS 1995*, pages 233–239, 1995. doi:10.1109/INBS.1995.404256.

48 Eric Rivals, Olivier Delgrange, Jean-Paul Delahaye, Max Dauchet, Marie-Odile Delorme, Alain Hénaut, and Emmanuelle Ollivier. Detection of significant patterns by compression algorithms: the case of approximate tandem repeats in DNA sequences. *Computer Applications in the Biosciences*, 13(2):131–136, 1997. doi:10.1093/BIOINFORMATICS/13.2.131.

49 Marie-France Sagot and Eugene W. Myers. Identifying satellites and periodic repetitions in biological sequences. *Journal of Computational Biology*, 5(3):539–553, 1998. doi:10.1089/CMB.1998.5.539.

50 Tiago José P. Sobreira, Alan M. Durham, and Arthur Gruber. TRAP: automated classification, quantification and annotation of tandemly repeated sequences. *Bioinformatics*, 22(3):361–362, 2006. doi:10.1093/bioinformatics/bti809.

51 Dina Sokol, Gary Benson, and Justin Tojeira. Tandem repeats over the edit distance. *Bioinformatics*, 23(2):e30–e35, January 2007. doi:10.1093/bioinformatics/btl309.

52 Dina Sokol and Justin Tojeira. Speeding up the detection of tandem repeats over the edit distance. *Theoretical Computer Science*, 525:103–110, 2014. Advances in Stringology. doi:10.1016/j.tcs.2013.04.021.

53 Axel Thue. Über unendliche Zeichenreihen. *Norske Videnskabers Selskabs Skrifter Mat.-Nat. Kl.*, 7:1–22, 1906.

54 Jean Vuillemin. A unifying look at data structures. *Communications of the ACM*, 23(4):229–239, April 1980. doi:10.1145/358841.358852.

55 Ydo Wexler, Zohar Yakhini, Yechezkel Kashi, and Dan Geiger. Finding approximate tandem repeats in genomic sequences. In Philip E. Bourne and Dan Gusfield, editors, *Proceedings of the Eighth Annual International Conference on Computational Molecular Biology, 2004, San Diego, California, USA, March 27-31, 2004*, pages 223–232. ACM, 2004. doi:10.1145/974614.974644.

56 Hongxia Zhou, Liping Du, and Hong Yan. Detection of tandem repeats in dna sequences based on parametric spectral estimation. *IEEE Transactions on Information Technology in Biomedicine*, 13(5):747–755, 2009. doi:10.1109/TITB.2008.920626.

MorphisHash: Improving Space Efficiency of ShockHash for Minimal Perfect Hashing

Stefan Hermann
Karlsruhe Institute of Technology, Germany

Abstract

A minimal perfect hash function (MPHF) maps a set of n keys to unique positions $\{1, \ldots, n\}$. Representing an MPHF requires at least $\log_2(e) \approx 1.443$ bits per key. ShockHash is a technique to construct an MPHF and requires just slightly more space. It gives each key two random candidate positions. If each key can be mapped to one of its two candidate positions such that there is exactly one key mapped to each position, then an MPHF is found. If not, ShockHash repeats the process with a new set of random candidate positions. ShockHash has to store how many repetitions were required and for each key to which of the two candidate positions it is mapped. However, when a given set of candidate positions can be used as MPHF then there is not only one but multiple ways of mapping the keys to one of their candidate positions such that the mapping results in an MPHF. This redundancy makes up for the majority of the remaining space overhead in ShockHash. In this paper, we present MorphisHash which almost completely eliminates this redundancy. Our theoretical result is that MorphisHash saves $\Theta(\ln(n))$ bits in expectation compared to ShockHash. This corresponds to a factor of 20 less space overhead in practice. Just like ShockHash, MorphisHash can be used as a building block within RecSplit to obtain MorphisHash-RS. When compared for same space consumption, MorphisHash-RS can be constructed up to 21 times faster than ShockHash-RS. The technique to accomplish this might be of a more general interest to compress data structures.

2012 ACM Subject Classification Theory of computation → Data compression; Information systems → Point lookups; Theory of computation → Bloom filters and hashing; Mathematics of computing → Random graphs

Keywords and phrases compressed data structure, perfect hashing, random graph, pseudoforest, component

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.9

Related Version *Extended Version*: `https://arxiv.org/abs/2503.10161` [11]

Supplementary Material *Software*: `https://github.com/stefanfred/MorphisHash` [10]
archived at `swh:1:dir:72bf7952109795567ca30d31efc4c557b44dfc17`

Funding This work was supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF).

Acknowledgements I thank Stefan Walzer for proofreading an earlier version of this paper.

1 Introduction

Given a set S of n keys, a minimal perfect hash function (MPHF) maps each key to a unique position in $[n] := \{1, \ldots, n\}$. MPHFs have a wide range of applications including compressed full-text indexes [2], computer networks [18], databases [6], prefix-search data structures [1], language models [24], bioinformatics [7, 23], and Bloom filters [5]. Different techniques exist for constructing an MPHF. They offer a variety of trade-offs between construction time, space consumption and query time. The space lower bound of an MPHF is $\log_2(e) \approx 1.443$ bits per key [19].


© Stefan Hermann;
licensed under Creative Commons License CC-BY 4.0

ShockHash. Our technique builds on ShockHash [15, 16]. Similar to Cuckoo hashing [22], each key is given two candidate positions using respective hash functions $h_{s,0} : S \to [n]$ and $h_{s,1} : S \to [n]$ with seed s. ShockHash finds a seed s such that all keys can be mapped to one of their candidate positions and there is exactly one key mapped to each position. ShockHash needs to store the seed s, once found. Additionally, it needs to store for each key $k \in S$ if the candidate position $h_{s,0}(k)$ or $h_{s,1}(k)$ is used. This can be represented using a function $f : S \to \{0, 1\}$. Such a mapping can be stored efficiently using a retrieval data structure [8] which requires about 1 bit per key. A key $k \in S$ is queried using $h_{s,f(k)}(k)$.

A different perspective on ShockHash is that with each seed s it samples a random graph. The $[n]$ possible output positions are the nodes of that graph. The keys are the edges, connecting the nodes of their respective candidate positions. A seed is accepted if the graph can be oriented, i.e. each edge is given a direction, such that the indegree of each node is 1. This is possible if and only if the graph is a *pseudoforest* – a graph where each component is a cycle with trees branching from it. The edges of the cycle of each component are oriented either all "clockwise" or "counterclockwise". The edges in the trees are uniquely oriented away from their cycle. Hence, the indegree of each node is 1. ShockHash arbitrarily chooses one of two possible orientations of each cycle and stores the according orientation of each edge in a retrieval structure. Hence, for each cycle there is one bit of redundancy.

Contribution. In this paper we introduce MorphisHash which exhausts this remaining redundancy. MorphisHash is a recursive acronym: **M**orphisHash is an **o**verloaded **r**etrieval structure for **p**erfect **h**ash**i**ng using **S**hock**Hash**. Our key observation is that the possible orientations of a pseudoforest can be described as the solution of a linear equation system. A retrieval structure that stores the edge orientations can also be described as the solution of a linear equation system. This allows us to concatenate the equation systems using matrix multiplication. We achieve compression by reducing the dimensionality of the solution space of the combined equation system. Our theoretical insight is that a random pseudoforest has $\Theta(\ln(n))$ components in expectation and MorphisHash can convert this into $\Theta(\ln(n))$ bits of expected space savings compared to ShockHash by utilizing the freedom of choosing the orientation of each component's cycle. Our experiments show that MorphisHash has about a factor of 20 less space overhead than ShockHash at the cost of roughly 4 times more construction time.

Partitioning. Note that within this paper, n denotes the input size for one instance of MorphisHash. In Section 5, an MPHF is obtained by splitting a large input key set into a linear number of MorphisHash instances, each of size n, and concatenating them afterwards. Partitioning is required mainly for keeping construction times feasible. Furthermore, our per instance space savings translate to linear space savings in terms of the large input.

Outline. We begin in Section 2 with related space efficient PHF construction techniques. We present MorphisHash in Section 3 and analyze it in Section 4. We explain implementation details in Section 5. Finally, Section 6 discusses experiments and the paper is concluded with Section 7. Our compression technique might be of a more general interest and we give further examples in the full version of this paper [11].

2 Related Work

We provide a brief overview of other space efficient PHF techniques. For a detailed survey of state-of-the-art techniques we refer to [14].

RecSplit. RecSplit [9, 3] first hashes the keys into partitions of about 2000 keys. RecSplit then finds a seed of a hash function that splits the partition into smaller subsets of equal size. This is applied recursively resulting in a tree-like structure. Once sufficiently small subsets are obtained, RecSplit uses brute-force search to find an MPHF within each leaf. Very recently a significant improvement to RecSplit has been made with the introduction of CONSENSUS [17]. Instead of allowing arbitrarily large seeds, CONSENSUS-RecSplit uses a fixed number of bits for each seed and backtracks in the splitting tree if a seed cannot be found within the allowed space. CONSENSUS-RecSplit is currently the most space efficient technique with just 0.001 bits per key overhead.

PHOBIC. Another PHF construction technique is PHOBIC [12]. Again, the keys are hashed into partitions of about 2000 keys. Within each partition the keys are hashed to buckets which have an average size of about 10. For each bucket, PHOBIC uses brute force search to find a seed of a hash function such that all keys of that bucket are hashed to positions in $[n]$ to which no keys of previous buckets are hashed to. The buckets are inserted in non-increasing order of size because it is much easier to insert the larger buckets when the output domain is almost empty. This effect is utilized further by intentionally making some of the buckets larger. PHOBIC has fast queries at the cost of more space overhead.

3 MorphisHash

ShockHash samples random graphs until stumbling on a pseudoforest. The only remaining degree of freedom when orienting the pseudoforest is that each component contains a cycle and there are two ways to orient each cycle. We address this remaining redundancy with MorphisHash.

The first ingredient of MorphisHash is the insight that all allowed orientations of a graph can be expressed as an affine subspace of $\mathbb{F}_2^S$, where S is the set of n keys. To show this, we define $y \in \mathbb{F}_2^S$ as the vector representing the orientation of each edge such that an edge $j \in S$ is oriented to node $h_{s,y_j}(j)$. We now consider a given pseudoforest and one possible orientation y. We can flip the orientation of a cycle by adding a vector $v \in \mathbb{F}_2^S$ to y where $v_j = 1$ if and only if j is part of that cycle. This can be done for each cycle independently. Clearly, the dimension of this subspace is equal to the number of components. Part of this section is to describe the linear equation system of which the solution space is our desired affine subspace.

The second ingredient is a 1-bit retrieval data structure. The retrieval structure works by storing a bit vector $x \in \mathbb{F}_2^b$, where the parameter $b \in \mathbb{N}_0$ is discussed in detail later. The orientation of an edge $j \in S$ is described using $h'_s(j)x$, where $h'_s : S \to \mathbb{F}_2^{1\times b}$ is a hash function and s is the ShockHash seed. Using linear equations that involve hash functions is a common technique for retrieval data structures [8].

The beauty of MorphisHash is that we can concatenate both linear equation systems using a simple matrix multiplication to find a retrieval structure which directly orients the edges correctly. We can then decrease the number of bits b that the retrieval structure is allowed to use which reduces the dimension of the solution space and therefore extracts the remaining redundancy.

We now show the equation system that describes the allowed edge orientations. As a first step we show that the constraints of an MPHF can be weakened in the following sense:

▶ **Lemma 1.** *A function $f : S \to [n]$ with $|S| = n$ is an MPHF (i.e. a bijection) if and only if for all $i \in [n]$ we have that $|\{f^{-1}(i)\}|$ is uneven.*

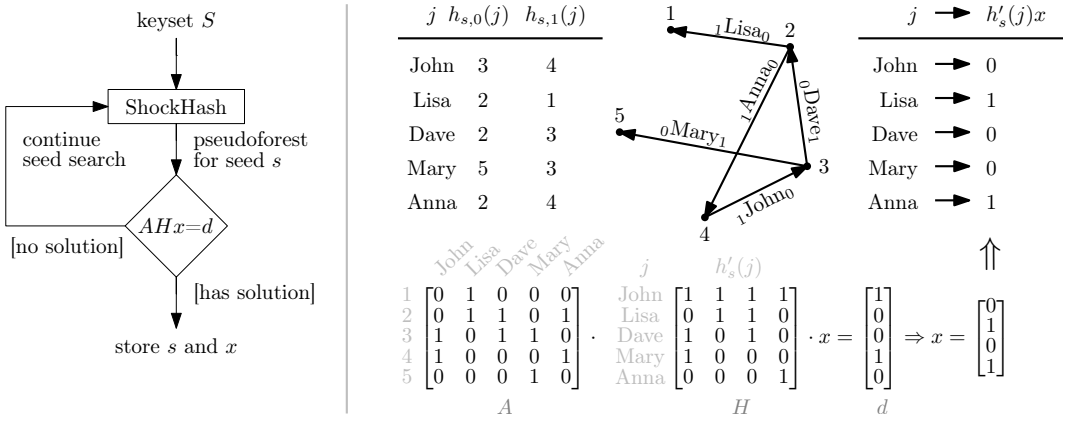


Figure 1 Left: MorphisHash uses ShockHash as a black box. Right: An example. ShockHash has found a pseudoforest using the candidate positions $h_{s,0}$ and $h_{s,1}$ as shown in the table and also illustrated as a graph. The description ${}_1\text{Lisa}_0$ indicates that $h_{s,1}$ is the left candidate and $h_{s,0}$ the right. This can be used to find d. For example, $d_3 = 0^{\text{Dave}} + 0^{\text{Mary}} + 1^{\text{John}} + 1^{\text{offset}} = 0$. The incidence matrix A and some random matrix H is shown. A solution x is found resulting in the orientations as shown both in the right table and using arrows in the graph. The solution x describes the orientation of 5 edges but only requires 4 bits to store. The figure is derived from [16, Fig. 1].

Proof. Clearly, if f is bijective then $|\{f^{-1}(i)\}| = 1$ is uneven. If f is not bijective then there is at least one i such that $|\{f^{-1}(i)\}| = 0$ which is even. ◀

Linear Equations in Graphs. This allows us to count the indegree of each node using $\mathbb{F}_2$: If the indegree of all nodes is $1_{\mathbb{F}_2}$ then the orientations result in a valid MPHF. Recall the definition of $y \in \mathbb{F}_2^S$ as the vector representing the orientation of each edge such that an edge $j \in S$ is oriented to node $h_{s,y_j}(j)$. A different perspective is that an edge j is oriented towards a node i if and only if $(h_{s,1}(j) = i \wedge y_j = 1) \vee (h_{s,0}(j) = i \wedge y_j + 1 = 1)$, which will be useful in the following equation. We define $A \in \mathbb{F}_2^{n \times S}$ as the incidence matrix of the graph. We also define $d \in \mathbb{F}_2^n$ where $d_i = |\{j \mid h_{s,0}(j) = i\}| + 1$ counts the number of edges $(+1)$ that are mapped to position i using the $h_{s,0}$ candidate function. Finally, this allows us to count the indegree of a node i as

$$\sum_{h_{s,1}(j)=i} y_j + \sum_{h_{s,0}(j)=i} (1 + y_j) = \Big(\sum_{\substack{h_{s,0}(j)=i \\ \oplus h_{s,1}(j)=i}} y_j \Big) + d_i + 1 = A_i y + d_i + 1$$

The $\oplus$ operator is logical XOR. According to Lemma 1 setting $A_i y + d_i + 1 = 1$ such that the indegree of each node i is uneven results in a valid orientation of all edges. The complete equation system therefore simplifies to $Ay = d$, which has a solution if and only if the graph is a pseudoforest. Note that in case of a loop, the respective column in the incidence matrix is a zero column.

The Retrieval Data Structure. To store the orientation of the edges we cannot use y directly because we do not know the index of a key during query time. We therefore employ the idea of a retrieval structure. Our retrieval structure consists of a bit vector $x \in \mathbb{F}_2^b$, where b is a tuning parameter. The orientation of edge j is then described using a scalar product $y_j = h'_s(j)x$ where $h'_s : S \to \mathbb{F}_2^{1 \times b}$ is a hash function and s is the ShockHash seed. MorphisHash needs to find x such that all edges are properly oriented. The above linear

equation is given for each key and can therefore be written using a matrix H. Each row H_j is the hash $h'_s(j)$ of the respective key j. We have $y = Hx$ and substitute it into $Ay = d$ resulting in $AHx = d$. If a solution x for $AHx = d$ exists then the graph is a pseudoforest, $y = Hx$ is a valid orientation and x requires exactly b bits to store using a bit string. We discuss the selection of b both in theory (Section 4) and practice (Section 6). Querying our MPHF for a key j is now straightforward $h_{s,h'_s(j)x}(j)$.

If the system $AHx = d$ does not have a solution for a sampled graph this can have two reasons. (1) The graph is no pseudoforest (2) A possible orientation of the pseudoforest is not within the solution space of the retrieval structure. If there is no solution we reject the seed and ShockHash continues with searching for a new seed.

The already existing step in ShockHash of checking whether a seed is a pseudoforest is therefore redundant. However, solving an equation system is computationally more expensive than using the original ShockHash pseudoforest check. We therefore leave the original pseudoforest check as a filter and only need to solve $AHx = d$ a few times in practice. MorphisHash is illustrated using an example in Figure 1.

Bipartite MorphisHash. A variant of ShockHash is bipartite ShockHash. Assume that n is even, the extension for uneven numbers can be found in the original paper [15]. In bipartite ShockHash, the ranges of the two hash functions are made disjoint using $h_{s_0,0} : S \to \{1, \ldots, \frac{n}{2}\}$ and $h_{s_1,1} : S \to \{\frac{n}{2} + 1, \ldots, n\}$. The seed is also split in two independent parts s_0 and s_1. Seeds where not all positions are hit by at least one candidate position are filtered out. Bipartite ShockHash only checks if pairs of s_0 and s_1 that passed the filter result in a pseudoforest. MorphisHash uses ShockHash as a black box and can be applied on the bipartite case just as well as on the non-bipartite case. In an obvious manner, we refer to bipartite MorphisHash if bipartite ShockHash is used.

4 Analysis

In this section we analyze non-bipartite MorphisHash for large n. Our analysis is split in two parts. First, we analyze the number of components of a random pseudoforest. We use this in the second part, to show the space efficiency of MorphisHash compared to ShockHash. Note that we employ the common simple uniform hashing assumption [21], which assumes that hash functions behave like truly random functions.

4.1 The Number of Components in a Random Pseudoforest

In a pseudoforest each component is a tree with one additional edge. An equivalent view is that each component of the pseudoforest is a cycle with trees branching from it.

Our first result uses the following graph model. Let $R = [n]^{[n]}$ be the set of all functions from $[n]$ to $[n]$. Every $r \in R$ corresponds to a directed graph $G_r = ([n], \{(v, r(v)) \mid v \in [n]\})$. All G_r are pseudoforests, because if we start from any node and follow its edges we will eventually end up in a cycle. Furthermore, there can only be one cycle in each component because each node has an out-degree of one. In the following we refer to all G_r as *maximal directed pseudoforests*. All edges of the pseudoforest which are not part of the cycle are pointed towards the cycle and all edges in the cycle are all directed in the same direction. We will first need the following results.

▶ **Lemma 2** ([25]). *Let $T(n, i)$ be the number of undirected forests having node set $[n]$ with i components where nodes $1, 2, \ldots, i$ belong to different trees. We have $T(n, i) = in^{n-i-1}$.*

▶ **Lemma 3.** $|\{r \in R \mid \text{all components in } G_r \text{ are cycles}\}| = n!$

Proof. If each node in G_r is part of a cycle then the indegree of each node is 1. Hence, r is a bijection and there are $n!$ possible bijections. ◀

We can now combine the previous results to analyze how the number of nodes in cycles is distributed in maximal directed pseudoforests.

▶ **Lemma 4.** $|\{r \in R \mid G_r \text{ is pseudoforest and has } i \text{ nodes in cycles}\}| = \frac{in^{n-1-i}n!}{(n-i)!}$.

Proof. There are $\binom{n}{i}$ ways of partitioning the nodes into (1) cycles with a total of i nodes and (2) trees that are attached to the cycles. According to Lemma 3 there are $i!$ ways to create cycles with i nodes. The i nodes in the cycles are the roots of trees. According to Lemma 2 the number of labeled rooted trees with n nodes and i roots is in^{n-i-1} . This number holds for graphs where each node has one outgoing edge, because all edges are uniquely oriented towards their parent node. We therefore have

$$\binom{n}{i} i! i n^{n-i-1} = \frac{n!}{i!(n-i)!} i! i n^{n-i-1} = \frac{i n^{n-1-i} n!}{(n-i)!}$$ ◀

The previous result is based on maximal directed pseudoforests. However, in MorphisHash we sample graphs in a different manner. Let $Q = [n]^{S \times \{0,1\}}$ be the set of functions from $S \times \{0,1\}$ to $[n]$. Each $q \in Q$ corresponds to a graph $G_q = ([n], \{\{q(x,0), q(x,1)\} \mid x \in S\})$. We refer to the set of all G_q as the *hashed* graph model. Graphs in this model may have multiple edges and loops. MorphisHash uniformly samples functions $q \in Q$. We are interested in the distribution of G_q conditioned on the event that G_q is a pseudoforest. In the following, we transfer our result of maximal directed pseudoforests to the hashed graph model.

▶ **Lemma 5.** *Let $c(G)$ be the number of components of G, C_n an appropriate normalization constant, $q \sim \mathcal{U}(Q)$ and $r \sim \mathcal{U}(R)$ then*

$$\begin{aligned} p_n(i) :=& \mathbb{P}[G_q \text{ has } i \text{ nodes in cycles} \mid G_q \text{ is pseudoforest}] \\ =& C_n \mathbb{E}[2^{-c(G_r)} \mid G_r \text{ has } i \text{ nodes in cycles}] \frac{i}{n^i (n-i)!} \end{aligned}$$

Proof. We begin with a relation between maximal directed pseudoforests G_r and hashed pseudoforests G_q. A hashed pseudoforest G_q is related to a maximal directed pseudoforest G_r if we can obtain G_r by orienting the edges of G_q. Let $a_{k,d,l}$ be the number of maximal directed pseudoforests and $b_{k,d,l}$ the number of hashed pseudoforest with i nodes in cycles, k components, d double edges and l loops. Within this subset, each cycle of the hashed graph can be oriented in two possible ways to obtain a maximal directed pseudoforest. However, changing the orientation of cycles that are a loop or double edge does not result in different maximal directed pseudoforests. Hence, we have $\frac{a_{k,d,l}}{b_{k,d,l}} = 2^{k-d-l}$ orientations which result in different maximal directed pseudoforest.

Let $c_{k,d,l}$ be the number of elements $q \in Q$ such that G_q is a pseudoforest with i nodes in cycles, k components, d double edges and l loops. We show that $\frac{c_{k,d,l}}{b_{k,d,l}} = n! 2^{n-d-l}$. We have to consider the number of functions in Q that result in the same hashed pseudoforest. The order of the edges can be permuted in $n!$ ways without changing the underlying pseudoforest, except for double edges. The number of possible permutations decreases by a factor of two for each double edge (2^{-d}). Analogously, the nodes within the n edges can be switched without changing the underlying graph in 2^n ways, except for loops (2^{-l}).

By definition we have $|Q_i| = \sum_{k,d,l} c_{k,d,l}$ and $|R_i| = \sum_{k,d,l} a_{k,d,l}$, where $Q_i \subset Q$ and $R_i \subset R$ are the pseudoforests with i nodes in cycles. For brevity let

$$p_{k,d,l} := \frac{a_{k,d,l}}{|R_i|} = \mathbb{P}[G_r \text{ has } k \text{ components, } d \text{ double edges and } l \text{ loops} \mid G_r \text{ has } i \text{ nodes cycles}]$$

We have

$$\begin{aligned}
p_n(i) &= \frac{|Q_i|}{\sum_j |Q_j|} = \frac{|R_i|}{\sum_j |Q_j|} \frac{\sum_{k,d,l} c_{k,d,l}}{|R_i|} = \frac{|R_i|}{\sum_j |Q_j|} \sum_{k,d,l} \frac{a_{k,d,l} c_{k,d,l}}{a_{k,d,l}|R_i|} \\
&= \frac{|R_i|}{\sum_j |Q_j|} \sum_{k,d,l} p_{k,d,l} \frac{c_{k,d,l}}{a_{k,d,l}} = \frac{|R_i|}{\sum_j |Q_j|} \sum_{k,d,l} p_{k,d,l} \frac{c_{k,d,l}}{b_{k,d,l}} \frac{b_{k,d,l}}{a_{k,d,l}} \\
&= \frac{|R_i|}{\sum_j |Q_j|} \sum_{k,d,l} p_{k,d,l} n! 2^{n-d-l} 2^{-k+d+l} = \frac{|R_i| n! 2^n}{\sum_j |Q_j|} \sum_{k,d,l} p_{k,d,l} 2^{-k} \\
&= \frac{|R_i| n! 2^n}{\sum_j |Q_j|} \mathbb{E}[2^{-c(G_r)} \mid G_r \text{ has } i \text{ nodes in cycles}] \\
&= C_n \mathbb{E}[2^{-c(G_r)} \mid G_r \text{ has } i \text{ nodes in cycles}] \frac{i}{n^i (n-i)!}
\end{aligned}$$

Where we used Lemma 4 for $|R_i|$. $\sum_j |Q_j|$ is the number of all hashed pseudoforests and thus only depends on n. C_n is chosen such that the probabilities add up to 1. ◀

The next step is to analyze $\mathbb{E}[2^{-c(G_r)} \mid G_r \text{ has } i \text{ nodes in cycles}]$. To this end, we first require some definitions and more general results.

▶ **Lemma 6.** *The number of components of a pseudoforest with n nodes and i nodes in cycles follows the same distribution as the number of components of a graph of i nodes where each component is a cycle.*

Note that this refers to both graph models G_r and G_q.

Proof. The order in which the edges of a graph are sampled does not change the distribution of the graph. Given a graph of n nodes and i edges such that all edges are part of cycles, the probability that the remaining $n - i$ edges form trees with the root being part of the existing cycles is independent of how the nodes of the cycles are connected to form any number of components. ◀

Configuration Model. The *configuration model* [20] can be used to describe distributions of random graphs. In the model, each node is given a fixed number of half-edges. The graph is obtained by repeatedly connecting half-edges by uniformly sampling from the set of all remaining half-edges.

▶ **Lemma 7.** *The distribution of the hashed graph G_q with $q \sim \mathcal{U}(Q)$ is equal to the distribution of the following graph.*

The graph is obtained in two steps. First, the degree of each node is revealed by distributing $2n$ half-edges. In a second step, the edges of the graph are obtained in a sequence of n rounds. In each round an unmatched half-edge $i \in [2n]$ is chosen arbitrarily and matched to a distinct unmatched half-edge $j \in [2n]$, chosen uniformly at random. The choice of i may depend on the set of half-edges matched previously.

We refer to ShockHash [15, Lemma 5] for a proof. We require an analogous result for G_r

▶ **Lemma 8.** *The distribution of the maximal directed pseudoforest G_r with $r \sim \mathcal{U}(R)$ is equal to the distribution of the following directed graph.*

The number of outgoing edges of each node is fixed to 1. The graph is obtained in two steps. First, the indegree of each node is revealed by distributing n incoming half-edges. In a second step, the edges of the graph are obtained in a sequence of n rounds. In each round an unmatched outgoing half-edge at node $i \in [n]$ is chosen arbitrarily and matched to a distinct unmatched incoming half-edge $j \in [n]$, chosen uniformly at random. The choice of i may depend on the set of half-edges matched previously.

Again, the proof is analogous to ShockHash [15, Lemma 5].

▶ **Lemma 9.** *For $i \in \mathbb{N}$: $\prod_{k=2}^{i}(1-\frac{1}{k}) = \frac{1}{i}$*

Proof. By induction $\prod_{k=2}^{1}(1-\frac{1}{k}) = 1$ and

$$\prod_{k=2}^{i+1}(1-\frac{1}{k}) = (1-\frac{1}{i+1})\prod_{k=2}^{i}(1-\frac{1}{k}) = (1-\frac{1}{i+1})\frac{1}{i} = \frac{1}{i+1}$$ ◀

We now have the available tools to analyze the number of components conditioned on the number of nodes in cycles.

▶ **Lemma 10.** $\frac{1}{4}i^{-1/2} \leq \mathbb{E}[2^{-c(G_r)} \mid G_r \text{ has } i \text{ nodes in cycles}] \leq i^{-1/2}$, *where $r \sim \mathcal{U}(R)$ and $c(G_r)$ is the number of components of G_r.*

Note that the following proof has similarities to ShockHash [15, Lemma 6].

Proof. We consider the number of components of a maximal directed pseudoforest G_r^i with i nodes, $R = [i]^{[i]}$, $r \sim \mathcal{U}(R)$, conditioned on the event that each component of G_r^i is a cycle. According to Lemma 6 the number of components of G_r^i follows the same distribution as the number of components of a maximal directed pseudoforest with n nodes and i nodes in cycles. We proceed as described in Lemma 8 by revealing the graph in a sequence of rounds. First, the indegree of each node is revealed. Each component is a circle and therefore each node has indegree one. We choose the outgoing half-edge at an arbitrary node x. The outgoing half-edge is matched with one of the i incoming half-edges. Let y be the node at the incoming half-edge. There are two cases.

1. With probability $\frac{1}{i}$ we have $x = y$. In this case, a loop is created, a cycle is closed and the next outgoing half-edge to match is chosen arbitrarily.
2. Otherwise we merge the nodes to a single one which does not change the number of components. The outgoing half-edge at node y is matched next.

In both cases, the distribution of the remaining graph is that of G_r^{i-1}. Because of this independence, we can multiply the expectation values to obtain the recurrence

$$\mathbb{E}[2^{-c(G_r^i)}] = \frac{1}{2}\frac{1}{i}\mathbb{E}[2^{-c(G_r^{i-1})}] + \left(1-\frac{1}{i}\right)\mathbb{E}[2^{-c(G_r^{i-1})}] = \left(1-\frac{1}{2i}\right)\mathbb{E}[2^{-c(G_r^{i-1})}]$$

With the base case $\mathbb{E}[2^{-c_0}] = 1$ the recurrence is solved and upper bounded by

$$\begin{aligned}\mathbb{E}[2^{-c(G_r^i)}] &= \prod_{k=1}^{i} 1-\frac{1}{2k} = \sqrt{\prod_{k=1}^{i}\left(1-\frac{1}{2k}\right)^2} \leq \sqrt{\prod_{k=1}^{i}\left(1-\frac{1}{2k}\right)\left(1-\frac{1}{2k+1}\right)} \\ &= \sqrt{\prod_{k=2}^{2i+1}\left(1-\frac{1}{k}\right)} = \sqrt{\frac{1}{2i+1}} \leq i^{-1/2}\end{aligned}$$

Where we used Lemma 9. Analogously, a lower bound is

$$\begin{aligned}\mathbb{E}[2^{-c(G_r^i)}] &= \sqrt{\left(1-\frac{1}{2}\right)^2 \prod_{k=2}^{i}\left(1-\frac{1}{2k}\right)^2} = \frac{1}{2}\sqrt{\prod_{k=2}^{i}\left(1-\frac{1}{2k}\right)^2} \\ &\geq \frac{1}{2}\sqrt{\prod_{k=2}^{i}\left(1-\frac{1}{2k-1}\right)\left(1-\frac{1}{2k-2}\right)} \geq \frac{1}{2}\sqrt{\prod_{k=2}^{2i}\left(1-\frac{1}{k}\right)} \\ &= \frac{1}{2}\sqrt{\frac{1}{2i}} \geq \frac{1}{4}i^{-1/2}\end{aligned}$$

◀

A similar result for hashed pseudoforests instead of maximal directed pseudoforests is the following.

▶ **Lemma 11.** *$\mathbb{P}[c(G_q) \geq \frac{1}{4}\ln(i) \mid G_q$ is pseudoforest with i nodes in cycles$] \in \Omega_{i\to\infty}(1)$, where $q \sim \mathcal{U}(Q)$ and $c(G_q)$ is the number of components of G_q.*

Proof. The proof is analogous to the previous one. We use Lemma 7 to match a half-edge to one of the remaining $2i-1$ half-edges. In each round the number of components increases by one with probability $\frac{1}{2i-1}$. Resolving the respective recurrence results in

$$\begin{aligned}&\mathbb{E}[c(G_q) \mid G_q \text{ is pseudoforest with } i \text{ nodes in cycles}] \\ =\ & (\frac{1}{2i-1}+\frac{1}{2i-3}+\frac{1}{2i-5}+\frac{1}{2i-7}+\ldots+\frac{1}{3}+1) \\ =&\frac{1}{2}(\frac{1}{2i-1}+\frac{1}{2i-1}+\frac{1}{2i-3}+\frac{1}{2i-3}+\ldots+1+1) \\ >&\frac{1}{2}(\frac{1}{2i-0}+\frac{1}{2i-1}+\frac{1}{2i-2}+\frac{1}{2i-3}+\ldots+\frac{1}{2}+1) \\ =&\frac{1}{2}H_{2i} > \frac{1}{2}\ln(i)\end{aligned}$$

Where H_k is the k-th harmonic number. We are interested in a lower bound for the probability that the number of components is at least $\frac{1}{4}\ln(i)$. We can use the variance to find such a bound. The probability distribution of the number of components can be described in terms of a Poisson binomial distribution. The variance of a Poisson binomial distribution can be bounded above by the expected value. Clearly, $H_{2r} < \ln(3i)$ is an upper bound for the variance. Using Cantelli's inequality ($\mathbb{P}[X \geq \mathbb{E}[X]+x] \geq \frac{x^2}{x^2+\sigma^2}$) we find with $x = -\frac{1}{4}\ln(i)$ that with probability $\frac{(\frac{1}{4}\ln(i))^2}{(\frac{1}{4}\ln(i))^2+\ln(3i)} \in \Omega(1)$ the pseudoforest has at least $\frac{1}{4}\ln(i)$ components. ◀

The previous result shows that the number of components increases logarithmically in the number of nodes in cycles. The next step is therefore to show that the cycles are sufficiently large.

▶ **Lemma 12.** *$\mathbb{P}[G_q$ has at least $\sqrt{n}$ nodes in cycles $\mid G_q$ is pseudoforest$] \in \Omega_{n\to\infty}(1)$, where $q \sim \mathcal{U}(Q)$.*

Proof. Let $p_n(i)$ be the probability of sampling a hashed pseudoforest with i nodes in cycles as determined in Lemma 5. We need to show that the probability of sampling a pseudoforest with at least $\sqrt{n}$ nodes in cycles is at least a constant factor larger than sampling a pseudoforest with less than $\sqrt{n}$ nodes in cycles, i.e. $\left(\sum_{i=\sqrt{n}}^{n} p_n(i)\right) / \left(\sum_{i=1}^{\sqrt{n}} p_n(i)\right) \in \Omega(1)$.

A stronger statement is $\left(\sum_{i=\sqrt{n}}^{2\sqrt{n}} p_n(i)\right) / \left(\sum_{i=1}^{\sqrt{n}} p_n(i)\right) \in \Omega(1)$. In the following we show $\frac{p_n(\sqrt{n}+x)}{p_n(\sqrt{n}-x)} \in \Omega(1)$ for all $0 \leq x \leq \sqrt{n}$. This pointwise comparison is an even stronger statement. For brevity, we omit rounding of $\sqrt{n}$. Using the bounds of Lemma 10 and Stirling's approximation we have:

$$
\begin{aligned}
&\frac{p_n(\sqrt{n}+x)}{p_n(\sqrt{n}-x)} \geq \frac{1}{4}\frac{(\sqrt{n}-x)^{1/2}}{(\sqrt{n}+x)^{1/2}} \cdot \frac{\sqrt{n}+x}{\sqrt{n}-x} \cdot \frac{n^{\sqrt{n}-x}}{n^{\sqrt{n}+x}} \cdot \frac{(n-\sqrt{n}+x)!}{(n-\sqrt{n}-x)!} \\
&= \frac{1}{4}\left(\frac{\sqrt{n}+x}{\sqrt{n}-x}\right)^{1/2} \cdot n^{-2x} \cdot \sqrt{\frac{n-\sqrt{n}+x}{n-\sqrt{n}-x}} \cdot \frac{e^{n-\sqrt{n}-x}}{e^{n-\sqrt{n}+x}} \cdot \frac{(n-\sqrt{n}+x)^{n-\sqrt{n}+x}}{(n-\sqrt{n}-x)^{n-\sqrt{n}-x}} \\
&> \frac{1}{4}n^{-2x} \cdot e^{-2x} \cdot \left(\frac{n-\sqrt{n}+x}{n-\sqrt{n}-x}\right)^{n-\sqrt{n}} \cdot \big((n-\sqrt{n}+x)(n-\sqrt{n}-x)\big)^{x} \\
&= \frac{1}{4}e^{-2x} \cdot \left(\frac{1+\frac{x}{n-\sqrt{n}}}{1-\frac{x}{n-\sqrt{n}}}\right)^{n-\sqrt{n}} \cdot \left(\frac{(n-\sqrt{n}+x)(n-\sqrt{n}-x)}{n^2}\right)^{x} \\
&> \frac{1}{4}e^{-2x} \cdot \frac{e^{x}}{e^{-x}} \cdot \left(\frac{(n-\sqrt{n}+\sqrt{n})(n-\sqrt{n}-\sqrt{n})}{n^2}\right)^{\sqrt{n}} \\
&= \frac{1}{4}\left(\frac{n(n-2\sqrt{n})}{n^2}\right)^{\sqrt{n}} = \frac{1}{4}\left(1-\frac{2}{\sqrt{n}}\right)^{\sqrt{n}} = \frac{1}{4}\frac{1}{e^2}
\end{aligned}
$$

◀

Combining the results gives us the following.

▶ **Theorem 13.** *$\mathbb{P}[c(G_q) \geq \frac{1}{8}\ln(n) \mid G_q$ is pseudoforest$] \in \Omega_{n\to\infty}(1)$, where $c(G_q)$ is the number of components of G_q and $q \sim \mathcal{U}(Q)$.*

Proof. According to Lemma 12 at least $\sqrt{n}$ nodes are in cycles with constant probability. Using Lemma 11 this results in $\frac{1}{4}\ln(\sqrt{n}) = \frac{1}{8}\ln(n)$ components with constant probability. ◀

▶ **Corollary 14.** *$\mathbb{E}[c(G_q) \mid G_q$ is pseudoforest$] \in \Theta_{n\to\infty}(\ln(n))$, where $c(G_q)$ is the number of components of G_q and $q \sim \mathcal{U}(Q)$.*

Proof. Use Theorem 13 for the lower bound. There can be at most all n nodes in cycles of the pseudoforest and we can apply Lemma 11 for the upper bound. ◀

4.2 Space Savings of MorphisHash

We now show that MorphisHash can convert each component into space savings compared to ShockHash. As a first step we transition the previous graph results into the world of matrices.

▶ **Lemma 15.** *The defect of the incidence matrix A of a pseudoforest is at least the number of the pseudoforests components.*

Proof. For each component C, summing up the rows A_j of the respective nodes $j \in C$ results in zero rows because for each component, the two endpoints of each edge are included in the summed up rows. The rows summed up for each zero row are disjoint and therefore in particular linearly independent combinations. ◀

▶ **Lemma 16** ([4])**.** *The probability that a random square matrix (of any field) with n rows and columns has full rank approaches 1 for large n.*

▶ **Lemma 17.** *The probability that a random rectangular matrix (of any field) with n rows and $m \in o(n)$ columns has full rank approaches 1 for large n.*

Proof. A necessary condition that a square matrix with n rows has full rank is that the first m columns have full rank. The probability that this rectangular submatrix has full rank is therefore bounded below by Lemma 16. ◀

We now show our main result.

▶ **Theorem 18.** *$\mathbb{P}[\exists x : AHx = d \mid G_q$ is pseudoforest$] \in \Omega_{n\to\infty}(1)$, where $q \sim \mathcal{U}(Q)$, incidence matrix A and vector d of G_q are as described in the algorithm, $b = n - \frac{1}{9}\ln(n)$ and $H \sim \mathcal{U}(\mathbb{F}_2^{n\times b})$.*

Proof. In Theorem 13 we showed that a hashed pseudoforest has at least $\frac{1}{8}\ln(n)$ components with constant probability. According to Lemma 15 a direct consequence is that the incidence matrix A of a random pseudoforest has a defect of at least $\frac{1}{8}\ln(n)$ with constant probability. According to Lemma 17, the probability that H has full rank is at least constant. The system $AHx = d$ has a solution if there is a vector y which solves $Ay = d$ and simultaneously $Hx = y$. Such a vector y exists if the two solution spaces have to intersect, which happens once $\mathrm{def}(A) + \mathrm{rank}(H) - n > 0$. Since A and H are uncorrelated we have with constant probability that both H has full rank b and simultaneously A has at least a defect of $\frac{1}{8}\ln(n)$. With $b = n - \frac{1}{9}\ln(n)$ we have $\mathrm{def}(A) + \mathrm{rank}(H) - n \geq \frac{1}{8}\ln(n) + (n - \frac{1}{9}\ln(n)) - n > 0$ with constant probability. ◀

We use the above result to measure the space savings compared to ShockHash.

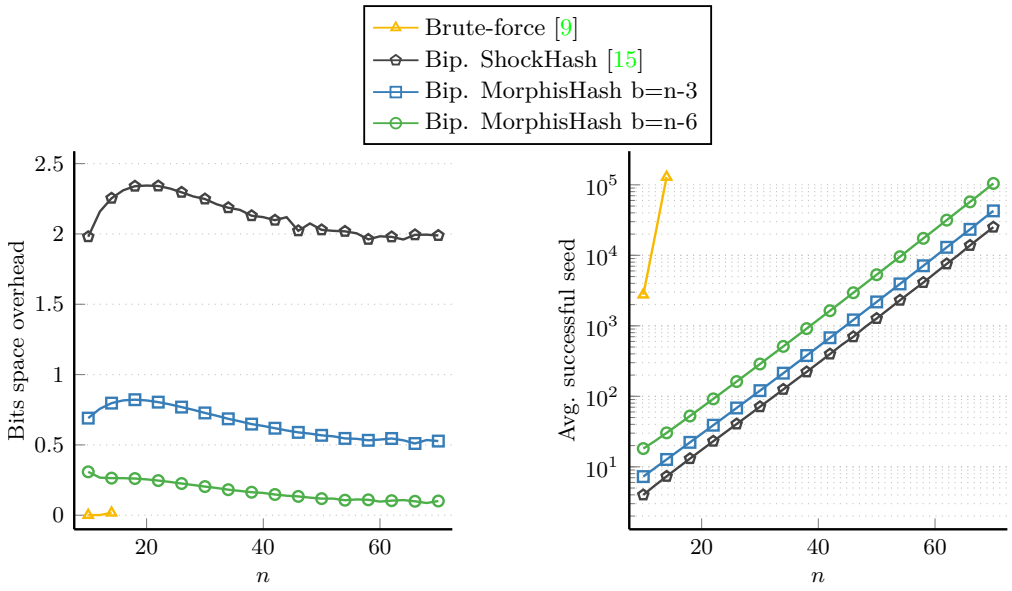
▶ **Corollary 19.** *Compared to ShockHash, MorphisHash is at least $\frac{1}{9}\ln(n) - \mathcal{O}(1)$ bits more space efficient in expectation while requiring a constant factor more time.*

Proof. ShockHash requires at least n bits to store the orientation of all keys in a retrieval structure. MorphisHash has to store the solution vector x instead, requiring exactly b bits. According to Theorem 18 we can choose $b = n - \frac{1}{9}\ln(n)$ to obtain the desired space savings. However, there is a constant probability that a seed pair has to be rejected because there is no solution for x. MorphisHash therefore has to check a constant factor more seeds in expectation, consequently increasing the expected space required to store the seed by a constant number of bits. Analogously, the construction time grows by a constant factor in expectation. The time required to solve an expected constant number of equation systems is dominated by the seed search. ◀

5 Partitioning

The time required to find a seed in MorphisHash and ShockHash grows exponentially with n. To keep construction feasible for a large number of keys, we first partition the input into equally sized subsets of manageable size. For consistency, we use n for the size of those subsets, and refer to them as *base cases* in the following. MorphisHash is then applied on those base cases. Note that the following two partitioning schemes are also applied on ShockHash and we refer to their paper for more details [15, Section 7].

MorphisHash-RS. We use RecSplit [9] to recursively split the input into smaller subsets. Once sufficiently small subsets are obtained we apply MorphisHash on those subsets as a base case. RecSplit is space efficient but has significant query time overheads caused by traversing the tree. In MorphisHash-RS we store the solution vector x of the retrieval structure directly next to the corresponding seed to improve locality.



(a) Idealized space overhead over the lower bound $\log_2(n^n/n!)$. For an average successful seed s we charge $\log_2(s)$ bits plus the bits required for retrieval i.e. n for ShockHash and b for MorphisHash.

(b) Average successful seed. Note that we use Bip. ShockHash and Bip. MorphisHash without the quad split technique (see [15, Section 8.3]).

Figure 2 Space and avg. successful seed for ShockHash, MorphisHash and brute force search.

MorphisHash-Flat. The input keys are first hashed into buckets. Using thresholds, some keys are bumped such that the bucket does not exceed the desired base case size n. The bumped keys are then used to fill up buckets which did not reach the desired size. We apply MorphisHash on each bucket.

6 Experiments

In this section we experimentally evaluate MorphisHash. We show the effect of the new parameter b introduced by MorphisHash. We then compare MorphisHash-Flat and MorphisHash-RS with state-of-the-art competitors. Our source code is public under the General Public License [10]. We integrate MorphisHash into an existing benchmark framework [13], which was used for the comparison with competitors. The benchmark framework is described in detail in [14]. We perform all experiments on a Core i7-11700 CPU which has 48 KiB L1 and 512 KiB L2 data cache per core. The CPU has a total of 16 MiB L3 cache. The machine has 64 GiB of dual-channel DDR4-3200 RAM. Note that our experiments are at a scale where variances are relatively small, we therefore omit them for better readability.

6.1 MorphisHash vs ShockHash without partitioning

In the following we compare Bipartite MorphisHash with Bip. ShockHash. MorphisHash has the additional parameter b, which determines the size of the retrieval structure. Smaller b values result in less space overhead at the cost of more seed tests. Note that in theory we choose $b = n - \frac{1}{9}\ln(n)$ but in practice where n is small it is more intuitive to work with n minus a small constant. Figure 2 shows that Bip. MorphisHash has to check roughly a constant factor more seeds than Bip. ShockHash when b is fixed to n minus a constant. For

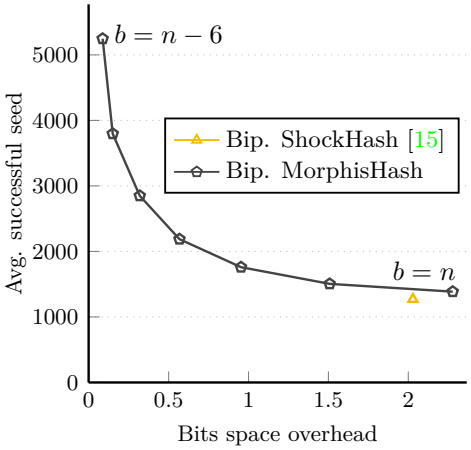


Figure 3 Comparing ShockHash with the MorphisHash trade-off between time and space for fixed $n = 50$ and $b = \{n-6, \ldots, n\}$.

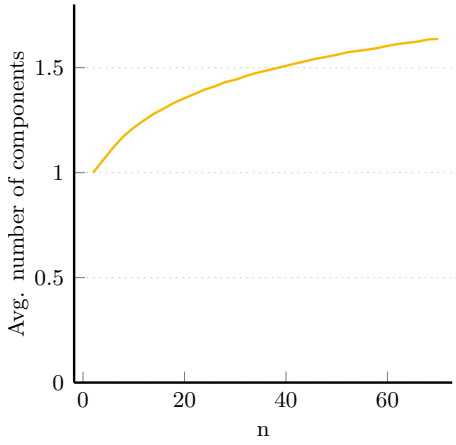


Figure 4 Experimentally measured average number of components when sampling bipartite pseudoforests.

$b = n - 6$ this factor is ≈ 4. At the same time, MorphisHash almost completely eliminates the remaining space overhead. MorphisHash comes below 0.1 bits of space overhead when using $b = n - 6$ while ShockHash has about 2 bits of space overhead. Thus, MorphisHash has roughly 20 times less space overhead. For $n = 54$, MorphisHash has $\frac{0.11}{54} \approx 0.002$ bits per key space overhead.

Figure 3 gives another perspective. In this plot, we fix n and vary b. Interestingly, ShockHash even outperforms MorphisHash for $b = n$. This is because MorphisHash requires n bits for the retrieval structure in this case just like ShockHash. However, there is still the chance that the equation system of MorphisHash has no solution resulting in more retries and therefore in a higher space consumption of the seed and a higher construction time. For smaller b the space overhead approaches 0. In the extreme case of $b = 0$ and respective zero dimensional retrieval vector, MorphisHash is equivalent to simple brute force search because all keys can only use the h_0 candidate function. The average successful seed grows rapidly with smaller b as it is increasingly less likely to stumble upon a pseudoforest which has at least $n - b$ many components. A pseudoforest with less than $n - b$ components may still result in a solvable (overdetermined) equation system, but this becomes exponentially less likely with every component below $n - b$. Figure 3 uses $n = 50$. The expected number of components for $n = 50$ is 1.56 as shown in Figure 4.

6.2 Choosing b in MorphisHash-RS and MorphisHash-Flat

Space improvements in MorphisHash-RS and MorphisHash-Flat can be made either by (1) increasing the base case size n which reduces the space overhead of the partitioning technique or (2) decreasing b which reduces the space overhead of MorphisHash. In both cases the construction time increases. We experimented with different values of b to identify the configurations which dominate the construction time and space trade-off. We determined that $b = n - 4$ is a good choice for MorphisHash-RS and $b = n - 2$ for MorphisHash-Flat. MorphisHash-Flat is a less space efficient partitioning technique compared to MorphisHash-RS. Space savings can be made more easily by increasing n instead of decreasing b.

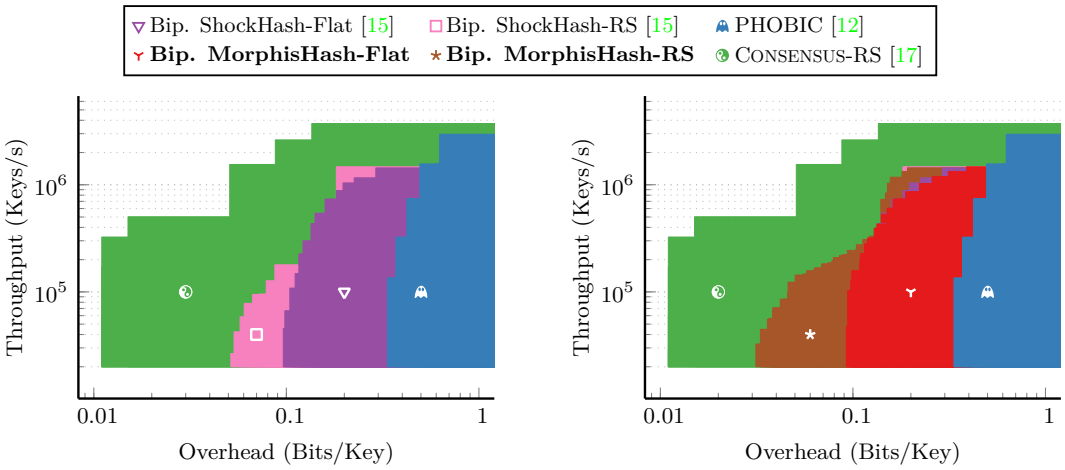


Figure 5 Dominance maps indicating the approach with the fastest queries, given a specific trade-off between space and construction time with (right) and without (left) MorphisHash on 100 M keys. Space overhead in bits per key over the lower bound of 1.44.

Table 1 Performance of various methods on 100 M keys. In the first part, configurations are chosen such that the construction times are about equal (sorted by space efficiency). In the second part, configurations are chosen such that space consumption is almost equal.

Method	Space (bits/key)	Query (ns/query)	Construction (ns/key)
Consensus-RS, k=32768, o=0.0025	1.447	222	6 733
Bip. MorphisHash-RS, base case size n=52, b=n-4	1.501	137	6 669
Bip. ShockHash-RS, base case size n=66	1.523	147	7 186
Bip. MorphisHash-Flat, base case size n=88, b=n-2	1.541	75	6 330
Bip. ShockHash-Flat, base case size n=96	1.554	76	6 676
PHOBIC, λ=8.85, IC-R	1.749	49	6 426
Bip. ShockHash-RS, base case size n=128	1.489	131	172 738
Bip. MorphisHash-RS, base case size n=64, b=n-4	1.489	139	8 085

6.3 Comparison to Competitors

We compare MorphisHash-RS and MorphisHash-Flat with state-of-the-art competitors. We select the following space efficient competitors based on a recent survey [14]: Consensus-RS[17], ShockHash-Flat, ShockHash-RS and PHOBIC [12]. We test a wide range of configurations for each competitor and compare them in Figure 5. A selection of configurations is shown in Table 1. As can be seen in the plot and the table, MorphisHash-RS is about 0.02 bits per key more space efficient than ShockHash-RS when compared for equal construction time. This corresponds to a reduction of 27% in space overhead. When compared for equal space consumption of 1.489 bits per key, MorphisHash-RS is $\frac{172\,738}{8085} \approx 21$ times faster to construct (see Table 1). For MorphisHash-Flat we select b less aggressively compared to MorphisHash-RS (Section 6.2) and obtain a space improvement of about 0.01 bits per key. According to Figure 5 MorphisHash dominates ShockHash in the overall space, construction and query time trade-off. The next best competitor in terms of space efficiency is PHOBIC which is a clear winner in terms of query throughput. In the other direction, we have the

recently published CONSENSUS-RS, which can reach space overheads as low as 0.001 bits per key at the cost of additional query time. A negative result regarding non-minimal PHFs can be found in the full version of this paper [11].

7 Conclusion and Future Work

MorphisHash almost completely eliminates the remaining redundancy in ShockHash. This is particularly effective when combined with a space efficient partitioning technique. Our compression scheme might be of a more general interest, further examples can be found in the full version of this paper [11].

In future work we plan to improve the space efficiency of partitioning techniques as those are a major source of space overhead. We are hopeful that a partitioning technique that involves the novel CONSENSUS technique puts further trade-offs for MorphisHash into reach.

References

1 Djamal Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. Fast prefix search in little space, with applications. In *ESA (1)*, volume 6346 of *Lecture Notes in Computer Science*, pages 427–438. Springer, 2010. doi:10.1007/978-3-642-15775-2_37.

2 Djamal Belazzougui and Gonzalo Navarro. Alphabet-independent compressed text indexing. *ACM Trans. Algorithms*, 10(4):23:1–23:19, 2014. doi:10.1145/2635816.

3 Dominik Bez, Florian Kurpicz, Hans-Peter Lehmann, and Peter Sanders. High performance construction of recsplit based minimal perfect hash functions. In *ESA*, volume 274 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.19.

4 Jean Bourgain, Van H Vu, and Philip Matchett Wood. On the singularity probability of discrete random matrices. *Journal of Functional Analysis*, 258(2):559–603, 2010.

5 Andrei Z. Broder and Michael Mitzenmacher. Survey: Network applications of Bloom filters: A survey. *Internet Math.*, 1(4):485–509, 2003. doi:10.1080/15427951.2004.10129096.

6 Chin-Chen Chang and Chih-Yang Lin. Perfect hashing schemes for mining association rules. *Comput. J.*, 48(2):168–179, 2005. doi:10.1093/COMJNL/BXH074.

7 Victoria G. Crawford, Alan Kuhnle, Christina Boucher, Rayan Chikhi, and Travis Gagie. Practical dynamic de bruijn graphs. *Bioinform.*, 34(24):4189–4195, 2018. doi:10.1093/BIOINFORMATICS/BTY500.

8 Peter C Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer. Fast succinct retrieval and approximate membership using ribbon. *arXiv preprint*, 2021. arXiv:2109.01892.

9 Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. RecSplit: Minimal perfect hashing via recursive splitting. In *ALENEX*, pages 175–185. SIAM, 2020. doi:10.1137/1.9781611976007.14.

10 Stefan Hermann. MorphisHash - GitHub. https://github.com/stefanfred/MorphisHash, 2025.

11 Stefan Hermann. Morphishash: Improving space efficiency of shockhash for minimal perfect hashing. *arXiv preprint*, 2025. doi:10.48550/arXiv.2503.10161.

12 Stefan Hermann, Hans-Peter Lehmann, Giulio Ermanno Pibiri, Peter Sanders, and Stefan Walzer. PHOBIC: perfect hashing with optimized bucket sizes and interleaved coding. In *ESA*, volume 308 of *LIPIcs*, pages 69:1–69:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ESA.2024.69.

13 Hans-Peter Lehmann. MPHF Experiments – GitHub. https://github.com/ByteHamster/MPHF-Experiments, 2025.

14 Hans-Peter Lehmann, Thomas Mueller, Rasmus Pagh, Giulio Ermanno Pibiri, Peter Sanders, Sebastiano Vigna, and Stefan Walzer. Modern minimal perfect hashing: A survey. *arXiv preprint*, 2025. doi:10.48550/arXiv.2506.06536.

15 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. ShockHash: Near optimal-space minimal perfect hashing beyond brute-force. *arXiv preprint, invited to Algorithmica*, 2024. doi:10.48550/arXiv.2310.14959.

16 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. Shockhash: Towards optimal-space minimal perfect hashing beyond brute-force. In *ALENEX*. SIAM, 2024. doi:10.1137/1.9781611977929.15.

17 Hans-Peter Lehmann, Peter Sanders, Stefan Walzer, and Jonatan Ziegler. Combined search and encoding for seeds, with an application to minimal perfect hashing. In *ESA*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025.

18 Yi Lu, Balaji Prabhakar, and Flavio Bonomi. Perfect hashing for network applications. In *ISIT*, pages 2774–2778. IEEE, 2006. doi:10.1109/ISIT.2006.261567.

19 Kurt Mehlhorn. On the program size of perfect and universal hash functions. In *FOCS*, pages 170–175. IEEE Computer Society, 1982. doi:10.1109/SFCS.1982.80.

20 MARK EJ Newman. Networks: an introduction, 2010.

21 Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM Journal on Computing*, 38(1):85–96, 2008. doi:10.1137/060658400.

22 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004. doi:10.1016/J.JALGOR.2003.12.002.

23 Giulio Ermanno Pibiri. Sparse and skew hashing of k-mers. *Bioinformatics*, 38(Supplement_1):i185–i194, 2022. doi:10.1093/BIOINFORMATICS/BTAC245.

24 Giulio Ermanno Pibiri and Rossano Venturini. Efficient data structures for massive N-gram datasets. In *SIGIR*, pages 615–624. ACM, 2017. doi:10.1145/3077136.3080798.

25 Lajos Takács. On cayley's formula for counting forests. *Journal of Combinatorial Theory, Series A*, 53(2):321–323, 1990. doi:10.1016/0097-3165(90)90064-4.

Connected Partitions via Connected Dominating Sets

Aikaterini Niklanovits
Hasso Plattner Institute, University of Potsdam, Germany

Kirill Simonov
Department of Informatics, University of Bergen, Norway

Shaily Verma
Hasso Plattner Institute, University of Potsdam, Germany

Ziena Zeif
Hasso Plattner Institute, University of Potsdam, Germany

Abstract

The classical theorem due to Győri and Lovász states that any k-connected graph G admits a partition into k connected subgraphs, where each subgraph has a prescribed size and contains a prescribed vertex, as long as the total size of target subgraphs is equal to the size of G. However, this result is notoriously evasive in terms of efficient constructions, and it is still unknown whether such a partition can be computed in polynomial time, even for $k = 5$.

We make progress towards an efficient constructive version of the Győri–Lovász theorem by considering a natural strengthening of the k-connectivity requirement. Specifically, we show that the desired connected partition can be found in polynomial time, if G contains k disjoint connected dominating sets. As a consequence of this result, we give several efficient approximate and exact constructive versions of the original Győri–Lovász theorem:

- On general graphs, a Győri–Lovász partition with k parts can be computed in polynomial time when the input graph has connectivity $\Omega(k \cdot \log^2 n)$;
- On convex bipartite graphs, connectivity of $4k$ is sufficient;
- On biconvex graphs and interval graphs, connectivity of k is sufficient, meaning that our algorithm gives a "true" constructive version of the theorem on these graph classes.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Győri–Lovász theorem, connected dominating sets, graph classes

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.10

Related Version *Full Version*: `https://arxiv.org/abs/2503.13112` [33]

1 Introduction

Partitioning a graph into connected subgraphs is a problem that naturally arises in various application areas such as image processing, road network decomposition and parallel graph systems for supporting computations on large-scale graphs [30, 31, 18, 21]. Usually in such applications, there is a certain number of parts k that the partition should have, as well as certain constraints on the size of each part. In particular, a prominent objective is for the size of the parts to be as equal as possible and, therefore, for the partition to be as balanced as possible, known as Balanced Connected Partition problem (BCP_k). This problem is proven to be NP-complete [19] and even hard to approximate [14] and, hence, it has been studied in restricted graph classes such as grid graphs for which it was again proven to be NP-hard [3].

From a theoretical point of view, partitioning a graph into connected components each of a certain, chosen in advance, size is of independent interest as it gives us an insight into the structure of the graph. With the simple example of a star graph, one can observe that it


© Aikaterini Niklanovits, Kirill Simonov, Shaily Verma, and Ziena Zeif;
licensed under Creative Commons License CC-BY 4.0



is not possible to partition it into connected sets where more than one of them has size 2 or more. In general, if there exists a set of t vertices whose removal disconnects the graph (called a *separator*), and if the number of parts k is greater than t, then the k parts of a connected partition might not necessarily be of arbitrary size. Graphs that do not contain a separator of size strictly less than t are called *t-connected*.

The essence of this observation was captured in 1976 by Győri [22] and Lovász [29] who independently proved that the k-connectivity of a graph is a necessary and sufficient condition for any partition of a specific size to be possible, even when each part is additionally required to contain a specified vertex called a *terminal*. This was later generalized to weighted graphs by Chandran et al. [12], Chen et al. [13] and Hoyer [24]. Formally, we define the following computational problem.

▶ **Definition 1.** *In the GL-Partition problem, the input is a graph $G = (V, E)$, an integer $k \geq 1$, a set of k distinct vertices $C = \{c_1, \ldots, c_k\}$ of G, and k integers $N = \{n_1, \ldots, n_k\}$ such that $\sum_{i=1}^{k} n_i = |V|$, in short denoted as (G, k, C, N). The task is to compute a partition of V into k disjoint subsets $V_1, \ldots, V_k$ such that $|V_i| = n_i$, $c_i \in V_i$, and the induced subgraph $G[V_i]$ on V_i is connected, for all $i \in [k]$.*

We call the partition satisfying the property above a *Győri–Lovász partition*, or a *GL-partition* for short. The result of Győri and Lovász can then be stated as follows.

▶ **Theorem** (Győri–Lovász Theorem)**.** *Given an instance of GL-Partition where the graph is k-connected, a GL-partition always exists.*

Although the proof of Győri [22] is constructive, the resulting algorithm runs in exponential time. To this day, it remains open whether finding a GL-partition of a given k-connected graph could be done in polynomial time in general. On the other hand, a few polynomial algorithms have been developed for special cases. The original research direction is concerned with smaller values of k. Specifically, in 1990, Suzuki et al. [37, 38] provided a polynomial algorithm for the cases $k = 2$ and $k = 3$. Later, in 1993, Wada et al. [39] extended the result for $k = 3$, and, in 1997, Nakano et al. [32] showed a linear-time algorithm for the case where $k = 4$ and the graph is planar with all the terminals located on the same face of the planar embedding. More recently, in 2016, Hoyer and Thomas [24] presented a polynomial-time algorithm for the case $k = 4$ without additional restrictions on the graph structure. It is still open whether a polynomial-time algorithm exists for any fixed value of k starting from 5. Another research direction focuses on polynomial-time constructions for the Győri–Lovász theorem on specific graph classes, for all values of k. Recently, Casel et al. [9] developed a polynomial-time algorithm for chordal graphs and (with a small additive violation of the size constraint) for HHI_4^2-free graphs. Additionally, Borndörfer et al. [6] have shown an approximate version of the Győri–Lovász theorem on general graphs, where the size of each part is within a factor of 3 from the given target.

In the light of the challenges above, it is natural to ask whether efficient constructions exist for more restrictive versions of the Győri–Lovász theorem. Censor-Hillel, Ghaffari, and Kuhn [11] pioneered the notion of a connected dominating set (CDS) partition as a proxy for vertex connectivity of the graph. Formally, a graph admits a CDS partition of size k, if there exists a partition of the vertices into k sets, such that each set forms a dominating set of the graph, and induces a connected subgraph. It is easy to see that any graph that admits a CDS partition of size k, is also k-connected. The converse, unfortunately, does not hold; however, Censor-Hillel, Ghaffari, and Kuhn show that there is only a polylogarithmic gap between the connectivity and the size of the maximum CDS partition. Moreover, a suitable CDS partition can be efficiently computed. The bound was later improved by Censor-Hillel et al. [10].

▶ **Theorem 2** (Corollary 1.6, [10]). *Every k-connected n-vertex graph has a CDS partition of size $\Omega(\frac{k}{\log^2 n})$, and such a partition can be computed in polynomial time.*

On the other hand, Censor-Hillel, Ghaffari, and Kuhn [11] show that a logarithmic gap between the connectivity and the size of a CDS partition is necessary.

▶ **Theorem 3** (Theorem 1.3, [11]). *For any sufficiently large n, and any $k \geq 1$, there exist k-connected n-vertex graphs, where the maximum CDS partition size is $O(\frac{k}{\log n})$.*

More recently, Draganić and Krivelevich [17] obtained a tighter bound of $(1 + o(1))d/\ln d$ for CDS partitions in d-regular pseudorandom graphs.

Our contribution

The core result of our work is an efficient version of the Győri–Lovász theorem that is based on the existence of a CDS partition of size k in the given graph, which strengthens the original requirement of k-connectivity.

▶ **Theorem 4.** *There is an algorithm that, given an instance of GL-Partition together with a CDS partition of the graph of size k, computes a GL-partition in polynomial time.*

However, on its own Theorem 4 would have a notable limitation, that computing an adequate CDS partition may in itself be an obstacle – as opposed to connectivity, which can always be computed exactly in polynomial time. Fortunately, it turns out that a sufficiently large CDS partition can be found efficiently in a range of scenarios. As the first example, Theorem 2 due to Censor-Hillel et al. [10] provides such a construction for general graphs. Combining it with our algorithm, we obtain the following efficient version of the Győri–Lovász theorem, which requires higher connectivity, but puts no additional restrictions on the graph.

▶ **Corollary 5.** *GL-Partition is poly-time solvable on $\Omega(k \cdot \log^2 n)$-connected graphs.*

Furthermore, we investigate the cases where Theorem 4 can be applied to obtain a tighter connectivity gap. Following the results of Censor-Hillel, Ghaffari, and Kuhn [11], the case of general graphs cannot be improved beyond a logarithmic factor, as there exist $\Omega(k \cdot \log n)$-connected graphs that do not admit CDS partitions of size k, see Theorem 3. Therefore, putting additional restrictions on the input graph is necessary.

The first natural target here would be efficient constructions for the Győri–Lovász theorem on certain graph classes in its exact version, that is, requiring connectivity exactly k. In the light of Theorem 4, it is sufficient to show a statement of the following form, that any k-connected graph in the class $\mathcal{G}$ admits a CDS partition of size k. This would imply that the Győri–Lovász theorem admits an efficient construction on graphs in $\mathcal{G}$. We identify two such cases, biconvex graphs and interval graphs.

▶ **Theorem 6.** *Let G be a k-connected biconvex graph. A CDS partition of G of size k exists, and can be computed in polynomial time.*

▶ **Theorem 7.** *Let G be a k-connected interval graph. A CDS partition of G of size k exists, and can be computed in polynomial time.*

Coupled with the algorithm of Theorem 4, this gives an efficient construction for the Győri–Lovász theorem on biconvex graphs and interval graphs.

▶ **Corollary 8.** *GL-Partition is poly-time solvable on k-connected biconvex graphs.*

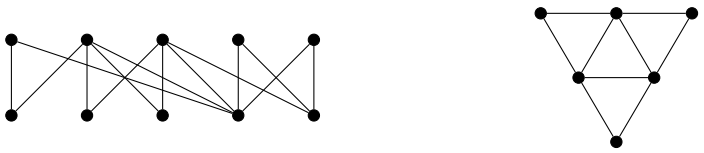


Figure 1 Examples of a 2-connected convex bipartite graph and a 2-connected chordal graph that do not admit a CDS partition of size 2.

We note that the efficient construction for chordal graphs was identified by Casel et al. [9], implying the same result for interval graphs, since interval graphs form a subclass of chordal graphs. Nonetheless, we state the algorithmic consequence of Theorem 7 next for completeness.

▶ **Corollary 9.** *GL-Partition is poly-time solvable on k-connected interval graphs.*

Unfortunately, there is a limit to the results of the form above, as already graph classes such as chordal graphs and convex bipartite graphs do not possess the property that k-connectivity implies the existence of a CDS partition of size k (see Figure 1 for examples). On the positive side, we are able to retain the productivity of Theorem 4 even in such cases. We focus on the class of convex bipartite graphs and show that $4k$-connected convex bipartite graphs always admit a CDS partition of size k.

▶ **Theorem 10.** *Let G be a $4k$-connected convex bipartite graph. A CDS partition of G of size k exists, and can be computed in polynomial time.*

This allows us to explore a trade-off between efficiency of the construction and the connectivity requirement for the class of convex bipartite graphs. As before, combined with the algorithm of Theorem 4, Theorem 10 gives us a "4-approximate" (in terms of connectivity) efficient version of the Győri–Lovász theorem on this graph class, considerably improving the connectivity requirement of the general case.

▶ **Corollary 11.** *GL-Partition is solvable in polynomial time on $4k$-connected convex bipartite graphs.*

Related work

The problem of computing the largest CDS partition of a graph was first introduced by Hedetniemi and Laskar [23], where the maximum size of a CDS partition is referred to as the connected domatic number of the graph. A number of bounds on the size of a CDS partition are due to Zelinka [41], including the fact that it is upper-bounded by the connectivity of the graph. Connected dominating sets have many practical applications for tasks such as routing and load-balancing in networks, for a detailed overview we refer to a survey due to Yu et al. [40]. Connections between CDS partitions and graph rigidity were explored by Krivelevich, Lew and Michaeli [26].

Chordal graphs are graphs that do not admit induced cycles of length more than 3. The class of chordal graphs is well-studied, and is known to possess many useful properties; in particular, many problems that are NP-hard on general graphs turn out to be polynomial-time solvable on chordal graphs, such as computing chromatic number, independence number, treewidth [34]. Interval graphs are intersection graphs of intervals on a real line, and form a subclass of chordal graphs [27].

Convex bipartite graphs[1] were introduced in 1967 by Glover [20] as they naturally arise in various industrial and scheduling applications. Formally, a bipartite graph $G = (A \cup B, E)$ is called *convex* if there exists an ordering σ of the vertices in A such that the neighborhood of any vertex of B is consecutive under σ. Restricting to the class of convex bipartite graphs has been proven fruitful for many problems such as maximum independent set, minimum feedback set, (dynamic) maximum matching and independent dominating set [35, 8, 28, 25, 36, 15]. On the other hand, several natural problems are known to be NP-complete on convex bipartite graphs. Most relevant to our work is the example of the k-path partition problem, proven to be NP-complete on convex bipartite graphs by Asdre and Nikolopoulos [2]. Belmonte and Vatshelle [4] show that convex bipartite graphs have bounded mim-width, which provides a unified perspective on polynomial-time algorithms for various classes. Bonomo-Braberman et al. [5] generalize this perspective further to extended notions of convexity.

Bipartite graphs with an ordering satisfying the convexity condition for both parts, are called biconvex. Biconvex graphs are thus a subclass of convex bipartite graphs. Abbas and Stewart studied properties of vertex orderings in biconvex graphs, yielding in particular that the vertex ranking problem is polynomial-time solvable on biconvex graphs [1]. Apart from their structural and algorithmic interest, biconvex graphs and convex bipartite graphs have been proven useful in practice, as they model well situations such as task allocation problems [20, 25]. In terms of other well-studied graph classes, convex bipartite graphs and biconvex graphs form a subclass of chordal bipartite graphs, which have a natural analogue of the chordal property among the bipartite graphs. For a more detailed introduction to the graph classes, their properties and relations between the classes, we refer to a survey by Brandstädt, Bang Le and Spinrad [7].

Paper organization. In Section 2, we introduce preliminaries that we use throughout the paper. Section 3 is dedicated to the main case of the algorithm constructing a Győri–Lovász partition on the basis of a CDS partition, and Section 4 completes the algorithm. We conclude with an overview and selected open problems in Section 5. Due to space restrictions, the results for biconvex, interval, and convex bipartite graphs, as well as details from Section 4 and selected proofs from Section 3 are omitted and we refer the reader to the full version of the paper [33]. Such results are marked by ($\star$).

2 Preliminaries

All graphs mentioned in this paper are finite, undirected and simple. We use standard graph theory notation, for reference see the full version of the paper [33] and the book of Diestel [16]. We call a graph $G = (V, E)$ a *path* if $V = \{v_1, \dots, v_n\}$ and $E = \{v_i v_{i+1} | \text{for } i \in [n-1]\}$. We usually denote a path by P and we call its order, the *length* of P. We call a graph $G = (V, E)$ a *cycle* if $V = \{v_1, \dots, v_n\}$ and $E = \{v_i v_{i+1}, v_1 v_n | \text{for } i \in [n-1]\}$. We call a graph $G = (V, E)$ a *tree* if it does not contain any cycle as a subgraph. Throughout this paper a vertex set of a tree is usually denoted by T, while its edge set is denoted by $E(T)$. Given two graphs H, G we say that G is *H-free* if G does not contain H as an induced subgraph. Given two graphs G and T, we say that T is a *spanning tree* of G if $V(G) = V(T)$ and $G[V(T)]$ induces a tree.

Given two vertex sets $A, B \in V(G)$ we say that A is *dominating* to B if every vertex in B is adjacent to some vertex in A. Also, when we say that a vertex set is dominating to some graph G, we mean that it is dominating to its vertex set $V(G)$. A vertex set A is called

[1] Often referred to simply as *convex graphs* in the literature.

a *connected dominating set* (CDS) if and only if it is a dominating set and $G[A]$ is connected. Given a graph G, a collection of vertex sets $\mathcal{V} = \{V_1, \ldots, V_l\}$ such that $V_i \subseteq V(G)$ is called a *connected dominating set partition* (CDS partition) if and only if the sets in $\mathcal{V}$ are pairwise disjoint and $\cup_{U \in \mathcal{V}} U = V(G)$, and each of them is a connected dominating set of G. We call $|\mathcal{V}|$ the *size* of the CDS partition $\mathcal{V}$. Throughout the paper, we also use an equivalent notion of vertex-disjoint dominating trees. Given a CDS partition of a graph it is immediate to get a set of vertex-disjoint dominating trees of the same size by considering the spanning tree of each set in the CDS partition. Moreover, given k disjoint dominating trees, one may obtain a CDS partition of size k by simply adding the remaining vertices arbitrarily to those trees.

3 From Simple CDS Partition to GL-partition

Before we move on to the general algorithm of Theorem 4 that computes a GL-partition from a CDS partition of size k, we cover in this section the main subroutine of this algorithm. Specifically, we present an algorithm (Algorithm 3) that, given as input an instance of GL-Partition and k dominating trees such that all terminals are contained in the same dominating tree, returns a partial GL-partition with l sets, for some $l \in [k]$, such that the remaining vertices admit a CDS partition of size $k - l$. The general algorithm, described in the next section, then uses this result as a subprocedure. We first describe the intuition behind the main steps of Algorithm 3, then provide a more detailed description of these steps, and finally define a formal procedure for each step and show its correctness.

Intuition of Algorithm 3. Let T_1 be the dominating tree that contains k terminal vertices. First we add all the vertices of T_1 to one of the k vertex sets while maintaining the connectivity of each set. There exist $k - 1$ dominating trees that have no terminal vertex. Now, we map these k vertex sets to one of these $k - 1$ dominating trees. We aim to fulfill the vertex set to its demand while using the corresponding dominating tree.

We use an iterative procedure that first assigns (but does not add) neighbors for each vertex set to these $k - 1$ dominating trees, and then we add assigned vertices to each vertex set whose demand is not met by the assignment. Now, to fulfill the demand of each such vertex set, we use its corresponding dominating tree while preserving the connectivity of the vertex set. At the end of this procedure, each part is either full or contains a whole dominating tree (apart from the one that was not assigned to any tree). The remaining vertices are arbitrarily added to the non-full sets, as each contains a dominating tree, apart from the one that has not been assigned a tree, for which we have ensured that there are enough adjacent vertices left.

Description of Algorithm 3. The algorithm receives an instance of GL-Partition, (G, k, C, N), where $C = \{c_1, \ldots, c_k\}$, $N = \{n_1, \ldots, n_k\}$ and a set of dominating trees $\mathcal{T} = \{T_1, \ldots, T_k\}$ as an input such that such that G is k-connected graph and $C \subseteq T_1$.

It is important to note that during this description of the algorithm, every time we add a vertex to a vertex set, it is implied that we also check whether the condition of having l full sets say $V_1, \ldots, V_l$, intersecting exactly l of the dominating trees, say $T_1, \ldots, T_l$ is met. If yes, then we output those l full sets and return an instance of the original problem, $G - \cup_{i=1}^{l} V_i$ and $k - l$, $\mathcal{T} = \mathcal{T} \setminus \{T_{k-l}, \ldots, T_k\}$, $C = C \setminus \{c_{k-l}, \ldots, c_k\}$ and $N = N \setminus \{n_{k-l}, \ldots, n_k\}$. We moreover assume that every time we add a vertex we also check if a set meets its size demand and if so we add it to the set Full that includes all the connected vertex sets that satisfy their size demands. We denote this procedure of adding a vertex u to set V by Add(V, u)

(similarly Add(V, U) when we have a vertex set U instead of singleton $\{u\}$). Similarly every time we remove a vertex from some set we check if it is in Full and if yes, we remove the set from Full. We denote this procedure of removing vertex u from the set V by Remove(V, u).

Step 1 We initialize each set V_i to contain the corresponding terminal vertex c_i for all $i \in [k]$ and then add the vertices of $T_1 \setminus C$ one by one to a vertex set that it is adjacent to (based only on the adjacencies occurring by T_1). Next, we add the vertices of $V(G) \setminus V(\mathcal{T})$ to an adjacent set one by one.

Step 2 Each vertex of $\mathcal{T} \setminus T_1$ is assigned through a function Vlabel to an adjacent vertex set. Depending on whether the amount of vertices assigned to some vertex set is less than the vertices needed to be added for the set to satisfy its demand or not, the set is categorized in the set Under or Over respectively (Algorithm 1).

Step 3 Through the function Tlabel each set in Under is assigned one of the trees in $\mathcal{T} \setminus T_1$. Then, one vertex of each such tree adjacent to the corresponding set is added. Priority is given to the vertices assigned to this set through the function Vlabel. However, if no such vertex exists, another adjacent vertex is arbitrarily chosen to be added, and we change the value of this vertex in the function Vlabel accordingly. If this results in a set moving from Over to Under, we again assign a tree to this set (that has not been assigned to any other set) through the function Tlabel (Algorithm 1).

Step 4 For each tree assigned to some set, we consider its vertices not belonging in that set one by one and add them to that set as long as the set connectivity is preserved. Again, if, during this procedure, a set moves from Over to Under, go to the previous Step. Repeat this procedure until each set in Under is either full or contains a whole tree (Algorithm 2).

Step 5 Add each of the remaining vertices, the ones not added to any set, to an adjacent non full set one by one (Algorithm 2).

Step 6 Output the k sets created.

As Step 1 works in a straightforward way, we do not provide a pseudocode for it. We describe in the proof of the lemma below how this can be done in polynomial time. In the main algorithm, we refer to this procedure as AddTrees$(G, C, \mathcal{T})$ that returns the sets $V_1, \ldots, V_k$.

▶ **Lemma 12** ($\star$). *Given an instance of GL-Partition (G, k, C, N) and k vertex disjoint dominating trees $T_1, \ldots, T_k$ such that $C \subseteq T_1$, we construct k vertex connected sets $V_1, \ldots, V_k$ such that $c_i \in V_i$ for all $i \in [k]$ and $T_1 \subseteq \cup_{i=1}^{k} V_i$, in polynomial time.*

Algorithm 1 realizes the two next steps of Algorithm 3. In particular, first each remaining vertex of the graph is assigned to some set through Vlabel and then each set is categorized in Over or Under depending on the number of vertices assigned to it. Then, for each set in Under the assigned vertices are added and through Tlabel we assign a distinct tree each such set. Afterwards we ensure that each set in Under contains a vertex from its assigned tree.

It is easy to notice that Step 2 runs in polynomial time. In order to prove the correctness of Step 3 in Lemma 13, it suffices to show that there are at most $k - 1$ sets in Under and this property is not affected while ensuring that each vertex set contains a vertex from the corresponding through Tlabel tree. This part of the algorithm also runs in polynomial time because each set in Under is considered at most once and a set from Over might move to Under but once a set is in Under it will never move to Over.

▶ **Lemma 13.** *Let (G, k, C, N) be an instance of GL-Partition, where $C = \{c_1, \ldots, c_k\}$, $\{V_1, \ldots, V_k\}$ are k connected vertex sets such that $c_i \in V_i$ for all $i \in [k]$, and a set of k vertex disjoint dominating trees $\mathcal{T} = \{T_1, \ldots, T_k\}$ such that $C \subseteq T_1$, are given as input to Algorithm 1. Then, after the loop in line 17 (Step 3) each vertex set is either in* Over*, or is assigned to a tree through* Tlabel *and contains a vertex from that tree. Moreover, $G[V_i \cup \text{Vlabel}(V_i)]$ is connected for each $i \in [k]$ and at least one set is in* Over*.*

Proof. First notice that $G[V_i \cup \mathsf{Vlabel}(V_i)]$ is connected for all $i \in [k]$, by the definition of the function Vlabel. We now show that there is at least one vertex set in Over. Assume that there is no set in Over and hence $n_i - |V_i| > |\mathsf{Vlabel}(V_i)|$ for all $i \in [k]$. Then $\sum_{i=1}^{k} n_i - |V_i| > \sum_{i=1}^{k} |\mathsf{Vlabel}(V_i)| \Rightarrow n > \sum_{i=1}^{k} |\mathsf{Vlabel}(V_i)| + \sum_{i=1}^{k} |V_i| = n$ which is a contradiction. This implies that regardless of whether a set moves from Over to Under, as long as all vertices of G are assigned to the sets $V_1, \ldots, V_k$, one of these sets will be in Over.

In the loop in line 17, we consider each tree assigned to some vertex set and first check whether this vertex set already contains some vertex of the tree. If yes, our lemma is satisfied. Otherwise, since the tree is dominating, there exists at least one vertex adjacent to the part of the vertex set intersecting T_1. This vertex is already added to some vertex set in Under or assigned to one in Over. If it belongs in some set in Over, then we simply add it to the vertex set. Then we remove this vertex from the set that occurs from the assignment Vlabel for the set previously in Over, and we check to see if it now meets the condition for this set belonging to Under. If yes, we go to line 12 to deal with this set similarly. Repeating this procedure for the other sets in Under does not affect them as every vertex set currently in Under already contains all the vertices assigned to it through the function Vlabel. Moreover, since in this step, we add a vertex to a vertex set, we also check if this set becomes full and intersects exactly one dominating tree. If that is true, we restart the algorithm for the adjusted values of k, G, $\mathcal{T}$, and $\mathcal{C}$.

Otherwise, if the adjacent vertex is part of some other set in Under, we add it to our current set and remove it from the previous one (again checking if the condition for restarting the algorithm is met). Moreover, we remove it from the set created through the function Vlabel to avoid this vertex from existing in two sets (in case we go to line 12) in some other iteration of this loop. Note that sets initially in Under might be added in Full or removed from it depending on whether we add or remove vertices from them, but they will never go from Under to Over as we need to maintain the property that each set in Over contains no vertices from $T_2, \ldots, T_k$. ◀

Algorithm 2 realizes essentially the last two steps of the algorithm and either returns the desired partition of size k or a smaller instance of a more general problem. First we ensure that each set in Under is either full or contains the dominating tree assigned to it through Tlabel. This is done by considering each set in *Under* separately and then adding the neighboring vertices of the corresponding tree one by one. If during this procedure we "steal" a vertex assigned in some set in Over and this causes it to move to Under we run Algorithm 1 starting from line 13 (only running the loop of line 12 once, for the set that moved to Under). This happens at most once for each set hence this part of Algorithm 2 runs in polynomial time. In the last part of this algorithm we simply add the remaining vertices to some adjacent non full set, first based on their assignment through Vlabel and then based on the adjacencies in G.

▶ **Lemma 14.** *Let an instance of GL-Partition $(G, k, C = \{c_1, \ldots, c_k\}, N)$, k connected vertex sets $\{V_1, \ldots, V_k\}$ such that $|V_i| < n_i$ and $c_i \in V_i$ for all $i \in [k]$, k dominating to G trees $\{T_1, \ldots, T_k\}$ and a function* Vlabel $: \{V_1, \ldots, V_k\} \to P(V(G))$*, are given as an input to Algorithm 2. Then, after the loop in line 1 (Step 5), each vertex set is connected and is either full or completely contains a dominating tree or is assigned through* Vlabel *to at least as many vertices as its remaining size demand that do not belong in any other vertex set.*

Proof. In order to see that the loop in line 1 terminates, it suffices to notice that it only considers sets in Under and the value of $\sum_{i=2}^{k} |T_i' \cap \mathsf{Tlabel}^{-1}(T_1)|$.

Algorithm 1 Labeling (Step 2, Step 3).

Require: An instance of GL-Partition $(G, k, C = \{c_1, \ldots, c_k\}, N = \{n_1, \ldots, n_k\})$, a set of k vertex disjoint dominating trees $\mathcal{T} = \{T_1, \ldots, T_k\}$ such that $C \subseteq T_1$, and k connected vertex sets $\{V_1, \ldots, V_k\}$ such that $c_i \in V_i$ for all $i \in [k]$

1: Initialize $\mathsf{Vlabel} : \{V_1, \ldots, V_k\} \to P(V(G))$ and $\mathsf{Tlabel} : \{V_1 \ldots, V_k\} \to \mathcal{T}$ such that $\mathsf{Vlabel}(V_i) \leftarrow \emptyset$, $\mathsf{Tlabel}(V_i) \leftarrow \emptyset$, $\forall i \in [k]$.
2: $\mathsf{Over} \leftarrow \emptyset$, $\mathsf{Under} \leftarrow \emptyset$
3: **for** $i \leftarrow 2$ to k **do**
4: **for** every vertex $v \in T_i$ **do** ▷ Step 2
5: Let $V_j \in \{V_1, \ldots, V_k\}$ such that $v \in N_G(V_j \cap T_1)$
6: $\mathsf{Vlabel}(V_j) \leftarrow \mathsf{Vlabel}(V_j) \cup v$
7: **for** $i \leftarrow 2$ to k **do**
8: **if** $n_i - |V_i| \leq |\mathsf{Vlabel}(V_i)|$ **then**
9: $\mathsf{Over} \leftarrow \mathsf{Over} \cup V_i$
10: **else**
11: $\mathsf{Under} \leftarrow \mathsf{Under} \cup V_i$
12: **for** every $V_i \in \mathsf{Under}$ **do** ▷ Step 3
13: $V_i \leftarrow \mathrm{Add}(V_i, \mathsf{Vlabel}(V_i))$
14: **if** $\mathsf{Tlabel}(V_i) = \emptyset$ **then**
15: Let T_j be a tree such that $T_j \notin \cup_{s=1}^{k} \mathsf{Tlabel}(V_s)$
16: $\mathsf{Tlabel}(V_i) \leftarrow T_j$
17: **for** every tree $T_i \in \mathcal{T} \setminus T_1$ such that $\mathsf{Tlabel}^{-1}(T_i) \neq \emptyset$ **do**
18: Let V_j be the vertex set such that $\mathsf{Tlabel}(V_j) = T_i$.
19: **if** $V_j \cap T_i = \emptyset$ **then**
20: Let $u \in T_i$ such that $u \in N(V_j \cap T_1)$ and V_s such that $u \in \mathsf{Vlabel}(V_s)$
21: **if** $V_s \in \mathsf{Over}$ **then**
22: $V_j \leftarrow \mathrm{Add}(V_j, u)$, $\mathsf{Vlabel}(V_s) \leftarrow \mathrm{Remove}(\mathsf{Vlabel}(V_s), u)$.
23: **if** $n_s - |V_s| > |\mathsf{Vlabel}(V_s)|$ **then**
24: $\mathsf{Over} \leftarrow \mathsf{Over} \setminus V_s$, $\mathsf{Under} \leftarrow \mathsf{Under} \cup V_s$
25: Go to line 12
26: **else**
27: $V_s \leftarrow \mathrm{Remove}(V_s, u)$, $\mathsf{Vlabel}(V_s) \leftarrow \mathsf{Vlabel}(V_s) \setminus u$, $V_j \leftarrow \mathrm{Add}(V_j, u)$
28: **return** $\mathsf{Over}, \mathsf{Under}, \mathsf{Vlabel}, \mathsf{Tlabel}$

We start by considering the vertex set V_j such that $\mathsf{Tlabel}(V_j) = T_i$ for some value of i. Due to Lemma 13, we know that T_i is either completely contained in V_j, in which case we do nothing and we proceed to consider the next tree, or it has a vertex of T_i that is adjacent to $V_j \cap T_1$ and not in V_j. We then add that vertex to V_j in the same manner as in the loop in line 17 of Algorithm 1 depending on whether this vertex belongs already in some set or not.

Notice that this action preserves the connectivity of the set V_j due to the way the vertex is chosen and also does not affect the connectivity of other sets as this is based only on the adjacencies of the parts of each vertex set intersecting T_1, which is not affected. Moreover it increases the value of $\sum_{i=2}^{k} |T_i \cap \mathsf{Tlabel}^{-1}(T_1)|$ by 1, which ensures that this loop terminates. Again, every time we add a vertex to a vertex set, it is implied that we also check if that set should meets its size demand and should be added to Full and also if the condition to terminate and output l full sets and a smaller instance of the general problem is met. ◀

Algorithm 2 Add Vertices (Step 4, Step 5).

Require: An instance of GL-Partition $(G, k, C = \{c_1, \ldots, c_k\}, N = \{n_1, \ldots, n_k\})$, a set of k vertex disjoint dominating trees $\mathcal{T} = \{T_1, \ldots, T_k\}$ such that $C \subseteq T_1$, k connected vertex sets $\{V_1, \ldots, V_k\}$ such that $c_i \in V_i$ for all $i \in [k]$, sets Under, Over, Full, and functions Vlabel, Tlabel.

1: **while** $\exists V_j \in$ Under such that $V_j \notin$ Full and $T_i \not\subseteq V_j$ $\forall i \in \{2, \ldots, k\}$ **do** ▷ Step 4
2: Let T_i be the tree such that Tlabel$(V_j) = T_i$
3: **while** V_j doesn't contain vertices only from T_i and $T_i \not\subseteq V_j$ and $V_j \notin$ Full **do**
4: Let $v \in T_i$ such that $v \in N(V_j \cap (T_1 \cup T_i))$
5: **if** v is assigned through the function Vlabel to some set $V_s \in$ Over **then**
6: $V_j \leftarrow$ Add(V_j, v), Vlabel$(V_s) \leftarrow$ Vlabel$(V_s) \setminus v$
7: **if** $n_s - |V_s| >$ |Vlabel(V_s)| **then**
8: Over $\leftarrow$ Over $\setminus V_s$, Under $\leftarrow$ Under $\cup V_s$
9: Run Algorithm 1 with input $(G, \{T_1, \ldots, T_k\}, C,$ Under, Over, Vlabel, $N)$ starting from line 13.
10: **else**
11: $V_s \leftarrow$ Remove(V_s, v), Vlabel$(V_s) \leftarrow$ Vlabel$(V_s) \setminus v$, $V_j \leftarrow$ Add(V_j, v)
12: **for** every $V \in$ Over and $V \notin$ Full **do** ▷ Step 5
13: Let u be a vertex in Vlabel(V) and not in V
14: $V \leftarrow$ Add(V, u)
15: **for** every $v \in \cup_{i=2}^{k} T_i \setminus \cup_{i=1}^{k} V_i$ **do**
16: Let V be a set such that $V \notin$ Full and $v \in N(V)$
17: $V \leftarrow$ Add(V, v)
18: **return** $\{V_1, \ldots, V_k\}$

Now that we have proved the correctness of Step 4, notice that in the last part of Algorithm 2 each non full set either contains a whole dominating tree or is adjacent to at least as many available vertices as it needs for its size demand to be met. We first start by the second case in which we add the assigned vertices to the set one by one until it is full. Then we are left with the case where each non full set contains a dominating tree, in which we can greedily add adjacent vertices to each such set one by one until all of them meet their size demands.

We now put everything together and provide the main algorithm of this section.

Algorithm 3 Győri–Lovász theorem for special case of graphs with k dominating trees.

Require: An instance of GL-Partition (G, k, C, N) and a set of k vertex disjoint dominating trees $\mathcal{T} = \{T_1, \ldots, T_k\}$ such that $C \subseteq T_1$.

1: $(V_1, \ldots, V_k) \leftarrow$ AddTrees$(G, C, \mathcal{T})$
2: (Over, Under, Vlabel, Tlabel) $\leftarrow$ Labeling$(G, C, \mathcal{T}, N, V_1, \ldots, V_k)$
3: $(V_1, \ldots, V_k) \leftarrow$ Add Vertices$(G, C, \mathcal{T}, N,$ Over, Under, Vlabel, Tlabel, Full)
4: Output $V_1, \ldots, V_k$

▶ **Lemma 15** (⋆). *Let an instance of GL-Partition $(G, k, C = \{c_1, \ldots, c_k\}, N = \{n_1, \ldots, n_k\})$ and k vertex disjoint dominating trees $T_1, \ldots, T_k$ such that $C \subseteq T_1$ are given as an input to Algorithm 3. Then Algorithm 3 returns l full sets $V_1, \ldots, V_l$ and a reduced instance, $(G - \cup_{i=1}^{l} V_i, k - l, C \setminus \{c_{k-l}, \ldots, c_k\}, N \setminus \{n_{k-l}, \ldots, n_k\})$ of GL-Partition with a set of vertex-disjoint dominating trees of reduced graph $\mathcal{T} \setminus \{T_{k-l}, \ldots, T_k\}$, for some $l \in [k]$, in polynomial time.*

In the next section we tackle the more general instance in which there is a dominating tree that contains strictly less than k terminals and some other contains at least one by showing that we are able to iteratively find a smaller instance in which all the terminals are contained in some dominating tree and then employ the algorithm developed in this section.

4 From General CDS Partition to GL-partition

In this section we provide the algorithm from Theorem 4. The algorithm receives an instance of GL-Partition (G, k, C, N) and k vertex disjoint dominating trees $T_1, \ldots, T_k$ as input and it outputs a GL-partition $V_1, \ldots, V_k$ of $V(G)$.

If all terminal vertices are contained in one dominating tree then we can directly use Algorithm 3 of the previous section. Notice that if a terminal is not contained in any tree we can simply add it to one since each tree is dominating. Hence, we assume without loss of generality that every terminal vertex is contained in some tree. Moreover, we assume that the dominating trees are separated into three sets $\mathcal{T}_0, \mathcal{T}_1$ and $\mathcal{T}_{>1}$ depending on whether they contain 0, 1 or more than 1 terminal vertices, respectively. This categorization can be done in polynomial time so for simplicity we assume that this is also part of the input. As in the previous section we first provide an intuitive description of Algorithm 4. The formal descriptions of the steps and the proof of correctness is deferred to the appendix.

Description of Algorithm 4. The Algorithm receives as input a GL-Partition instance (G, k, C, N), where $C = \{c_1, \ldots, c_k\}$, $N = \{n_1, \ldots, n_k\}$, and a set of k vertex-disjoint dominating trees $\mathcal{T} = \{T_1, \ldots, T_k\}$ separated into three sets $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_{>1}$ depending on how many terminal vertices they contain. It is important to note that, like in the previous section, during this description of the algorithm, every time we add a vertex to a vertex set, it is implied that we also check if the condition of having l full sets say $V_1, \ldots, V_l$, intersecting exactly l of the dominating trees, say $T_1, \ldots, T_l$ is met. If yes, then we output those l full sets and restart the algorithm for $G - \cup_{i=1}^{l} V_i$ and $k - l$, $\mathcal{T} = \mathcal{T} \setminus \{T_{k-l}, \ldots, T_k\}$, $C = \{c_{k-l}, \ldots, c_l\}$ and $N = \{n_{k-l}, \ldots, n_k\}$. For simplicity and space constraints, we do not include this in every step of this description nor in the pseudocodes provided in this section. We moreover assume both in pseudocodes of this section and in this description that every time we add a vertex we also check if a set meets its size demand and if so we add it to the set Full that includes all the connected vertex sets that satisfy their size demands. Similarly every time we remove a vertex from some set we check if it is in Full and if yes, we remove it from Full.

Step 1 We first initialize each vertex set V_i to contain the corresponding terminal vertex c_i, where $i \in [k]$. Then, for each tree in $\mathcal{T}_1 \cup \mathcal{T}_{>1}$, we add its vertices one by one to a vertex set that it is adjacent to (based only on the adjacencies occurring only by the tree considered). Next, we add the vertices of $V(G) \setminus V(\mathcal{T})$ to an adjacent set one by one.

Step 2 For each tree $T \in \mathcal{T}_{>1}$ that contains l terminals, we create a tree set $\mathcal{T}'$ containing T and $l - 1$ trees from $\mathcal{T}_0$, which we then remove from $\mathcal{T}_{>1}$ and $\mathcal{T}_0$ respectively. Among those sets, we choose one that has enough vertices available to satisfy the size demands of all of the l vertex sets intersecting T. (Algorithm Choose a Tree Set)

Step 3 Call Algorithm 3 that has as input those l dominating trees, the corresponding vertex sets and terminal vertices, while the graph we work on is the one that occurs from the union of those sets and trees. Notice that the sum of the demands of those sets might be less than the size of the graph but this does not affect the correctness of the algorithm.

Algorithm 4 Győri–Lovász theorem for graphs with k dominating trees.

Require: A graph G, three sets of dominating trees $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_{>1}$, a set of k vertices $C = \{c_1, \ldots, c_k\}$, a set of k natural numbers $N = \{n_1, \ldots, n_k\}$ and k connected vertex sets $\{V_1, \ldots, V_k\}$ such that $c_i \in V_i$ for all $i \in [k]$

1: $(V_1, \ldots, V_k) \leftarrow$ Add Vertices$(\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_{>1}, V_1 \ldots, V_k, C, N, U)$
2: $(T, T_1, \ldots, T_{l-1}) \leftarrow$ Choose a Tree Set$(\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_{>1}, C, N, V_1, \ldots, V_k)$
3: Algorithm 3$(\cup_{i=1}^{l} V_i \cup_{i=1}^{l-1} T'_i \cup T, T, T'_1, \ldots, T'_{l-1}, c_1, \ldots, c_l, n_1, \ldots, n_l)$.

5 Conclusion

As we showcase in this work, the algorithm of Theorem 4 opens up a new avenue for efficiently computing connected partitions, in the settings where sufficiently large CDS partitions are available. This motivates further research into improving bounds for CDS partitions beyond the worst-case instances, as well as into parameterized and approximation algorithms for computing a CDS partition of the largest size.

To the best of our knowledge, Corollaries 5 and 11 are the first examples of a Győri–Lovász type of results that allow for a trade-off between efficiency of the construction and the connectivity requirement. It would be interesting to investigate whether a more refined trade-off is possible, interpolating smoothly between the polynomial time for higher connectivity and exponential time for extremal connectivity.

Our work also initiates the study of the gap between connectivity and the size of a CDS packing in structured graph classes. It remains a curious open question to characterize the class of graphs, for which k-connectivity implies the existence of a k-CDS partition. So far, our results show that this class contains interval and biconvex graphs (Theorems 7 and 6), but does not contain chordal and convex bipartite graphs.

References

1 Nesrine Abbas and Lorna K. Stewart. Biconvex graphs: ordering and algorithms. *Discrete Applied Mathematics*, 103(1):1–19, 2000. doi:10.1016/S0166-218X(99)00217-6.

2 Katerina Asdre and Stavros D. Nikolopoulos. Np-completeness results for some problems on subclasses of bipartite and chordal graphs. *Theoretical Computer Science*, 381(1):248–259, 2007. doi:10.1016/j.tcs.2007.05.012.

3 Ronald I. Becker, Isabella Lari, Mario Lucertini, and Bruno Simeone. Max-min partitioning of grid graphs into connected components. *Networks*, 32(2):115–125, 1998. doi:10.1002/(SICI)1097-0037(199809)32:2\%3C115::AID-NET4\%3E3.0.CO;2-E.

4 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013. Exact and Parameterized Computation. doi:10.1016/j.tcs.2013.01.011.

5 Flavia Bonomo-Braberman, Nick Brettell, Andrea Munaro, and Daniël Paulusma. Solving problems on generalized convex graphs via mim-width. *Journal of Computer and System Sciences*, 140:103493, 2024. doi:10.1016/j.jcss.2023.103493.

6 Ralf Borndörfer, Katrin Casel, Davis Issac, Aikaterini Niklanovits, Stephan Schwartz, and Ziena Zeif. Connected k-partition of k-connected graphs and c-claw-free graphs. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.APPROX/RANDOM.2021.27.

7 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999. doi:10.1137/1.9780898719796.

8 Gerth Stølting Brodal, Loukas Georgiadis, Kristoffer Arnsfelt Hansen, and Irit Katriel. Dynamic matchings in convex bipartite graphs. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 406–417. Springer, 2007. doi:10.1007/978-3-540-74456-6_37.

9 Katrin Casel, Tobias Friedrich, Davis Issac, Aikaterini Niklanovits, and Ziena Zeif. Efficient constructions for the Győri-Lovász theorem on almost chordal graphs. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science - 49th International Workshop, WG 2023, Fribourg, Switzerland, June 28-30, 2023, Revised Selected Papers*, volume 14093 of *Lecture Notes in Computer Science*, pages 143–156. Springer, 2023. doi:10.1007/978-3-031-43380-1_11.

10 Keren Censor-Hillel, Mohsen Ghaffari, George Giakkoupis, Bernhard Haeupler, and Fabian Kuhn. Tight bounds on vertex connectivity under sampling. *ACM Trans. Algorithms*, 13(2), May 2017. doi:10.1145/3086465.

11 Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. A new perspective on vertex connectivity. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 546–561. SIAM, 2014. doi:10.1137/1.9781611973402.41.

12 L. Sunil Chandran, Yun Kuen Cheung, and Davis Issac. Spanning tree congestion and computation of generalized győri-lovász partition. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 32:1–32:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.32.

13 Jiangzhuo Chen, Robert D. Kleinberg, László Lovász, Rajmohan Rajaraman, Ravi Sundaram, and Adrian Vetta. (almost) tight bounds and existence theorems for single-commodity confluent flows. *J. ACM*, 54(4):16, 2007. doi:10.1145/1255443.1255444.

14 Janka Chlebíková. Approximating the maximally balanced connected partition problem in graphs. *Information Processing Letters*, 60(5):225–230, 1996. doi:10.1016/S0020-0190(96)00175-5.

15 Peter Damaschke, Haiko Müller, and Dieter Kratsch. Domination in convex and chordal bipartite graphs. *Information Processing Letters*, 36(5):231–236, 1990. doi:10.1016/0020-0190(90)90147-P.

16 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

17 Nemanja Draganić and Michael Krivelevich. Disjoint connected dominating sets in pseudorandom graphs, 2024. arXiv:2410.16072.

18 Wenfei Fan, Wenyuan Yu, Jingbo Xu, Jingren Zhou, Xiaojian Luo, Qiang Yin, Ping Lu, Yang Cao, and Ruiqi Xu. Parallelizing sequential graph computations. *ACM Trans. Database Syst.*, 43(4):18:1–18:39, 2018. doi:10.1145/3282488.

19 Michael R. Garey and David S. Johnson. A note on "A note on 'Some simplified NP-complete graph problems' ". *SIGACT News*, 9(4):17, 1978. doi:10.1145/1008369.1008371.

20 Fred Glover. Maximum matching in a convex bipartite graph. *Naval research logistics quarterly*, 14(3):313–316, 1967.

21 Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In Chandu Thekkath and Amin Vahdat, editors, *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 17–30. USENIX Association, 2012. URL: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/gonzalez.

22 Ervin Győri. On division of graphs to connected subgraphs. *Colloq. Math. Soc. Janos Bolyai*, 18:485–494, 1976.

23 S. T. Hedetniemi and R. Laskar. Connected domination in graphs. Technical report, Technical Report 414, Clemson University, SC, USA, 1983.

24 Alexander Hoyer. On the independent spanning tree conjectures and related problems. *PhD thesis, Georgia Institute of Technology*, 2016.

25 Witold Lipski Jr. and Franco P. Preparata. Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Informatica*, 15:329–346, 1981. doi:10.1007/BF00264533.

26 Michael Krivelevich, Alan Lew, and Peleg Michaeli. Rigid partitions: from high connectivity to random graphs, 2023. arXiv:2311.14451.

27 C. Lekkeikerker and J. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962. URL: http://eudml.org/doc/213681.

28 Y. Daniel Liang and Maw-Shang Chang. Minimum feedback vertex sets in cocomparability graphs and convex bipartite graphs. *Acta Informatica*, 34(5):337–346, 1997. doi:10.1007/S002360050088.

29 Lázló Lovász. A homology theory for spanning trees of a graph. *Acta Math. Acad. Sci. Hungaricae*, 30(3-4):241–251, 1977.

30 Mario Lucertini, Yehoshua Perl, and Bruno Simeone. Most uniform path partitioning and its use in image processing. *Discret. Appl. Math.*, 42(2):227–256, 1993. doi:10.1016/0166-218X(93)90048-S.

31 Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning graphs to speedup dijkstra's algorithm. *ACM J. Exp. Algorithmics*, 11, 2006. doi:10.1145/1187436.1216585.

32 Shin-Ichi Nakano, Md. Saidur Rahman, and Takao Nishizeki. A linear-time algorithm for four-partitioning four-connected planar graphs. *Inf. Process. Lett.*, 62(6):315–322, 1997. doi:10.1016/S0020-0190(97)00083-5.

33 Aikaterini Niklanovits, Kirill Simonov, Shaily Verma, and Ziena Zeif. Connected partitions via connected dominating sets, 2025. doi:10.48550/arXiv.2503.13112.

34 Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976. doi:10.1137/0205021.

35 José Soares and Marco Aurelio Stefanes. Algorithms for maximum independent set in convex bipartite graphs. *Algorithmica*, 53(1):35–49, 2009. doi:10.1007/S00453-007-9006-9.

36 George Steiner and Julian Scott Yeomans. A linear time algorithm for maximum matchings in convex, bipartite graphs. *Computers & Mathematics with Applications*, 31(12):91–96, 1996.

37 Hitoshi Suzuki, Naomi Takahashi, and Takao Nishizeki. A linear algorithm for bipartition of biconnected graphs. *Inf. Process. Lett.*, 33(5):227–231, 1990. doi:10.1016/0020-0190(90)90189-5.

38 Hitoshi Suzuki, Naomi Takahashi, Takao Nishizeki, H Miyano, and S Ueno. An algorithm for tripartitioning 3-connected graphs. *Journal of Information Processing Society of Japan*, 31(5):584–592, 1990.

39 Koichi Wada and Kimio Kawaguchi. Efficient algorithms for tripartitioning triconnected graphs and 3-edge-connected graphs. In Jan van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science, 19th International Workshop, WG '93, Utrecht, The Netherlands, June 16-18, 1993, Proceedings*, volume 790 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 1993. doi:10.1007/3-540-57899-4_47.

40 Jiguo Yu, Nannan Wang, Guanghui Wang, and Dongxiao Yu. Connected dominating sets in wireless ad hoc and sensor networks – a comprehensive survey. *Computer Communications*, 36(2):121–134, 2013. doi:10.1016/j.comcom.2012.10.005.

41 Bohdan Zelinka. Connected domatic number of a graph. *Mathematica Slovaca*, 36(4):387–392, 1986. URL: http://eudml.org/doc/34243.

Courcelle's Theorem for Lipschitz Continuity

Tatsuya Gima
Hokkaido University, Sapporo, Japan

Soh Kumabe
CyberAgent, Tokyo, Japan

Yuichi Yoshida
National Institute of Informatics, Tokyo, Japan

Abstract

Lipschitz continuity of algorithms, introduced by Kumabe and Yoshida (FOCS'23), measures the stability of an algorithm against small input perturbations. Algorithms with small Lipschitz continuity are desirable, as they ensure reliable decision-making and reproducible scientific research. Several studies have proposed Lipschitz continuous algorithms for various combinatorial optimization problems, but these algorithms are problem-specific, requiring a separate design for each problem.

To address this issue, we provide the first algorithmic meta-theorem in the field of Lipschitz continuous algorithms. Our result can be seen as a Lipschitz continuous analogue of Courcelle's theorem, which offers Lipschitz continuous algorithms for problems on bounded-treewidth graphs. Specifically, we consider the problem of finding a vertex set in a graph that maximizes or minimizes the total weight, subject to constraints expressed in monadic second-order logic (MSO_2). We show that for any $\varepsilon > 0$, there exists a $(1 \pm \varepsilon)$-approximation algorithm for the problem with a polylogarithmic Lipschitz constant on bounded treewidth graphs. On such graphs, our result outperforms most existing Lipschitz continuous algorithms in terms of approximability and/or Lipschitz continuity. Further, we provide similar results for problems on bounded-clique-width graphs subject to constraints expressed in MSO_1. Additionally, we construct a Lipschitz continuous version of Baker's decomposition using our meta-theorem as a subroutine.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Approximation algorithms analysis

Keywords and phrases Fixed-Parameter Tractability, Algorithmic Meta-Theorem, Lipschitz Continuity

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.11

Related Version *Full Version*: http://arxiv.org/abs/2506.21118

Funding *Tatsuya Gima*: JSPS KAKENHI Grant Numbers JP24K23847, JP25K03077
Yuichi Yoshida: JSPS KAKENHI Grant Number JP24K02903

1 Introduction

Lipschitz continuity of algorithms, introduced by Kumabe and Yoshida [14], is a measure of an algorithm's stability in response to errors or small perturbations in the input for weighted optimization problems. Roughly speaking, it is the maximum ratio of the (weighted) Hamming distance between the outputs of an algorithm for two different weights to the ℓ_1 distance between those weights (see Section 2.4 for the precise definition). It is desirable for algorithms to have small Lipschitz constants, as large constants can undermine the reliability of decision-making and the reproducibility of scientific research.

Since its introduction, Lipschitz continuous algorithms have been proposed for various optimization problems [14, 16]. However, we need to design a different algorithm and analyze its Lipschitz continuity for each problem, which can be impractical. To address this limitation, we present an algorithmic meta-theorem for Lipschitz continuous algorithms, which can be seen as Lipschitz continuity analogue of celebrated Courcelle's theorem [3].


© Tatsuya Gima, Soh Kumabe, and Yuichi Yoshida;
licensed under Creative Commons License CC-BY 4.0



To present our results, we first introduce some notation. Let $G = (V, E)$ be a graph of n vertices with treewidth tw, $\varphi(X)$ be an MSO_2 formula with a free vertex set variable X (see Section 2.3 for details about logic), and $w \in \mathbb{R}_{\geq 0}^V$ be a weight vector. We consider the problem of finding a vertex subset $X \subseteq V$ such that $G \models \varphi(X)$ and $w(X)$ is maximized, which we call the *MSO_2 maximization problem*. We also consider the problem of finding X such that $w(X)$ is minimized, which we call the *MSO_2 minimization problem*. Our main results are the following.

▶ **Theorem 1.** *For any $\varepsilon \in (0, 1]$, there is a $(1-\varepsilon)$-approximation algorithm for the MSO_2 maximization problem with Lipschitz constant $O\left((f(\mathrm{tw}, |\varphi|) + \log \varepsilon^{-1} + \log\log n)\varepsilon^{-1}\log^2 n\right)$, where f is some computable function. The time complexity is bounded by $O\left(f(\mathrm{tw}, |\varphi|)n\right)$.*

▶ **Theorem 2.** *For any $\varepsilon \in (0, 1]$, there is a $(1+\varepsilon)$-approximation algorithm for the MSO_2 minimization problem with Lipschitz constant $O\left((f(\mathrm{tw}, |\varphi|) + \log \varepsilon^{-1} + \log\log n)\varepsilon^{-1}\log^2 n\right)$, where f is some computable function. The time complexity is bounded by $O\left(f(\mathrm{tw}, |\varphi|)n\right)$.*

We note that the trivial upper bound on Lipschitz constant is n [16]; therefore the bounds in the theorems above are significantly smaller for fixed tw and φ. We also remark that our meta-theorems yield randomized approximation algorithms. This is necessary since, for most problems, it is known that exact or deterministic algorithms cannot be Lipschitz continuous [14].

When the treewidth of the input graph is bounded by a constant, Theorems 1 and 2 provide algorithms that outperform existing algorithms in terms of approximability and/or Lipschitz continuity:

- For the minimum weight vertex cover problem, Theorem 2 yields an algorithm with a better approximation ratio than the previous 2-approximation algorithm with Lipschitz constant 4 [16], at the polylogarithmic sacrifice of the Lipschitz constant.
- For the minimum weight feedback vertex set problem, Theorem 2 outperforms the previous $O(\log n)$-approximation algorithm with Lipschitz constant $O(\sqrt{n}\log^{3/2} n)$ [16] in terms of both approximability and Lipschitz continuity.
- For the maximum weight matching problem[1], Theorem 1 yields an algorithm with a better approximation ratio than the previous $\left(\frac{1}{8} - \varepsilon\right)$-approximation algorithm with Lipschitz constant $O(\varepsilon^{-1})$ [14], at the polylogarithmic sacrifice of the Lipschitz constant.
- For the shortest path problem, Theorem 2 slightly improves the Lipschitz continuity compared to the previous $(1-\varepsilon)$-approximation algorithm with Lipschitz constant $O(\varepsilon^{-1}\log^3 n)$ [14], without losing approximability.

For a fixed φ, by explicitly specifying the transitions in the dynamic programming within the algorithm, we can provide more precise bounds on the function f that appears in Theorems 1 and 2. In particular, considering the case that φ is an MSO_2 formula representing the independent set constraint, we have the following.

▶ **Theorem 3.** *For any $\varepsilon \in (0, 1]$, there is a $(1-\varepsilon)$-approximation algorithm for the maximum weight independent set problem with Lipschitz constant $O\left((\mathrm{tw} + \log \varepsilon^{-1} + \log\log n)\varepsilon^{-1}\log^2 n\right)$. The time complexity is bounded by $2^{O(\mathrm{tw})}n$.*

This result is surprising as the dependence of tw on the Lipschitz constant is subexponential and, even more remarkably, linear. We note that through similar arguments, analogous results hold for several other problems, such as the minimum weight vertex cover problem and the minimum weight dominating set problem.

[1] Although the output of this problem is an edge set rather than a vertex set, this problem can be expressed by an MSO_2 formula on a new graph $G' = (V \cup E, \{(v, e) \colon e \text{ is incident to } v\})$

We further demonstrate that Theorems 1 and 2 lead to Lipschitz continuous version of Baker's technique [1]. As a representative example, we consider the maximum weight independent set problem on planar graphs, where a vertex subset X is an *independent set* of a graph G if no two vertices in X are adjacent in G. We prove the following.

▶ **Theorem 4.** *For any $\varepsilon \in (0,1]$, there is a $(1-\varepsilon)$-approximation algorithm for the maximum weight independent set problem on planar graphs with Lipschitz constant $O\left((\varepsilon^{-1}+\log\log n)\varepsilon^{-1}\log^2 n\right)$. The time complexity is bounded by $2^{O(\varepsilon^{-1})}n$.*

Using similar algorithms, Lipschitz continuous PTASes can be obtained for many problems, including the minimum weight vertex cover problem and the minimum weight dominating set problem.

As a lower bound result, we prove that the Lipschitz constant of any $(1-\varepsilon)$-approximation algorithm for some MSO_2 maximization problem is large on general graphs, which justifies considering the MSO_2 maximization problem on a restricted graph class. Specifically, we consider the *max ones problem* [6, 9], where the instance is a 3CNF formula over a variable set X and a weight function $w : X \to \mathbb{R}_{\geq 0}$, and the goal is to find a satisfying assignment $\sigma : X \to \{0,1\}$ that maximizes the weight $\sum_{x\in\sigma^{-1}(1)} w(x)$. It is easy to see that there exists a fixed MSO_2 formula $\varphi(X)$ such that the max ones problem can be reduced to a problem on bipartite graphs, where the task is to find a vertex set X that maximizes the weight $\sum_{x\in X} w(x)$ subject to φ. We show the following.

▶ **Theorem 5.** *There exist $\varepsilon, \delta > 0$ such that any $(1-\varepsilon)$-approximation algorithm for the max ones problem has Lipschitz constant $\Omega(n^{\delta})$, where n is the number of variables.*

We note that it is possible to construct an algorithm for the max ones problem with a Lipschitz constant similar to that in Theorem 3, where tw denotes the treewidth of the "incidence graph" of the 3CNF instance (see Section 4.2 for details). Since tw is upper bounded by n for any instance, Theorem 5 implies that the dependency of the Lipschitz constant on tw cannot be improved to $\mathrm{tw}^{o(1)}$.

We also prove analogous meta-theorems when parameterizing by clique-width. Let an *MSO_1 maximization (resp., minimization) problem* be the variant of an MSO_2 problem in which the formula φ is restricted to MSO_1. Denote by cw the clique-width of the graph G. We then obtain the following.

▶ **Theorem 6.** *For any $\varepsilon \in (0,1]$, there is a $(1-\varepsilon)$-approximation algorithm for the MSO_1 maximization problem with Lipschitz constant $O\left((f(\mathrm{cw},|\varphi|)+\log\varepsilon^{-1}+\log\log n)\varepsilon^{-1}\log^2 n\right)$, where f is some computable function. The time complexity is bounded by $O\left(f(\mathrm{cw},|\varphi|)n\right)$.*

▶ **Theorem 7.** *For any $\varepsilon \in (0,1]$, there is a $(1+\varepsilon)$-approximation algorithm for the MSO_1 minimization problem with Lipschitz constant $O\left((f(\mathrm{cw},|\varphi|)+\log\varepsilon^{-1}+\log\log n)\varepsilon^{-1}\log^2 n\right)$, where f is some computable function. The time complexity is bounded by $O\left(f(\mathrm{cw},|\varphi|)n\right)$.*

We note that the functions f in Theorems 6 and 7 are much larger than those in Theorems 1 and 2. In particular, the bounds for Theorems 3 and 4 do not follow from Theorem 6.

1.1 Technical Overview

Now we provide a technical overview of our framework. Since the arguments for clique-width are similar, we focus here on the treewidth results and omit the clique-width case.

For simplicity, here we consider the maximum weight independent set problem on a full binary tree (a rooted tree in which every vertex has exactly 0 or 2 children). This corresponds to the case where $\varphi(X) = \forall x \forall y((x \in X \wedge y \in X) \to \neg\mathsf{adj}(x,y))$. Let $w \in \mathbb{R}^V_{\geq 0}$ be the weight vector.

If we do not care about Lipschitzness, this problem can exactly be solved by the following algorithm. For each vertex $v \in V$, define $\mathsf{DP}[v][0]$ to be an independent set in the subtree rooted at v that does not include v with the maximum weight. Similarly, define $\mathsf{DP}[v][1]$ to be an independent set in the subtree rooted at v with the maximum weight. If v is a leaf, then $\mathsf{DP}[v][0] = \emptyset$ and $\mathsf{DP}[v][1] = \{v\}$. Otherwise, let u_1 and u_2 be the two children of v. We have $\mathsf{DP}[v][0] = \mathsf{DP}[u_1][1] \cup \mathsf{DP}[u_2][1]$, and $\mathsf{DP}[v][1]$ is the one with the larger weight of $X_{v,0} := \mathsf{DP}[u_1][0] \cup \mathsf{DP}[u_2][0] \cup \{v\}$ and $X_{v,1} := \mathsf{DP}[u_1][1] \cup \mathsf{DP}[u_2][1]$. By performing this dynamic programming in a bottom-up manner, the problem can be solved exactly.

However, this algorithm is not Lipschitz continuous. This is because when we compute $\mathsf{DP}[v][1]$, which of $X_{v,0}$ or $X_{v,1}$ has a larger weight is affected by small changes in the weights. To address this issue, we use the exponential mechanism [7]. Specifically, for some constant $c > 0$, we select $X_{v,i}$ with probability proportional to $\exp(c \cdot w(X_{v,i}))$. This approach makes the algorithm Lipschitz continuous, with only a slight sacrifice in the approximation ratio. Specifically, we can prove that by appropriately choosing c for a given $\varepsilon' \in (0, 1]$, the increase in the Lipschitz constant can be bounded by $\tilde{O}(\varepsilon'^{-1})$ by reducing the approximation guarantee by a factor of $1 - \varepsilon'$ at each vertex where the exponential mechanism is applied.

While this approach makes the algorithm Lipschitz continuous, the Lipschitz constant is still too large when the height h of the tree is large. Specifically, to achieve an approximation guarantee of $1 - \varepsilon$, we need to set $\varepsilon' = \frac{\varepsilon}{h}$, leading to a Lipschitz constant of $h \cdot \tilde{O}(\varepsilon'^{-1}) = \tilde{O}(\varepsilon^{-1}h^2)$. This is larger than the trivial bound n when $h = O(n)$. We resolve this issue by using the fact that any tree has a tree decomposition of width 5 and height $O(\log n)$ [2]. By performing dynamic programming with the exponential mechanism on this tree decomposition, we can obtain $(1-\varepsilon)$-approximation algorithm with Lipschitz constant $\tilde{O}(\varepsilon^{-1})$. This argument can be naturally extended to the case where G is a bounded treewidth graph. By following the proof of Courcelle's Theorem [3, 5], we further extend this argument to the case where φ is a general MSO_2 formula with a free vertex variable.

We prove Theorem 5 by leveraging a lower bound on a related notion of sensitivity for the maximum cut problem [8], where sensitivity measures the change in the output with respect to the Hamming distance under edge deletions [21]. Specifically, we reduce the maximum cut problem, where stability is defined with respect to edge deletions, to the max ones problem, where stability is measured with respect to weight changes.

1.2 Related Work

Lipschitz continuity of discrete algorithms is defined by Kumabe and Yoshida [14], and they provided Lipschitz continuous approximation algorithms for the minimum spanning tree, shortest path, and maximum weight matching problems. In a subsequent work [16], they further provided such algorithms for the minimum weight vertex cover, minimum weight set cover, and minimum weight feedback vertex set problems. In another work [15], they defined Lipschitz continuity for allocations in cooperative games and provided Lipschitz continuous allocation schemes for the matching game and the minimum spanning tree game.

A variant known as *pointwise Lipschitz continuity* has also been studied, which is defined using the unweighted Hamming distance instead of the weighted Hamming distance. Kumabe and Yoshida [14] defined pointwise Lipschitz continuity and provided pointwise Lipschitz continuous algorithms for the minimum spanning tree and maximum weight bipartite matching problems. Liu et al. [18] proposed the *proximal gradient method* as a general technique for solving LP relaxations stably. Using this, they provided pointwise Lipschitz continuous approximation algorithms for the minimum vertex (S, T)-cut, densest subgraph, maximum weight (b-)matching, and packing integer programming problems.

(Average) sensitivity, introduced by Varma and Yoshida [21] with preliminary ideas by Murai and Yoshida [19], is a notion similar to Lipschitz continuity. While Lipschitz continuity evaluates an algorithm's stability against unit changes in the input weights, (average) sensitivity evaluates an algorithm's stability against (random) deletions of elements from the input. Algorithms with small (average) sensitivity have been studied for several problems, such as maximum matching [21, 23], minimum cut [21], knapsack problem [13], Euclidean k-means [22], spectral clustering [20], and dynamic programming problems [12]. Recently, Fleming and Yoshida [8] constructed a PCP framework to prove the sensitivity lower bound for the constraint satisfaction problem.

1.3 Organization

The rest of this paper is organized as follows. In Section 2, we describe the necessary preliminaries on tree decomposition (Sections 2.1 and 2.2), logic (Section 2.3), and Lipschitz continuity (Section 2.4). In Section 3, we prove Theorem 1 by providing a Lipschitz continuous algorithm for the MSO_2 maximization problem. Since the algorithm and analysis for MSO_2 minimization are similar, we defer the proof of Theorem 2 to the full version. In Section 4, we give a more precise Lipschitzness analysis for specific formulas φ, which includes the proof of Theorem 3. Proofs of Theorems 4, 5, 6, and 7 are given in the full version.

2 Preliminaries

2.1 Tree Decomposition

Let G be a graph with n vertices. A pair $(\mathcal{B}, \mathcal{T})$ consisting of a family $\mathcal{B}$ of subsets (called *bags*) of $V(G)$ and a rooted tree $\mathcal{T}$ whose vertex set is $\mathcal{B}$ is a *(rooted) tree decomposition* of G if it satisfies the following three conditions.

- $\bigcup_{B\in\mathcal{B}} B = V(G)$.
- For each edge $e \in E(G)$, there is a bag $B \in \mathcal{B}$ such that $e \subseteq B$.
- For each vertex $v \in V(G)$, the set of bags $\{B \in \mathcal{B} \colon v \in B\}$ induces connected subgraph in T.

For a tree decomposition $(\mathcal{B}, \mathcal{T})$, we may refer to the *root node*, *leaf nodes*, and the *height* of $\mathcal{T}$ as those of $(\mathcal{B}, \mathcal{T})$, respectively. Moreover, $(\mathcal{B}, \mathcal{T})$ is binary if $\mathcal{T}$ is a binary tree. The *width* of a tree decomposition $(\mathcal{B}, \mathcal{T})$ is the maximum size of a bag in $\mathcal{B}$ minus one. The *treewidth* of G is the minimum possible width among all possible tree decompositions of G. It is known that the tree decomposition of G of width at most $2k+1$ can be computed in $2^{O(k)}n$ time [10], where k is the treewidth of G. Moreover, any tree decomposition of G of width k can be transformed into a binary tree decomposition of width at most $3k+2$ and height at most $O(\log n)$ [2]. Thus, we obtain the following lemma.

▶ **Lemma 8** ([2, 10]). *Let G be a graph with n vertices and k be the treewidth of G. Then, a binary tree decomposition of G with width $O(k)$ and height $O(\log n)$ can be computed in $2^{O(k)}n$ time.*

2.2 HR-algebra

We introduce the notions of HR-algebra[2], which is one of the algebraic definitions of tree decomposition. A *k-graph* is a tuple $G = \langle V, E, \mathsf{src}\rangle$ of a set of vertices V, a set of edges E, and a k-vector $\mathsf{src} \in (V \cup \{\mathtt{nil}\})^k$. We write the i-th element of a vector src by $\mathsf{src}(i)$. If $\mathsf{src}(i) = u$, we say that u is a *i-source*. We write the set $\{\mathsf{src}(i) : i \in [k]\} \setminus \{\mathtt{nil}\}$ by $\mathsf{src}(G)$.

Let G and H be k-graphs. The *parallel-composition* of G and H, denoted by $G /\!\!/ H$, is the graph generated by the following procedure. First, create the disjoint union of G and H, and identify the (non-$\mathtt{nil}$) vertices $\mathsf{src}_G(i)$ and $\mathsf{src}_H(i)$ for each $i \in [k]$. Finally, remove all the self-loops and multi-edges. Let B be a non-empty subset of $[k]$. The *source forgetting operation* fg_B is the function that maps a k-graph G to a k-graph G' such that $V_G = V_{G'}$, $E_G = E_{G'}$, and $\mathsf{src}_{G'}(i) = \mathtt{nil}$ if $i \in B$, and $\mathsf{src}_{G'}(i) = \mathsf{src}_G(i)$ otherwise.

▶ **Definition 9** (HR-algebra)**.** *A* term *of a HR-algebra over k-graphs is either*

- *a constant symbol* $\mathtt{i}$, $\mathtt{ij}$, *or* $\emptyset$ *denoting a k-graph* $\langle\{v\}, \emptyset, \mathsf{src}\rangle$ *with* $\mathsf{src}(i) = v$, *a k-graph* $\langle\{u, v\}, \{\{u, v\}\}, \mathsf{src}\rangle$ *with* $\mathsf{src}(i) = u$ *and* $\mathsf{src}(j) = v$, *or the empty graph, respectively;*
- $t /\!\!/ s$ *for any terms t and s;*
- $\mathsf{fg}_B(t)$ *for any term t and any $B \subseteq [k]$.*

We may associate a term with the graphs represented by it. A term of an HR-algebra t can be decomposed into the parse tree in the usual way, called the *(rooted) HR-parse tree* of t. The *height* of a HR-parse tree is the maximum distance from the root to a leaf. It is known that tree decompositions are equivalent to HR-parse trees in the following sense.

▶ **Proposition 10** (see e.g. [4])**.** *The treewidth of a graph is at most k if and only if the graph can be denoted by a term of a HR-algebra over $(k+1)$-graphs.*

Moreover, binary tree decomposition can be transformed into an HR-parse tree with approximately the same height.

▶ **Proposition 11** ($\star$)**.** *Given a rooted binary tree decomposition $(\mathcal{B}, \mathcal{T})$ of G with width k and height h, we can compute a HR-parse tree over $(k+1)$-graph denoting G with height $O(\log k + h)$.*

The proof of Proposition 11 is essentially the same as the proof of Proposition 10 but we give in the full version.

2.3 Monadic Second-Order Logic

We provide a slightly less formal definition of *monadic second-order (MSO) logic* over k-graphs for accessibility (see, e.g., [17] for more formal definition). An *atomic formula* is a formula of the form $s = t$, $\mathsf{adj}(s, t)$, $t \in X$, and Boolean constants $\mathtt{True}$ and $\mathtt{False}$. Here, $\mathsf{adj}(s, t)$ is the predicate that represents whether s and t are adjacent. A *monadic second-order formula* over k-graphs is a formula built from atomic formulas, the usual Boolean connectives $\wedge, \vee, \neg, \to$, first-order quantifications $\exists x$ and $\forall x$ (quantifications over vertices), and second-order quantifications $\exists X$ and $\forall X$ (quantification over sets of vertices). The notation $\varphi(X)$ indicates that φ has exactly one free variable X. The *quantifier height* (or *quantifier rank*) of a formula φ, denoted by $|\varphi|$, is the maximum depth of quantifier nesting in φ (see e.g. [17] for a formal definition). The *counting monadic second-order (CMSO) logic*

[2] The acronym HR stands for hyperedge replacement [4].

is an expansion of MSO logic that have additional predicates $\mathsf{Card}_{m,r}(X)$ for a second-order variable X, meaning $|X| \equiv m \pmod r$. A $\mathsf{C}_r\mathsf{MSO}^q$-formula is a CMSO formula φ such that the quantifier height of φ is at most q and $r' \leq r$ holds for any predicate $\mathsf{Card}_{m,r'}$ in φ. For a logical formula φ and a graph G, we write $G \models \varphi$ when graph G satisfies the property expressed by φ.

There are two variants of MSO logic: MSO_1 (or simply MSO), which is described above, allows quantification over vertices and sets of vertices only, and MSO_2 which allows quantification over edges and sets of edges as well. It is well known that, for any graph G and MSO_2-formula φ, there exists an MSO_1 formula φ' such that $G \models \varphi \iff G' \models \varphi'$, where G' is the *incidence graph*, obtained by subdividing each edge of G (with two colors meaning the original and the subdivision vertices) (see e.g. [11]). It is easy to see that the treewidth of the incidence graph is at most the treewidth of the original graph. Thus, since this paper focuses on graphs with bounded treewidth, we mainly consider MSO_1. However, all the theorems are held for MSO_2 and graphs of bounded treewidth.

We introduce some more notations. For a set V, two families $\mathcal{A} \subseteq 2^V$ and $\mathcal{B} \subseteq 2^V$ are *separated* if $A \cap B = \varnothing$ for all $A \in \mathcal{A}$, and $B \in \mathcal{B}$. We write $A \cup B$ by $A \uplus B$ if $A \cap B = \varnothing$, and $\mathcal{A} \boxtimes \mathcal{B} \coloneqq \{A \cup B : A \in \mathcal{A}, B \in \mathcal{B}\}$ if $\mathcal{A}$ and $\mathcal{B}$ are separated. For a formula $\varphi(X)$ and a graph G, the set of vertex sets satisfying φ is denoted by $\mathsf{sat}(G, \varphi)$, that is, $\mathsf{sat}(G, \varphi) = \{A : G \models \varphi(A)\}$.

The following theorem is a key of the MSO model-checking algorithm [3, 4].

▶ **Theorem 12** ([4]). *Let r, q, k be positive integers. Let $\varphi(X)$ be a $\mathsf{C}_r\mathsf{MSO}^q$-formula over k-graphs with a second-order free variable X. Then, the following hold.*

1. *For any $B \subseteq [k]$, there exists a $\mathsf{C}_r\mathsf{MSO}^q$-formula $\psi(X)$ such that for any k-graph G, we have*

$$\mathsf{sat}(\mathsf{fg}_B(G), \varphi) = \mathsf{sat}(G, \psi).$$

2. *There exists a family of tuples $\{\theta_i(X), \psi_i(X)\}_{i \in [p]}$ of $\mathsf{C}_r\mathsf{MSO}^q$-formulas with a free variable X such that, for any k-graphs G and H,*

$$\mathsf{sat}(G /\!\!/ H, \varphi) = \biguplus_{i \in [p]} \{S \cup P : S \in \mathsf{sat}(G, \theta_i), P \in \mathsf{sat}(H, \psi_i)\}.$$

It is known that, for any r, q, k, up to logical equivalence, there are only finitely many different $\mathsf{C}_r\mathsf{MSO}^q$-formulas $\varphi(X)$ over k-graphs (see e.g., [17]). Thus, we can obtain a linear-time algorithm, based on dynamic programming over trees, for $\mathsf{C}_r\mathsf{MSO}^q$ model checking over bounded treewidth. Courcelle and Mosbah [5] adjusted Theorem 12 to address optimization, counting, and other problems. We use a slightly modified version of the theorem of Courcelle and Mosbah.

Let $\mathsf{nosrc}(X) \equiv \bigwedge_{i \in [k]} (\mathsf{src}(i) \notin X)$, and $\varphi\restriction_S(X)$ denote $\varphi(X \cup S) \wedge \mathsf{nosrc}(X)$ for a set $S \subseteq \mathsf{src}(G)$.

▶ **Observation 13.** *Let G be a k-graph, and $\varphi(X)$ be a formula with a free set variable X. Then, we have $\mathsf{sat}(G, \varphi) = \biguplus_{S \subseteq \mathsf{src}(G)} \mathsf{sat}(G, \varphi\restriction_S) \boxtimes \{S\}$.*

The proof of Corollary 14 is given in the full version.

▶ **Corollary 14** (⋆). *Let r, q, k be positive integers. Let $\varphi(X)$ be a $\mathsf{C}_r\mathsf{MSO}^q$-formula over k-graphs with a second-order free variables X. Then, for all $S \subseteq \mathsf{src}(G)$, the following hold.*

1. *For any $B \subseteq [k]$, there exists a $\mathsf{C}_r\mathsf{MSO}^q$-formula $\psi(X)$ such that for any k-graph G, we have*

$$\mathsf{sat}(\mathsf{fg}_B(G), \varphi\restriction_S) = \biguplus_{S' \subseteq B} \mathsf{sat}(G, \psi\restriction_{S \cup S'}) \boxtimes \{S'\}.$$

2. *There exists a family of tuples* $\{\theta_i\restriction_{S_i}(X), \psi_i\restriction_{S'_i}(X)\}_{i\in[p]}$ *of* $\mathsf{C}_r\mathsf{MSO}^q$*-formulas with a free variable* X *such that, for any* k*-graphs* G *and* H*,*

$$\mathsf{sat}(G /\!\!/ H, \varphi\restriction_S) = \biguplus_{i\in[p]} \mathsf{sat}(G, \theta_i\restriction_{S_i}) \boxtimes \mathsf{sat}(H, \psi_i\restriction_{S'_i}).$$

Moreover, $S_i \cup S'_i = S$ *for all* $i \in [p]$*.*

Then, we can design an algorithm for an (C)MSO maximization (minimization) problem over graphs of bounded treewidth. For simplicity, we assume that the given graph G has no sources, that is, $\mathsf{src}(G) = \varnothing$. Let t be a term denoting G_t and $\varphi(X)$ be the (C)MSO-formula describing the constraint of the problem. We recursively compute the value $\mathsf{opt}(t', \varphi'\restriction_S) = \max\{w(A) : G \models \varphi'\restriction_S(A)\}$ for any subterm t' of t and MSO-formula $\varphi'\restriction_S$ as follows, where $w \in \mathbb{R}^V_{\geq 0}$ is the given weight vector and $w(X) = \sum_{x\in X} w_x$ for any $X \subseteq V(G)$. If t' is of the form $\mathsf{fg}_B(t'')$, then $\mathsf{opt}(\mathsf{fg}_B(t''), \varphi'\restriction_S) = \max_{S'\subseteq B}\{\mathsf{opt}(t'', \psi\restriction_{S\cup S'}) + w(S')\}$, where, ψ is the formula obtained from Corollary 14. If t' is of the form $t_1 /\!\!/ t_2$, then $\mathsf{opt}(t', \varphi'\restriction_S) = \max_{i\in[p]}\{\mathsf{opt}(t_1, \theta\restriction_{S_i}) + \mathsf{opt}(t_2, \psi\restriction_{S'_i})\}$, where, $\{\theta_i\restriction_{S_i}(X), \psi_i\restriction_{S'_i}(X)\}_{i\in[p]}$ are the formulas obtained from Corollary 14. Then, $\mathsf{opt}(t, \varphi\restriction_\varnothing)$ is the maximum value for the problem since t has no sources. In the following, for simplicity, we may omit the $\restriction_S$ notation and consider only formulas φ such that $\mathsf{sat}(G, \varphi)$ does not contain any sources.

In the above setting, the graphs are considered without any special vertex sets (called as colors) or special vertices (called as labels). However, for k-graphs with a constant number of colors and/or labels, an appropriate HR-algebra and logics can be defined, and the similar results for Corollary 14 hold (see e.g. [4, 17]).

2.4 Lipschitz Continuity

We formally define Lipschitz continuity of algorithms. As this is sufficient for this work, we only consider algorithms for vertex-weighted graph problems. The definition can be naturally extended to other settings, such as edge-weighted problems. Let $G = (V, E)$ be a graph. For vertex sets $X, X' \subseteq V$ and weight vectors $w, w' \in \mathbb{R}^V_{\geq 0}$, we define the *weighted Hamming distance* between (X, w) and (X', w') by

$$d_{\mathrm{w}}((X, w), (X', w')) := \left\| \sum_{v\in X} \mathbf{1}_v w_v - \sum_{v\in X'} \mathbf{1}_v w'_v \right\|_1 = \sum_{v\in X\cap X'} |w_v - w'_v| + \sum_{v\in X\setminus X'} w_v + \sum_{v\in X'\setminus X} w'_v,$$

where $\mathbf{1}_v \in \{0,1\}^V$ denotes the *characteristic vector* of v, that is, the vector $\mathbf{1}_{v,u} = 1$ holds if and only if $u = v$. For two probability distributions $\mathcal{X}$ and $\mathcal{X}'$ over subsets of V, we define

$$\mathsf{EM}\left((\mathcal{X}, w), (\mathcal{X}', w')\right) := \inf_{\mathcal{D}} \mathbb{E}_{(X,X')\sim\mathcal{D}}\left[d_{\mathrm{w}}((X, w), (X', w'))\right],$$

where the minimum is taken over *couplings* of $\mathcal{X}$ and $\mathcal{X}'$, that is, distributions over pairs of sets such that its marginal distributions on the first and second coordinates are equal to $\mathcal{X}$ and $\mathcal{X}'$, respectively. Consider an algorithm $\mathcal{A}$, that takes a graph $G = (V, E)$ and a weight vector $w \in \mathbb{R}^V_{\geq 0}$ as an input and outputs a vertex subset $X \subseteq V$. We denote the output distribution of $\mathcal{A}$ for weight w as $\mathcal{A}(w)$. The *Lipschitz constant* of $\mathcal{A}$ is defined by

$$\sup_{\substack{w,w'\in\mathbb{R}_{\geq 0},\\ w\neq w'}} \frac{\mathsf{EM}\left((\mathcal{A}(w), w), (\mathcal{A}(w'), w')\right)}{\|w - w'\|_1}.$$

For two random variables $\boldsymbol{X}$ and $\boldsymbol{X}'$, the *total variation distance* between them is given as:

$$\mathsf{TV}\left(\boldsymbol{X}, \boldsymbol{X}'\right) := \inf_{\mathcal{D}} \Pr_{(X,X')\sim\mathcal{D}}\left[X \neq X'\right],$$

where the minimum is taken over couplings between $\boldsymbol{X}$ and $\boldsymbol{X}'$, that is, distributions over pairs such that its marginal distributions on the first and the second coordinates are equal to $\boldsymbol{X}$ and $\boldsymbol{X}'$, respectively. For an element $u \in V$, we use $\mathbf{1}_u \in \mathbb{R}^V_{\geq 0}$ to denote the *characteristic vector* of u, that is, $\mathbf{1}_u(u) = 1$ and $\mathbf{1}_u(v) = 0$ for $v \in V \setminus \{u\}$. The following lemma indicates that, to bound the Lipschitz constant, it suffices to consider pairs of weight vectors that differ by one coordinate.

▶ **Lemma 15** ([14]). *Suppose that there exist some $c > 0$ and $L > 0$ such that*

$$\mathsf{EM}\left((\mathcal{A}(G,w),w),(\mathcal{A}(G,w+\delta\mathbf{1}_u),w+\delta\mathbf{1}_u)\right) \leq \delta L$$

holds for all $w \in \mathbb{R}^V_{\geq 0}$, $u \in V$ and $0 < \delta \leq c$. Then, $\mathcal{A}$ is L-Lipschitz.

In our algorithm, we use the following procedures *softmax* and *softmin*. These procedures are derived from the *exponential mechanism* in the literature of *differential privacy* [7] and frequently appear in the literature on Lipschitz continuity [14, 16]. Here, we organize them into a more convenient form for our use. Let $p \in \mathbb{Z}_{\geq 1}$, $x_1, \ldots, x_p \in \mathbb{R}_{\geq 0}$, and $\epsilon \in (0, 1]$. The *softmax* of $x_1, \ldots, x_p$ is taken as follows. If $\max_{i\in[p]} x_i = 0$, we set $\mathsf{softargmax}^{\epsilon}_{i\in[p]} x_i$ to be an arbitrary index i with $x_i = 0$ and $\mathsf{softmax}^{\epsilon}_{i\in[p]} x_i = 0$. Assume otherwise. First, we sample $\boldsymbol{c}$ uniformly from $\left[\frac{2\epsilon^{-1}\log(2p\epsilon^{-1})}{\max_{i\in[p]} x_i}, \frac{4\epsilon^{-1}\log(2p\epsilon^{-1})}{\max_{i\in[p]} x_i}\right]$. Let $\boldsymbol{i}^*$ be a probability distribution over $[p]$ such that

$$\Pr\left[\boldsymbol{i}^* = i\right] = \frac{\exp(\boldsymbol{c}x_i)}{\sum_{i'\in[p]}\exp(\boldsymbol{c}x_{i'})}.$$

holds for all $i \in [p]$. Then, we define $\mathsf{softargmax}^{\epsilon}_{i\in[p]} x_i := \boldsymbol{i}^*$ and $\mathsf{softmax}^{\epsilon}_{i\in[p]} x_i := x_{\boldsymbol{i}^*}$. We have the following. The proofs of Lemmas 16 and 17 are given in the full version.

▶ **Lemma 16.** *We have $\mathbb{E}\left[\mathsf{softmax}^{\epsilon}_{i\in[p]} x_i\right] \geq (1-\epsilon)\max_{i\in[p]} x_i$.*

▶ **Lemma 17.** *Let $\delta > 0$ and assume $\delta \leq \frac{\max_{i\in[p]} x_i}{4\epsilon^{-1}\log(2p\epsilon^{-1})}$. Let $x'_1, \ldots, x'_p \in \mathbb{R}_{\geq 0}$ be numbers such that $x_i \leq x'_i \leq x_i + \delta$ holds for all $i \in [p]$. Then, $\mathsf{TV}\left(\mathsf{softargmax}^{\epsilon}_{i\in[p]} x_i, \mathsf{softargmax}^{\epsilon}_{i\in[p]} x'_i\right) \leq \frac{10\epsilon^{-1}\log(2p\epsilon^{-1})\delta}{\max_{i\in[p]} x_i}$.*

Similarly, the *softmin* of $x_1, \ldots, x_p$ is taken as follows. If $\min_{i\in[p]} x_i = 0$, we set $\mathsf{softargmin}^{\epsilon}_{i\in[p]} x_i$ be an arbitrary index i with $x_i = 0$ and $\mathsf{softmin}^{\epsilon}_{i\in[p]} x_i = 0$. Assume otherwise. First, we sample $\boldsymbol{c}$ uniformly from $\left[\frac{2\epsilon^{-1}\log(2p\epsilon^{-1})}{\min_{i\in[p]} x_i}, \frac{4\epsilon^{-1}\log(2p\epsilon^{-1})}{\min_{i\in[p]} x_i}\right]$. Let $\boldsymbol{i}^*$ be a probability distribution over $[p]$ such that

$$\Pr\left[\boldsymbol{i}^* = i\right] = \frac{\exp(-\boldsymbol{c}x_i)}{\sum_{i'\in[p]}\exp(-\boldsymbol{c}x_{i'})}.$$

holds for all $i \in [p]$. Then, we define $\mathsf{softargmin}^{\epsilon}_{i\in[p]} x_i := \boldsymbol{i}^*$ and $\mathsf{softmin}^{\epsilon}_{i\in[p]} x_i := x_{\boldsymbol{i}^*}$. We have the following. The proofs of Lemmas 18 and 19 are given in the full version.

▶ **Lemma 18.** *We have $\mathbb{E}\left[\mathsf{softmin}^{\epsilon}_{i\in[p]} x_i\right] \leq (1+\epsilon)\min_{i\in[p]} x_i$.*

▶ **Lemma 19.** *Let $\delta > 0$ and assume $\delta \leq \frac{\min_{i\in[p]} x_i}{4\epsilon^{-1}\log(2p\epsilon^{-1})}$. Let $x'_1, \ldots, x'_p \in \mathbb{R}_{\geq 0}$ be numbers such that $x_i \leq x'_i \leq x_i + \delta$ holds for all $i \in [p]$. Then, $\mathsf{TV}\left(\mathsf{softargmin}^{\epsilon}_{i\in[p]} x_i, \mathsf{softargmin}^{\epsilon}_{i\in[p]} x'_i\right) \leq \frac{10\epsilon^{-1}\log(2p\epsilon^{-1})\delta}{\min_{i\in[p]} x_i}$.*

3 Lipschitz Continuous Algorithms for MSO_2 Optimization Problems

In this section, we prove Theorems 1–3. Since the proofs are similar to that for Theorem 1, we provide most of the proof of Theorem 2 in the full version. In Section 3.1, we define the notations and give a brief overview of the algorithm. Section 3.2 handles the base case, where the graph consists of a single vertex. Sections 3.3 and 3.4 analyze the impact of the parallel composition and the forget operations, respectively, on approximability and the Lipschitz constant. In Section 3.5, we put everything together to prove Theorem 1. Finally, in Section 4.1, we provide a more refined analysis for the special case of the maximum weight independent set problem to prove Theorem 3.

3.1 Definitions

For a graph G, a weight vector w, and an MSO_2 formula φ, we define

$$\mathsf{opt}_w[G,\varphi] = \max_{S\subseteq V(G), G\models\varphi(S)} w(S). \tag{1}$$

for maximization problems and

$$\mathsf{opt}_w[G,\varphi] = \min_{S\subseteq V(G), G\models\varphi(S)} w(S). \tag{2}$$

for minimization problems. If $\mathsf{sat}(G,\varphi)=\emptyset$, we define $\mathsf{opt}_w[G,\varphi] = \mathtt{nil}$. For graphs G and MSO_2 formulas φ with $\mathsf{sat}(G,\varphi)\neq\emptyset$, our algorithm recursively computes the vertex set $\mathsf{DP}_w[G,\varphi]$ that approximately achieves the maximum in Equation (1) or minimum in Equation (2). When it is clear from the context, we omit the subscript w and simply write $\mathsf{opt}[G,\varphi]$ and $\mathsf{DP}[G,\varphi]$. We perform the dynamic programming algorithm using the formulas defined in Corollary 14 over the parse tree of a term obtained from Lemma 8 and Proposition 11. Specifically, at every stage of the algorithm, it is guaranteed that each $X\in\mathsf{sat}(G,\varphi)$ satisfies $X\subseteq V(G)\setminus\mathsf{src}(G)$. Furthermore, our algorithm is randomized. Thus, $\mathsf{DP}[G,\varphi]$ can be considered as a probability distribution over vertex sets in $\mathsf{sat}(G,\varphi)$.

To bound the approximation ratio, for a weight vector $w\in\mathbb{R}^V_{\geq 0}$, we bound the ratio between $\mathbb{E}\left[w(\mathsf{DP}[G,\varphi])\right]$ and $\mathsf{opt}[G,\varphi]$. To bound the Lipschitz constant, for a weight vector $w\in\mathbb{R}^V_{\geq 0}$, $u\in V$, and $\delta>0$, we bound $\mathsf{EM}\left(\mathsf{DP}_w[G,\varphi],\mathsf{DP}_{w+\delta\mathbf{1}_u}[G,\varphi]\right)$. From now on, we will concentrate on maximizing problems. The algorithm for minimization problems is similar to that for maximization problems, while the detail of the analysis is slightly differ. We discuss the minimization version in the full version.

3.2 Base Case

Here we consider the base case. Let G be a graph with a single vertex v and no edges, where v is a source of G, and φ be an MSO_2 formula such that $\mathsf{sat}(G,\varphi)$ is nonempty and contains no set containing v. In particular, we have $\mathsf{sat}(G,\varphi)=\{\emptyset\}$. We set $\mathsf{DP}[G,\varphi]:=\emptyset$. Since $w(\emptyset)=0$, it is clear that $w\left(\mathsf{DP}[G,\varphi]\right)=\mathsf{opt}[G,\varphi]=0$. Furthermore, it is obvious that

$$\mathsf{EM}\left(\mathsf{DP}_w[G,\varphi],\mathsf{DP}_{w+\delta\mathbf{1}_u}[G,\varphi]\right)=\mathsf{EM}(\emptyset,\emptyset)=0.$$

3.3 Parallel Composition

Next, we consider the parallel composition. Let

$$\mathsf{sat}(G /\!\!/ H,\varphi) = \biguplus_{i\in[p]} \mathsf{sat}(G,\theta_i)\boxtimes\mathsf{sat}(H,\psi_i). \tag{3}$$

Assume we have already computed $\mathsf{DP}[G, \theta_i]$ and $\mathsf{DP}[H, \psi_i]$ for each $i \in [p]$. We compute $\mathsf{DP}[G /\!\!/ H, \varphi]$. For each $i \in [p]$, we denote $\mathsf{opt}_i := \mathsf{opt}[G, \theta_i] + \mathsf{opt}[H, \psi_i]$. By definition, we have $\max_{i \in [p]} \mathsf{opt}_i = \mathsf{opt}[G /\!\!/ H, \varphi]$. Then, we take $\boldsymbol{i}^* = \mathsf{softargmax}^{\varepsilon}_{i \in [p]} \mathsf{opt}_i$ and define $\mathsf{DP}[G /\!\!/ H, \varphi] := \mathsf{DP}[G, \theta_{\boldsymbol{i}^*}] \cup \mathsf{DP}[H, \psi_{\boldsymbol{i}^*}]$. First we analyze the approximation ratio.

▶ **Lemma 20.** *Let $0 < \alpha \leq 1$ and suppose $\mathbb{E}[w(\mathsf{DP}[G, \theta_i])] \geq \alpha \mathsf{opt}[G, \theta_i]$ and $\mathbb{E}[w(\mathsf{DP}[H, \psi_i])] \geq \alpha \mathsf{opt}[H, \psi_i]$ hold for all $i \in [p]$. Then, we have $\mathbb{E}[w(\mathsf{DP}[G /\!\!/ H, \varphi])] \geq (1 - \varepsilon)\alpha \mathsf{opt}[G /\!\!/ H, \varphi]$.*

Proof. We have

$$\begin{aligned}
&\mathbb{E}[w(\mathsf{DP}[G /\!\!/ H, \varphi])] = \mathbb{E}[w(\mathsf{DP}[G, \theta_{\boldsymbol{i}^*}] \cup \mathsf{DP}[H, \psi_{\boldsymbol{i}^*}])] \geq \alpha \mathbb{E}[\mathsf{opt}[G, \theta_{\boldsymbol{i}^*}] + \mathsf{opt}[H, \psi_{\boldsymbol{i}^*}]] \\
&= \alpha \mathbb{E}\left[\mathsf{softmax}^{\varepsilon}_{i \in [p]} \mathsf{opt}_i\right] \geq (1 - \varepsilon)\alpha \max_{i \in [p]} \mathsf{opt}_i = (1 - \varepsilon)\alpha \mathsf{opt}[G /\!\!/ H, \varphi].
\end{aligned}$$

◀

Now we analyze the Lipschitz constant. We denote the variable $\boldsymbol{i}^*$ corresponding to the weight w and $w + \delta \mathbf{1}_u$ by $\boldsymbol{i}^*_w$ and $\boldsymbol{i}^*_{w+\delta \mathbf{1}_u}$, respectively. We note that $\mathsf{DP}_w[G, \theta_i]$ and $\mathsf{DP}_{w+\delta \mathbf{1}_u}[G, \theta_i]$ are the same as a distribution unless $u \in V(G) \setminus \mathsf{src}_G$. The same also holds for H. Since $V(G) \setminus \mathsf{src}_G$ and $V(H) \setminus \mathsf{src}_H$ are disjoint, without loss of generality, we can assume the $\mathsf{DP}_w[H, \psi_i]$ and $\mathsf{DP}_{w+\delta \mathbf{1}_u}[H, \psi_i]$ are the same as a distribution. We have the following.

▶ **Lemma 21.** *Let $\beta \in \mathbb{R}_{\geq 0}$ and suppose $\mathsf{EM}(\mathsf{DP}_w[G, \theta_i], \mathsf{DP}_{w+\delta \mathbf{1}_u}[G, \theta_i]) \leq \beta$ holds for all $i \in [p]$. Then, we have $\mathsf{EM}(\mathsf{DP}_w[G /\!\!/ H, \varphi], \mathsf{DP}_{w+\delta \mathbf{1}_u}[G /\!\!/ H, \varphi]) \leq 30\varepsilon^{-1} \log(2p\varepsilon^{-1})\delta + \beta$.*

Proof. We have

$$\begin{aligned}
&\mathsf{EM}(\mathsf{DP}_w[G /\!\!/ H, \varphi], \mathsf{DP}_{w+\delta \mathbf{1}_u}[G /\!\!/ H, \varphi]) \\
&\leq \mathsf{TV}\left(\boldsymbol{i}^*_w, \boldsymbol{i}^*_{w+\delta \mathbf{1}_u}\right)\left(\mathsf{opt}_w[G /\!\!/ H, \varphi] + \mathsf{opt}_{w+\delta \mathbf{1}_u}[G /\!\!/ H, \varphi]\right) + \max_{i \in [p]}\left(\mathsf{EM}(\mathsf{DP}_w[G, \theta_i], \mathsf{DP}_{w+\delta \mathbf{1}_u}[G, \theta_i])\right) \\
&\leq \frac{10\varepsilon^{-1} \log(2p\varepsilon^{-1})\delta}{\mathsf{opt}_w[G /\!\!/ H, \varphi]}\left(\mathsf{opt}_w[G /\!\!/ H, \varphi] + \mathsf{opt}_{w+\delta \mathbf{1}_u}[G /\!\!/ H, \varphi]\right) + \beta \leq 30\varepsilon^{-1} \log(2p\varepsilon^{-1})\delta + \beta,
\end{aligned}$$

where the last inequality is from $\varepsilon \leq 1$ and $\delta \leq \mathsf{opt}_w[G /\!\!/ H, \varphi]$. ◀

3.4 Forget

Here we consider forget operation. Let $B \subseteq \mathsf{src}_G$ and

$$\mathsf{sat}(\mathsf{fg}_B(G), \varphi) = \biguplus_{i \in [p]} \mathsf{sat}(G, \varphi_i) \boxtimes \{S_i\}. \tag{4}$$

Assume we have already computed $\mathsf{DP}[G, \varphi_i]$ for each $i \in [p]$. We compute $\mathsf{DP}[\mathsf{fg}_B(G), \varphi]$.

We first sample $\boldsymbol{c} \in \mathbb{R}_{>0}$ uniformly from $\left[\frac{2\log(2p\varepsilon^{-1})}{\varepsilon \mathsf{opt}[G, \varphi, S]}, \frac{4\log(2p\varepsilon^{-1})}{\varepsilon \mathsf{opt}[G, \varphi, S]}\right]$. For each $i \in [p]$, we denote $\mathsf{opt}_i = \mathsf{opt}[G, \varphi_i] + w(S_i)$. By definition, we have $\max_{i \in [p]} \mathsf{opt}_i = \mathsf{opt}[\mathsf{fg}_B(G), \varphi]$. Then, we take $\boldsymbol{i}^* = \mathsf{softargmax}^{\varepsilon}_{i \in [p]} \mathsf{opt}_i$ and define $\mathsf{DP}[\mathsf{fg}_B(G), \varphi] := \mathsf{DP}[G, \varphi_{\boldsymbol{i}^*}] \cup S_{\boldsymbol{i}^*}$. First we analyze the approximation ratio.

▶ **Lemma 22.** *Let $0 < \alpha \leq 1$ and suppose $\mathbb{E}[w(\mathsf{DP}[G, \varphi_i])] \geq \alpha \mathsf{opt}[G, \varphi_i]$ holds for all $i \in [p]$. Then, we have $\mathsf{DP}[\mathsf{fg}_B(G), \varphi] \geq (1 - \varepsilon)\alpha \mathsf{opt}[\mathsf{fg}_B(S), \varphi]$.*

Proof. We have

$$\begin{aligned}
\mathbb{E}[w(\mathsf{DP}[G, \varphi])] &= \mathbb{E}[w(\mathsf{DP}[G, \varphi_{\boldsymbol{i}^*}] \cup S_{\boldsymbol{i}^*})] \geq \mathbb{E}[\alpha \mathsf{opt}[G, \varphi_{\boldsymbol{i}^*}] + w(S_{\boldsymbol{i}^*})] \geq \alpha \mathbb{E}[\mathsf{opt}[G, \varphi_{\boldsymbol{i}^*}] + w(S_{\boldsymbol{i}^*})] \\
&= \alpha \mathbb{E}\left[\mathsf{softmax}^{\varepsilon}_{i \in [p]} \mathsf{opt}_i\right] \geq (1 - \varepsilon)\alpha \max_{i \in [p]} \mathsf{opt}_i = (1 - \varepsilon)\alpha \mathsf{opt}[\mathsf{fg}_B(G), \varphi].
\end{aligned}$$

◀

Now we analyze the Lipschitz constant.

▶ **Lemma 23.** *Let $\beta \in \mathbb{R}_{\geq 0}$ and suppose $\mathsf{EM}(\mathsf{DP}_w[G, \varphi_i], \mathsf{DP}_{w+\delta \mathbf{1}_u}[G, \varphi_i]) \leq \beta$ holds for all $i \in [p]$. Then, we have $\mathsf{EM}(\mathsf{DP}_w[\mathsf{fg}_B(G), \varphi], \mathsf{DP}_{w+\delta \mathbf{1}_u}[\mathsf{fg}_B(G), \varphi]) \leq 31\varepsilon^{-1} \log(2p\varepsilon^{-1})\delta + \beta$.*

Proof. We have

$$
\begin{aligned}
&\mathsf{EM}(\mathsf{DP}_w[\mathsf{fg}_B(G), \varphi], \mathsf{DP}_{w+\delta \mathbf{1}_u}[\mathsf{fg}_B(G), \varphi]) \\
&\quad \leq \mathsf{TV}\left(\boldsymbol{i}^*_w, \boldsymbol{i}^*_{w+\delta \mathbf{1}_u}\right)\left(\mathsf{opt}_w[\mathsf{fg}_B(G), \varphi] + \mathsf{opt}_{w+\delta \mathbf{1}_u}[\mathsf{fg}_B(G), \varphi]\right) \\
&\qquad + \max_{i \in [p]} \left(\mathsf{EM}\left(\mathsf{DP}_w[G, \varphi_i] \cup S_i, \mathsf{DP}_{w+\delta \mathbf{1}_u}[G, \varphi_i] \cup S_i\right)\right) \\
&\quad \leq \frac{10\varepsilon^{-1} \log(2p\varepsilon^{-1})\delta}{\mathsf{opt}_w[\mathsf{fg}_B(G), \varphi]}\left(\mathsf{opt}_w[\mathsf{fg}_B(G), \varphi] + \mathsf{opt}_{w+\delta \mathbf{1}_u}[\mathsf{fg}_B(G), \varphi]\right) \\
&\qquad + \max_{i \in [p]} \left(\mathsf{EM}\left(\mathsf{DP}_w[G, \varphi_i], \mathsf{DP}_{w+\delta \mathbf{1}_u}[G, \varphi_i]\right)\right) + \max_{i \in [p]} \left[d_{\mathrm{w}}((S_i, w), (S_i, w + \delta \mathbf{1}_u))\right] \\
&\quad \leq 30\varepsilon^{-1} \log(2p\varepsilon^{-1})\delta + \beta + \delta \leq 31\varepsilon^{-1} \log(2p\varepsilon^{-1})\delta + \beta.
\end{aligned}
$$

◀

3.5 Putting Together

Let $G = (V, E)$ be a graph with treewidth k. From Lemma 8 and Proposition 11, we can compute a HR-parse tree t over $(k+1)$-graph denoting G with height $O(\log k + \log n) \leq O(\log n)$. We perform the dynamic programming algorithm on this parse tree. We have the following.

▶ **Lemma 24.** *Let $\varepsilon \in (0, 1]$ and h be the height of t. Our algorithm outputs a solution $X \in \mathsf{sat}(G, \varphi)$ such that $\mathbb{E}[w(X)] \geq (1 - h\varepsilon)\mathsf{opt}[G, \varphi]$. The Lipschitz constant is bounded by $31h\varepsilon^{-1} \log(2p_{\max}\varepsilon^{-1})$, where $p_{\max}$ is the maximum of p among all update formulas (3) or (4) that the algorithm uses.*

Proof. The approximability bound is obtained by repeatedly applying Lemmas 20 and 22 and $(1-\varepsilon)^h \geq 1 - h\varepsilon$. The Lipschitzness bound is obtained by repeatedly applying Lemmas 21 and 23. ◀

Since $p_{\max}$ is bounded by a function of k and $|\varphi|$, we have the following.

Proof of Theorem 1. The claim follows by substituting ε, h, and k in Lemma 24 with $\frac{\varepsilon}{\Theta(\log n)}$, $O(\log n)$, and $3k + 2$, respectively. ◀

4 Special Cases

4.1 Independent Set

From the proof of Theorem 1, $f(k, \varphi)$ in Theorem 1 is bounded by $\log(2p_{\max})$. Particularly, if $p_{\max}$ is bounded by $2^{O(k)}$, the Lipschitz constant depends linearly on k. Here, we prove Theorem 3 for the maximum weight independent set problem by showing that $p_{\max} \leq 2^{O(k)}$. Similar arguments can be applied to several other problems, such as the minimum weight vertex cover problem and the minimum weight dominating set problem. For a k-graph G and $S \subseteq \mathsf{src}(G)$, let

$$
\varphi_S(X) \equiv \forall x(x \in \mathsf{src}(G) \to \neg x \in X) \wedge \forall x \forall y(((x \in X \vee x \in S) \wedge (y \in X \vee y \in S)) \to \neg \mathsf{adj}(x, y)).
$$

In words, $\mathsf{sat}(G, \varphi_S)$ is the family of subsets X of V such that X is disjoint from $\mathsf{src}(G)$ and $X \cup S$ is an independent set of G. Then, we have

$$\mathsf{sat}(\mathsf{fg}_B(G), \varphi_S) = \biguplus_{S' \subseteq B} \mathsf{sat}(G, \varphi_{S \cup S'}) \boxtimes \{S'\}, \quad \mathsf{sat}(G /\!\!/ H, \varphi_S) = \mathsf{sat}(G, \varphi_S) \boxtimes \mathsf{sat}(H, \varphi_S).$$

Particularly, we have $p_{\max} \leq 2^{O(k)}$ and therefore Theorem 3 is proved.

4.2 Max Ones

Recall that in *max ones problem*, we given a 3CNF formula Φ over a variable set X and a weight function $w : X \to \mathbb{R}_{\geq 0}$, the goal is to find a satisfying assignment $\sigma : X \to \{0, 1\}$ that maximizes the weight $\sum_{x \in \sigma^{-1}(1)} w(x)$. We reduce the problem to a graph problem. Let $G_\Phi = (X \cup \bar{X} \cup \mathcal{C}, E)$ be the graph such that X is the variable set of Φ, $\bar{X} = \{\bar{x}_i : x_i \in X\}$, $\mathcal{C}$ is the clause set of Φ, $\{x_i, \bar{x}_i\} \in E$ for all $x_i \in X$, $\{C_j, x_i\} \in E$ iff clause C_j contains x_i as a positive literal, and $\{C_j, \bar{x}_i\} \in E$ iff clause C_j contains x_i as a negative literal. Let $w'(x) = w(x)$ if $x \in X$, and otherwise $w'(x) = 0$. Note that the treewidth of G_Φ is at most $2|X|$.

Here, the notations $a \in A \cup B$, $a \in A \cap B$, $a \in A \setminus B$, $\forall a \in A\ \psi$, and $\exists a \in A\ \psi$ are syntactic sugar defined in the usual sense. For a k-graph G and $S, D \subseteq \mathsf{src}(G)$, let

$$\varphi_{S,D}(A) \equiv \exists B \begin{bmatrix} & \forall a \in A \cup S\ \forall b \in B\ (a \in X \wedge b \in \bar{X} \wedge \neg \mathsf{adj}(a, b)) \\ \wedge & \forall x \in X \cup \bar{X}\ \forall y \in X \cup \bar{X}\ (\mathsf{adj}(x, y) \to (x \in A \cup S \cup B \vee y \in A \cup S \cup B)) \\ \wedge & \forall s \in \mathsf{src}(G)\ (\neg s \in A) \wedge \forall c \in \mathcal{C} \setminus D\ \exists x \in A \cup S \cup B\ (\mathsf{adj}(x, c)) \end{bmatrix}.$$

The first and the second rows says that $A \cup S$ and B represent the sets $\sigma^{-1}(1)$ and $\sigma^{-1}(0)$ of an assignment σ, respectively. The third row says that A has no sources, and all clauses except in D are satisfied by the σ defined by $A \cup S$ and B. Then, the max ones problem is equivalent to find a vertex set Y maximizes the weight $\sum_{x \in Y} w'(x)$ and satisfies $G_\Phi \models \varphi_{\emptyset,\emptyset}(Y)$. Here, we have

$$\mathsf{sat}(\mathsf{fg}_B(G), \varphi_{S,D}) = \begin{cases} \biguplus_{S' \subseteq B} \mathsf{sat}(G, \varphi_{S \cup S', D}) \boxtimes \{S'\} & \text{if } D \cap B = \emptyset, \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathsf{sat}(G /\!\!/ H, \varphi_{S,D}) = \biguplus_{\substack{D_1 \cap D_2 = D \\ D_1, D_2 \subseteq \mathsf{src}(G)}} \mathsf{sat}(G, \varphi_{S,D_1}) \boxtimes \mathsf{sat}(H, \varphi_{S,D_2}).$$

Particularly, we have $p_{\max} \leq 2^{O(k)}$.

References

1. Brenda Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994. doi:10.1145/174644.174650.
2. Hans Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.
3. Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.
4. Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. doi:10.1017/CBO9780511977619.

5 Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1&2):49–82, 1993. doi:10.1016/0304-3975(93)90064-Z.

6 Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of Boolean constraint satisfaction problems*. SIAM, 2001.

7 Cynthia Dwork. Differential privacy. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4052, pages 1–12, 2006. doi:10.1007/11787006_1.

8 Noah Fleming and Yuichi Yoshida. Sensitivity lower bounds for approximation algorithms. *arXiv preprint arXiv:2411.02744*, 2024. doi:10.48550/arXiv.2411.02744.

9 Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2001. doi:10.1137/S0097539799349948.

10 Tuukka Korhonen. Lower bounds on dynamic programming for maximum weight independent set. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 198 of *LIPIcs*, pages 87:1–87:14, 2021. doi:10.4230/LIPICS.ICALP.2021.87.

11 Stephan Kreutzer. Algorithmic meta-theorems. In *Finite and Algorithmic Model Theory*, volume 379 of *London Mathematical Society Lecture Note Series*, pages 177–270. Cambridge University Press, 2011.

12 Soh Kumabe and Yuichi Yoshida. Average sensitivity of dynamic programming. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1925–1961, 2022. doi:10.1137/1.9781611977073.77.

13 Soh Kumabe and Yuichi Yoshida. Average sensitivity of the knapsack problem. In *European Symposium on Algorithms (ESA)*, volume 244, pages 75:1–75:14, 2022. doi:10.4230/LIPICS.ESA.2022.75.

14 Soh Kumabe and Yuichi Yoshida. Lipschitz continuous algorithms for graph problems. In *Symposium on Foundations of Computer Science (FOCS)*, pages 762–797, 2023. doi:10.1109/FOCS57990.2023.00051.

15 Soh Kumabe and Yuichi Yoshida. Lipschitz continuous allocations for optimization games. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 297, pages 102:1–102:16, 2024. doi:10.4230/LIPICS.ICALP.2024.102.

16 Soh Kumabe and Yuichi Yoshida. Lipschitz continuous algorithms for covering problems. In *Symposium on Discrete Algorithms (SODA)*, pages 1057–1093, 2025. doi:10.1137/1.9781611978322.31.

17 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

18 Quanquan Liu, Grigoris Velegkas, Yuichi Yoshida, and Felix Zhou. Pointwise lipschitz continuous graph algorithms via proximal gradient analysis. *arXiv preprint*, 2024. doi:10.48550/arXiv.2405.08938.

19 Shogo Murai and Yuichi Yoshida. Sensitivity analysis of centralities on unweighted networks. In *The world wide web conference*, pages 1332–1342, 2019. doi:10.1145/3308558.3313422.

20 Pan Peng and Yuichi Yoshida. Average sensitivity of spectral clustering. In *ACM SIGKDD international conference on knowledge discovery & data mining (KDD)*, pages 1132–1140, 2020. doi:10.1145/3394486.3403166.

21 Nithin Varma and Yuichi Yoshida. Average sensitivity of graph algorithms. *SIAM Journal on Computing*, 52(4):1039–1081, 2023. doi:10.1137/21M1399592.

22 Yuichi Yoshida and Shinji Ito. Average sensitivity of euclidean k-clustering. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:32487–32498, 2022.

23 Yuichi Yoshida and Samson Zhou. Sensitivity analysis of the maximum matching problem. In *Innovations in Theoretical Computer Science Conference (ITCS)*, volume 185, pages 58:1–58:20, 2021. doi:10.4230/LIPIcs.ITCS.2021.58.

Subtrajectory Clustering and Coverage Maximization in Cubic Time, or Better

Jacobus Conradi
University of Bonn, Germany

Anne Driemel
University of Bonn, Germany

Abstract

Many application areas collect unstructured trajectory data. In subtrajectory clustering, one is interested to find patterns in this data using a hybrid combination of segmentation and clustering. We analyze two variants of this problem based on the well-known SetCover and CoverageMaximization problems. In both variants the set system is induced by metric balls under the Fréchet distance centered at polygonal curves. Our algorithms focus on improving the running time of the update step of the generic greedy algorithm by means of a careful combination of sweeps through a candidate space. In the first variant, we are given a polygonal curve P of complexity n, distance threshold Δ and complexity bound ℓ and the goal is to identify a minimum-size set of center curves $\mathcal{C}$, where each center curve is of complexity at most ℓ and every point p on P is covered. A point p on P is covered if it is part of a subtrajectory π_p of P such that there is a center $c \in \mathcal{C}$ whose Fréchet distance to π_p is at most Δ. We present an approximation algorithm for this problem with a running time of $\mathcal{O}((n^2\ell + \sqrt{k_\Delta}n^{5/2}) \log^2 n)$, where k_Δ is the size of an optimal solution. The algorithm gives a bicriterial approximation guarantee that relaxes the Fréchet distance threshold by a constant factor and the size of the solution by a factor of $\mathcal{O}(\log n)$. The second problem variant asks for the maximum fraction of the input curve P that can be covered using k center curves, where $k \leq n$ is a parameter to the algorithm. For the second problem variant, our techniques lead to an algorithm with a running time of $\mathcal{O}((k+\ell)n^2 \log^2 n)$ and similar approximation guarantees. Note that in both algorithms $k, k_\Delta \in O(n)$ and hence the running time is cubic, or better if $k \ll n$.

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

Keywords and phrases Clustering, Set cover, Fréchet distance, Approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.12

Related Version *Full Version*: https://arxiv.org/abs/2504.17381

Funding *Jacobus Conradi*: Partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 313421352 (FOR 2535 Anticipating Human Behavior) and the iBehave Network: Sponsored by the Ministry of Culture and Science of the State of North Rhine-Westphalia. Affiliated with Lamarr Institute for Machine Learning and Artificial Intelligence.
Anne Driemel: Affiliated with Lamarr Institute for Machine Learning and Artificial Intelligence.

1 Introduction

Trajectory analysis is a broad field ranging from gait analysis of full-body motion trajectories [16, 20, 13], via traffic analysis [21] and epidemiological hotspot detection [25] to Lagrangian analysis of particle simulations [24]. Using the current availability of cheap tracking technology, vast amounts of unstructured spatio-temporal data are collected. Clustering is a fundamental problem in this area and can be formulated under various similarity metrics, among them the Fréchet distance. However, the unstructured nature of trajectory data poses the additional challenge that the data first needs to be segmented before the trajectory segments, which we call subtrajectories, can be clustered in a metric space. Subtrajectory clustering aims to find clusters of subtrajectories that represent complex patterns that reoccur along the trajectories [1, 8, 17], such as commuting patterns in traffic data.


© Jacobus Conradi and Anne Driemel;
licensed under Creative Commons License CC-BY 4.0



We study two variants of a problem posed by Akitaya, Brüning, Chambers, and Driemel [3], which are based on the well-known set cover and coverage maximization problems. Both rely on the definition of a geometric set system built using metric balls under the Fréchet distance. Given a polygonal curve, we are asked to produce a set of "center" curves that either

1. has minimum cardinality and covers the entirety of the input curve, or
2. covers as much as possible of the input curve under cardinality constraints.

In either setting a point p of the input-curve is considered as "covered", if there is a subcurve π_p of the input curve that contains p and has small Fréchet distance to some center curve. Note that a point p may be covered by multiple center curves. For the precise formulation refer to Section 2. This formulation extends immediately to a set of input curves – instead of one – where one is now tasked to cover the entirety (respectively as much as possible) of all points on all input curves combined.

In this paper we study the continuous variant of the problem where we need to cover the continuous set of points given by the input polygonal curve. It is instructive, however, to first consider a discrete variant of the set cover instance as follows. Given a discrete point sequence $A = a_1, \ldots, a_n$, consider the set of contiguous subsequences of the form $A_{ij} = a_i, \ldots, a_j$. We say a_k is contained in the set defined by A_{ij} if there exist $i' \leq k \leq j'$ such that $A_{i'j'}$ is similar to A_{ij}. Note that a full specification of this set system has size $\Theta(n^3)$ in the worst case. This phenomenon is shared with the continuous setting where all known discretizations lead to a cubic-size set system. In this paper, we examine the question of whether we can obtain subcubic-time algorithms by internally representing the set system implicitly.

2 Preliminaries

An **edge** is a map $e : [0,1] \to \mathbb{R}^d$ defined by two points $p, q \in \mathbb{R}^d$ as the linear interpolation $t \mapsto p + t(q-p)$. A **polygonal curve** is a map $P : [0,1] \to \mathbb{R}^d$ defined by an ordered set $(p_1, \ldots, p_n) \subset \mathbb{R}^d$ as the result of concatenating the edges $(e_1, \ldots, e_{n-1})$, where e_i is the edge from p_i to p_{i+1}. Any point $P(t)$ is said to lie on the edge e_i if t lies in the preimage of e_i. Any point $P(t)$ lies on at most two edges, exactly if it defines the endpoint of edge e_i which coincides with the start point of e_{i+1}. The number of points n defining the polygonal curve P is its **complexity**, and we denote it by $|P|$. The set of all polygonal curves in $\mathbb{R}^d$ of complexity at most ℓ we denote by $\mathbb{X}_\ell^d$. For any polygonal curve P and values $0 \leq s \leq t \leq 1$ we define $P[s,t]$ to be the subcurve of P from s to t, which itself is a polygonal curve of complexity at most $|P|$. For two polygonal curves P and Q we define the **continuous Fréchet distance** to be $\mathrm{d}_{\mathcal{F}}(P,Q) = \inf_{f,g} \max_{t\in[0,1]} \|P(f(t)) - Q(g(t))\|$ where f and g range over all non-decreasing surjective functions from $[0,1]$ to $[0,1]$.

▶ **Definition 1** (Δ-coverage [3]). *Let P be a polygonal curve in $\mathbb{R}^d$, and let $\Delta \in \mathbb{R}_{>0}$ and $\ell \in \mathbb{N}_{\geq 2}$ be given. For any $\mathcal{Q} \subset \mathbb{X}_\ell^d$, define the Δ-coverage of $\mathcal{Q}$ on P (refer to Figure 1) as*

$$\mathrm{Cov}_P(\mathcal{Q}, \Delta) = \bigcup_{Q\in\mathcal{Q}} \left(\bigcup_{0\leq s\leq t\leq 1,\, \mathrm{d}_{\mathcal{F}}(P[s,t],Q)\leq\Delta} [s,t] \right) \subset [0,1].$$

For a curve $Q \in \mathbb{X}_\ell^d$ we denote the Δ-coverage $\mathrm{Cov}_P(\{Q\}, \Delta)$ by $\mathrm{Cov}_P(Q, \Delta)$.

Problem 1: Subtrajectory Covering (SC). Let P be a polygonal curve in $\mathbb{R}^d$, and let $\Delta \in \mathbb{R}_{>0}$ and $\ell \in \mathbb{N}_{\geq 2}$ be given. The task is to identify a set $\mathcal{Q}$ of curves in $\mathbb{X}_\ell^d$ minimizing $|\mathcal{Q}|$ such that $\mathrm{Cov}_P(\mathcal{Q}, \Delta) = [0,1]$.

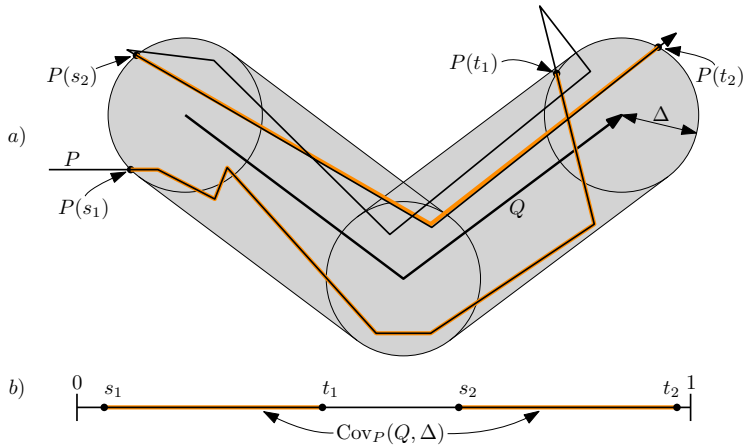


Figure 1 *a*): Example of all points on P that lie on subcurve of P that have Fréchet distance at most Δ to a curve Q of complexity 3. *b*): The set $\mathrm{Cov}_P(Q,\Delta) \subset [0,1]$.

▶ **Definition 2** (Bicriteria approximation for SC). *Let P be a polygonal curve in $\mathbb{R}^d$, and let $\Delta \in \mathbb{R}_{>0}$ and $\ell \in \mathbb{N}_{\geq 2}$ be given. Let $C^* \subset \mathbb{X}_\ell^d$ be a set of minimum cardinality such that $\mathrm{Cov}_P(C^*,\Delta) = [0,1]$. Let $\alpha, \beta \geq 1$ be given. An algorithm that outputs a set $C \subset \mathbb{X}_\ell^d$ of size $\alpha|C^*|$ such that $\mathrm{Cov}_P(C,\beta\Delta) = [0,1]$ is called an (α,β)-approximation for SC.*

Problem 2: Subtrajectory Coverage Maximization (SCM). Let P be a polygonal curve in $\mathbb{R}^d$, and let $\Delta \in \mathbb{R}_{>0}$, $\ell \in \mathbb{N}_{\geq 2}$ and $k \in \mathbb{N}_{\geq 1}$ be given. The task is to identify a set $\mathcal{Q}$ of k curves in $\mathbb{X}_\ell^d$ maximizing the Lebesgue measure $\|\mathrm{Cov}_P(\mathcal{Q},\Delta)\|$.

▶ **Definition 3** (Bicriteria approximation for SCM). *Let P be a polygonal curve in $\mathbb{R}^d$, and let $\Delta \in \mathbb{R}_{>0}$ and $\ell \in \mathbb{N}_{\geq 2}$ be given. Let $C^* \subset \mathbb{X}_\ell^d$ be a set of size k such that $\|\mathrm{Cov}_P(C^*,\Delta)\|$ is maximum. Let $\alpha \geq 0$ and $\beta \geq 1$ be given. An algorithm that outputs a set $C \subset \mathbb{X}_\ell^d$ of size k such that $\|\mathrm{Cov}_P(C,\beta\Delta)\| \geq \alpha \|\mathrm{Cov}_P(C^*,\Delta)\|$ is called an (α,β)-approximation for SCM.*

2.1 Prior work on Subtrajectory Covering and Coverage Maximization

We first discuss prior work on the SC problem. The SC problem was introduced by Akitaya, Brüning, Chambers, and Driemel [3]. Their results were later improved by Brüning, Conradi, and Driemel [4]. Both works identified a curve S that approximates the input P, such that any solution to the problem induces an approximate solution of similar size consisting of only subcurves of S. This set of subcurves defines a set system which turns out to have low VC-dimension. This enables randomized set cover algorithms with improved approximation factors and expected running time in $\mathcal{O}(n(k_\Delta)^3(\log^4(\Lambda/\Delta) + \log^3(n/k_\Delta)) + n\log^2 n)$, where Λ corresponds to the arc length of the curve P. While the complexity of the main algorithm in [3] depends on the relative arclength of the input curve, the latter [4] also identified a set of points on S that define $\mathcal{O}(n^3)$ subcurves of complexity 2 which induce the set-system of the SetCover instance resulting in an expected running time of $\tilde{\mathcal{O}}(k_\Delta n^3)$, where $\tilde{\mathcal{O}}(\cdot)$ hides polylogarithmic factors. The approximation quality of both approaches depends on ℓ, k_Δ as well as on the VC-dimension of the set system. A drawback of these algorithms is that they generate cluster centers that are line segments only and that the algorithm is randomized with large constants in the approximation factors. Conradi and Driemel [9] took a different approach by focusing on the greedy set cover algorithm, which successively adds the candidate

curve maximizing the added coverage to a partial solution until the entirety of $[0,1]$ is covered. Their algorithm computes $\mathcal{O}(n^3\ell)$ candidate curves that have complexity up to ℓ instead of 2. Applying the greedy algorithm, this results in a deterministic $(\mathcal{O}(\log n), 11)$-approximation algorithm with running time $\tilde{\mathcal{O}}(k_\Delta n^4\ell + n^4\ell^2)$. Recently, van der Hoog, van der Horst and Ophelders showed [23] that the cardinality of the candidate curve set can be further reduced to a size of $\mathcal{O}(n^2 \log n)$. They improve the quality of the approximating curve S resulting in a $(\mathcal{O}(\log n), 4)$-approximation with running time in $\tilde{\mathcal{O}}(k_\Delta n^3)$.

As for the SCM problem, by standard sub-modular function maximization arguments [19, 15], the greedy algorithm used in all these approaches [3, 4, 9, 23] yields an $(\mathcal{O}(1), \mathcal{O}(1))$-approximation. Using this reduction, the candidate curve set identified in [23] yields a $(\mathcal{O}(1), 4)$-approximation to the SCM problem with a running time of $\tilde{\mathcal{O}}(kn^3)$.

2.2 Structure and contributions

In Section 3 we describe how to obtain a set of $\tilde{\mathcal{O}}(n^2)$ candidate center curves $\mathcal{C}$ based on known transformations [23]. Unlike the recent line of work discussed above [3, 4, 9, 23], we do not improve any further on the size of the set $\mathcal{C}$. Instead, our focus lies on the update step of the greedy set cover algorithm. In this step, we have to find the element $c \in \mathcal{C}$ that maximizes the coverage that is added to a partial solution. In Section 4 we describe our first contribution: a partition of the set $\mathcal{C}$ into few ordered lists, called **sweep-sequences**, which we each process via a sweep algorithm computing their coverage reusing prior computations. Unfortunately, the coverage does not immediately allow efficient maintenance along a sweep-sequence. Here, we introduce our second contribution: a candidate-specific approximation of the coverage, called the **proxy coverage**. In Section 5, we describe how sweep-sequences allow efficient maintenance of the proxy coverage and with Theorem 26 describe the culmination of our techniques enabling our algorithm: for a weighted set $A \subset [0,1]$ of points we are able to compute the total weight of points that lie inside the proxy coverage of every element in the sweep-sequence in logarithmic time per element.

In Section 6 we present two $(\mathcal{O}(\log n), 4)$-approximation algorithms for Subtrajectory Covering. The algorithms discretize the ground-set $[0,1]$ by the arrangement (of size $\tilde{\mathcal{O}}(n^3)$) of the intervals representing the coverage of each element in $\mathcal{C}$. Computing the arrangement together with standard greedy set cover techniques and the subroutine from Theorem 26 results in a first approximation algorithm for SC with running time in $\tilde{\mathcal{O}}(|\mathcal{C}|n + k_\Delta|\mathcal{C}|) = \tilde{\mathcal{O}}(n^3)$ (see Corollary 30). Our third contribution is a sampling technique improving the running time: we identify a representative subset (comparable to a sampling) of size roughly $\mathcal{O}(\sqrt{k_\Delta} n^{3/2})$ of the arrangement. This enables a faster algorithm by first covering the representative subset and with it almost the entire arrangement in an initial pass. The remaining $\mathcal{O}(\sqrt{k_\Delta} n^{5/2})$ elements of the arrangement are then identified and covered resulting in the following theorem.

▶ **Theorem 4.** *There is a* $(96\ln(n) + 128, 4)$*-approximation for SC. Given a polygonal curve* P *of complexity* n*,* $\Delta > 0$ *and* $\ell \leq n$*, its running time is in* $\mathcal{O}\left(\left(n^2\ell + \sqrt{k_\Delta} n^{\frac{5}{2}}\right)\log^2 n\right)$*, where* k_Δ *is the size of the smallest subset* $C^* \subset \mathbb{X}_\ell^d$ *such that* $\mathrm{Cov}_P(C^*, \Delta) = [0,1]$.

As trivially $k_\Delta \leq \lceil \frac{n}{\ell} \rceil$, Theorem 4 yields an algorithm with (near-)cubic running time. Further, in the case that $k_\Delta \in \mathcal{O}(n^{1-\varepsilon})$, it yields an algorithm with subcubic running time. This compares favorably to the best known algorithm with running time $\tilde{\mathcal{O}}(k_\Delta n^3)$ [23].

In Section 7, we sketch how our techniques can be applied to obtain faster algorithms for the SCM problem which compares favorably to the best known algorithm with running time $\tilde{\mathcal{O}}(kn^3)$ [23]. All omitted proofs and discussions can be found in the full version.

▶ **Theorem 5.** *Let $\varepsilon \in \left(0, \frac{1}{5}\right]$. There is a $(\frac{e-1}{16e}, 4+\varepsilon)$-approximation algorithm for SCM, where e is the base of the natural logarithm. Given a polygonal curve P of complexity n, $\Delta > 0$, $\ell \leq n$ and $k > 0$, its running time is in $\mathcal{O}((k+\ell)n^2\varepsilon^{-2}\log^2 n \log \varepsilon^{-1})$.*

2.3 Related work

Prior to the line of work on the SC and SCM problem discussed in Section 2.1, Buchin et al. [7] presented one of the earlier works on subtrajectory clustering for both the discrete and continuous Fréchet distance. Their work focuses on finding the largest subtrajectory cluster, where different variants of "largest" are considered. They present hardness results for $(2-\varepsilon)$-approximations for any ε and a matching polynomial time 2-approximation algorithm. Gudmundsson and Wong [12] present a cubic lower bound conditioned on SETH for the same problem and show that this lower bound is tight. In subsequent experimental work [11, 5, 6, 8] the formulation from [7] was studied, but these works do not offer any theoretical guarantees.

Agarwal et al. [1] base their formulation of the subtrajectory clustering problem on FacilityLocation. In this formulation there is an opening cost associated with every center curve, a cost for every point on the input that is assigned to a cluster, and a cost for points that are not assigned. They present conditional NP-hardness results and an $O(\log^2 n)$-approximation for well-behaved classes of curves under the discrete Fréchet distance.

3 Set system discretization

In this section we show that, given a curve P of complexity n, there is a curve S of complexity n and a set $\mathcal{C}_S(P)$ of at most $\mathcal{O}(n^2 \log n)$ subcurves of S, each of complexity at most ℓ, such that for any set of curves $C \subset \mathbb{X}_\ell^d$ there is a subset $\hat{C} \subset \mathcal{C}_S(P)$ of size $\mathcal{O}(|C|)$ such that $\mathrm{Cov}_P(C, \Delta) \subset \mathrm{Cov}_P(\hat{C}, 4\Delta)$. This result is known and follows the line of research in [4, 9, 23]. All of these approaches compute a "maximally simplified" version S of P, see also [10]. Our definition of a simplification follows the notion of *pathlet-preserving simplifications* from [23], as it yields the best constants.

▶ **Definition 6.** *Let a polygonal curve P, $\Delta \geq 0$, and $\ell \geq 1$ be given. A curve S is a simplification of P if $\mathrm{d}_{\mathcal{F}}(S, P) \leq 2\Delta$, and for any curve $Q \in \mathbb{X}_\ell^d$ and $P[s,t]$ with $\mathrm{d}_{\mathcal{F}}(Q, P[s,t]) \leq \Delta$ there is a subcurve $S[s',t']$ with $\mathrm{d}_{\mathcal{F}}(P[s,t], S[s',t']) \leq 2\Delta$ and $|S[s',t']| \leq \ell + 2$.*

▶ **Theorem 7** ([23])**.** *For any curve P of complexity n a simplification S of P of complexity $\mathcal{O}(n)$ can be computed in $\mathcal{O}(n \log n)$ time.*

By definition of the simplification S of P and the triangle inequality for any curve $\pi \in \mathbb{X}_\ell^d$ there is a subcurve $\hat{\pi}$ of S of complexity at most $\ell + 2$ such that $\mathrm{Cov}_P(\pi, \Delta) \subset \mathrm{Cov}_P(\hat{\pi}, 4\Delta)$.

▶ **Definition 8** (Free space diagram)**.** *Let S and P be two polygonal curves parametrized over $[0,1]$. The free space diagram of S and P is defined as the joint parametric space $[0,1]^2$ together with a not necessarily uniform grid, where each vertical grid line corresponds to a vertex of P and each horizontal grid line to a vertex of S. The Δ-**free space** (refer to Figure 2) of S and P is defined as*

$$\mathcal{D}_\Delta(S, P) = \left\{(x, y) \in [0,1]^2 \mid \|P(x) - S(y)\| \leq \Delta\right\}.$$

*This is the set of points in the parametric space, whose corresponding points on S and P are at a distance at most Δ. The edges of S and P segment the free space diagram into cells. We call the intersection of $\mathcal{D}_\Delta(S, P)$ with the boundary of cells the **free space intervals**.*

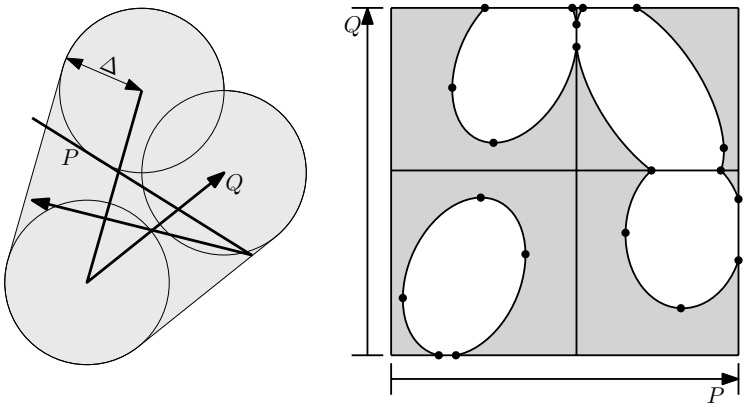


Figure 2 Illustration of the Δ-free space of two curves P and Q as well as their extremal points.

As was noted by Alt and Godau [2], the Fréchet distance of S and P is at most Δ if and only if there is a monotone (in both x- and y-direction) path from $(0,0)$ to $(1,1)$ in Δ-free space of S and P. They further observed that the free space inside any cell is described by an ellipse intersected with the cell and thus is convex and has constant complexity. Observe that two subcurves $P[a,c]$ and $S[b,d]$ have Fréchet distance at most Δ if and only if there is a monotone (in both x- and y-direction) path from (a,b) to (c,d) in Δ-free space of S and P.

▶ **Definition 9** (Extremal points [4]). *As the Δ-free space of two curves S and P is convex in any cell C, the set of points of $\mathcal{D}_\Delta(S,P)$ minimizing the x-coordinate in C are described by either a single point or a vertical line segment. We call either the single point or the start and end point of the line segment the leftmost points of $\mathcal{D}_\Delta(S,P)$ in C. Similarly C has at most two bottommost, rightmost and topmost points. The set of all these at most eight points of $\mathcal{D}_\Delta(S,P)$ in every cell defines the set of extremal points of $\mathcal{D}_\Delta(S,P)$.*

Similar to [23], we define specific subcurves via the extremal points of the 4Δ-free space.

▶ **Definition 10** ((I)-, (II)- and (III)-subcurves). *Let P be a polygonal curve and let S be a simplification of P. Let Δ and ℓ be given. Define the following three types of subcurves of S via the (sorted) set $\mathcal{E}(S,P)$ of y-coordinates of extremal points in $\mathcal{D}_\Delta(S,P)$. For every edge e let $\mathcal{E}(e,P)$ be the subset of $\mathcal{E}(S,P)$ corresponding to the extremal points in $\mathcal{D}_\Delta(e,P)$ (refer to Figure 3).*

- **(I):** *A (I)-subcurve of S is a subcurve of S that starts at some ith vertex of S and ends at the $(i+j)$th vertex for $j \in \{2^m \mid 0 \le m \le \lfloor \log_2(\ell) \rfloor\}$.*
- **(II):** *A (II)-subcurve of S is either a subcurve $e[0,\varepsilon_i]$ or $e[\varepsilon_i,1]$ of an edge e of S defined by a vertex of e and a value $\varepsilon_i \in \mathcal{E}(e,P)$ or its reversal* $\mathrm{rev}(e[0,\varepsilon_i])$ *or* $\mathrm{rev}(e[\varepsilon_i,1])$.
- **(III):** *A (III)-subcurve of S is either a subcurve $e[\varepsilon_i,\varepsilon_{i+j}]$ of an edge e of S defined by two values $\varepsilon_i,\varepsilon_{i+j} \in \mathcal{E}(e,P)$ such that $j \in \{2^m \mid 0 \le m \le \lfloor \log_2(|\mathcal{E}(e,P)|) \rfloor\}$, or its reversal* $\mathrm{rev}(e[\varepsilon_i,\varepsilon_{i+j}])$.

The set containing all Type (I)-, (II)- and (III)-subcurves which we denote by $\mathcal{C}_S(P)$ describes the set of candidate curves similar to the candidate set from [23].

▶ **Theorem 11** (adapted from [23]). *Let P be a polygonal curve, and let S be a simplification of P. Then for any curve $\pi \in \mathbb{X}_\ell^d$ there is a set $S_\pi \subset \mathcal{C}_S(P)$ of size at most 8 such that*

$$\mathrm{Cov}_P(\pi,\Delta) \subset \bigcup_{s \in S_\pi} \mathrm{Cov}_P(s,4\Delta).$$

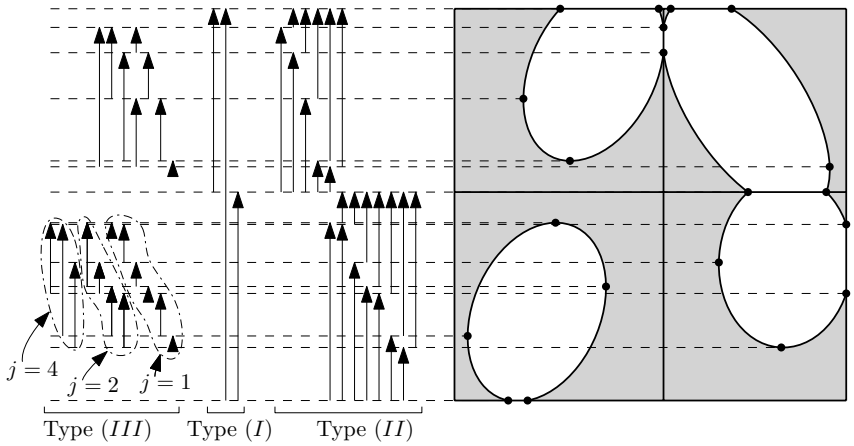


Figure 3 Illustration of all Type (I)-, (II)- and (III)-subcurves of Q (as vertical lines) that are not reversals, induced by the 4Δ-free space of Q and P from Figure 2. Further denoted are the values j, which induced the set of Type (III)-subcurves on the first edge of Q.

4 Sweep-sequences and the proxy coverage

In this section we define an ordering of Type (II) and Type (III) subcurves in the set $\mathcal{C}_S(P)$ and introduce an approximate Δ-coverage called the **proxy coverage**. Later on, we show that a coarse representation of the proxy coverage can be maintained along the ordering.

Throughout this section we are given a polygonal curve P of complexity n, values $\Delta > 0$ and $\ell \leq n$, as well as a simplification S of P. As $|\mathcal{E}(e, P)| = \mathcal{O}(n)$ for any edge e, there are $\mathcal{O}(n \log \ell)$ Type (I)-, $\mathcal{O}(n^2)$ Type (II)- and $\mathcal{O}(n^2 \log n)$ Type (III)-subcurves in $\mathcal{C}_S(P)$.

▶ **Definition 12** (Sweep-sequence). *Let $E = (e_1, \ldots)$ be a sorted list of values in $\mathbb{R}$. We say $\mathfrak{s}$ is a sweep-sequence of E if $\mathfrak{s}$ is a list of tuples of E such that either for all consecutive tuples (e_a, e_b) and (e_c, e_d) it holds that $a \leq b$, $c \leq d$, $0 \leq a - c \leq 1$ and $0 \leq b - d \leq 1$ or for all consecutive tuples it holds that $a \geq b$, $c \geq d$, $0 \leq c - a \leq 1$ and $0 \leq d - b \leq 1$.*

We are able to construct few sweep-sequences that already capture the entirety of $\mathcal{C}_S(P)$.

▶ **Lemma 13.** *Let e be an edge of S. In time $\mathcal{O}(n \log n)$, one can compute a set $\mathfrak{S}_e$ of $\mathcal{O}(\log n)$ sweep-sequences of $\mathcal{E}(e, P)$, each of length $\mathcal{O}(n)$, such that for any Type (II)- or Type (III)-subcurve $\pi \in \mathcal{C}_S(P)$ on the edge e there is a tuple $(\varepsilon_i, \varepsilon_j)$ in one of the sweep-sequences in $\mathfrak{S}_e$ such that $\pi = e[\varepsilon_i, \varepsilon_j]$ if $\varepsilon_i \leq \varepsilon_j$ (refer to Figure 4), and $\pi = \mathrm{rev}(e[\varepsilon_j, \varepsilon_i])$ if $\varepsilon_i > \varepsilon_j$. Further, for the first and last pair (e_a, e_b) in every sweep-sequence it holds that $a = b$.*

We focus our analysis on sweep-sequences of $\mathcal{E}(e, P)$ where for every tuple $(\varepsilon_a, \varepsilon_b)$ it holds that $a \leq b$ (refer to Figure 4). This analysis carries over to all sweep-sequences of $\mathcal{E}(e, P)$ by setting $e \leftarrow \mathrm{rev}(e)$, and thus to all sweep-sequences in $\mathfrak{S}_e$.

We now turn to defining the proxy coverage on P which approximates the 4Δ-coverage.

▶ **Definition 14.** *Let e be an edge of S. Define $l_i(\cdot)$ to be the function mapping any y to the x-coordinate of the leftmost point of the ith cell in $\mathcal{D}_{4\Delta}(e, P)$ at height y. If this point does not exist, $l_i(y) = \infty$. Similarly define $r_j(y)$ to be the x-coordinate of the rightmost point at height y, and ∞ otherwise.*

We assume that for every edge e and every $i < n$ the functions $l_i(\cdot)$ and $r_i(\cdot)$ of the free space $\mathcal{D}_\Delta(e, P)$ can be evaluated in $\mathcal{O}(1)$ time.

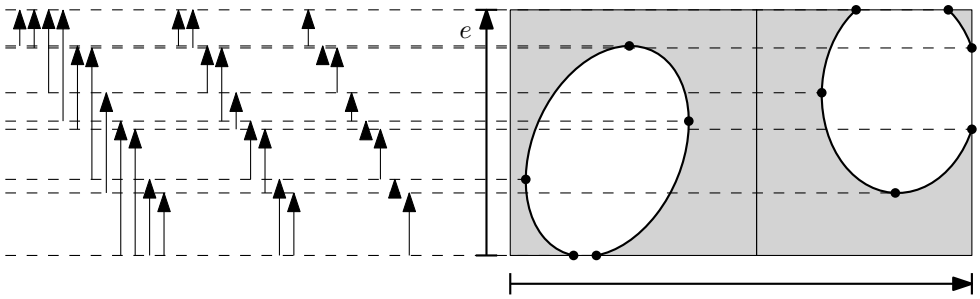


Figure 4 Illustration of three of the eight sweep-sequences in $\mathfrak{S}_e$ of the edge e that are constructed for Type (III)-subcurves. From left to right one for $j = 4$, one for $j = 2$ and one for $j = 1$.

▶ **Definition 15** (Combinatorial representation). *Let e be an edge of S and let $0 \leq s \leq t \leq 1$ be given. The combinatorial representation $\mathcal{R}(e[s,t])$ of the 4Δ-coverage $\mathrm{Cov}_P(e[s,t], 4\Delta)$ of $e[s,t]$ is the set of all inclusionwise-maximal pairs of indices (i,j) such that there are points s' and t' on the ith and jth edge of P respectively such that there is a monotone path from (s',s) to (t',t) in $\mathcal{D}_\Delta(e,P)$. An index pair (i,j) includes (i',j') if $i \leq i'$ and $j' \leq j$.*

The combinatorial representation can be partitioned into two sets, the global group $\mathcal{G}(e[s,t])$ consisting of all index-pairs $(i,j) \in \mathcal{R}(e[s,t])$ such that $i < j$ and the local group $\mathcal{L}(e[s,t])$ consisting of all index-pairs $(i,i) \in \mathcal{R}(e[s,t])$.

▶ **Observation 16.** *Let $e[s,t]$ be a subcurve of an edge e of S. Then (refer to Figure 5)*

$$\mathrm{Cov}_P(e[s,t], 4\Delta) = \bigcup_{(i,j)\in\mathcal{R}(e[s,t])} [l_i(s), r_j(t)] = \bigcup_{(i,j)\in\mathcal{L}(e[s,t])\cup\mathcal{G}(e[s,t])} [l_i(s), r_j(t)]$$

Let an edge e of S together with $0 \leq s \leq t \leq 1$ be given. We say an index i is bad for $e[s,t]$, if all topmost points in cell i of the free space of e and P are to the left of both $l_i(s)$ and $r_i(t)$, and both $l_i(s)$ and $r_i(t)$ are to the left of all bottom most points in cell i. Call this set of bad indices $\mathcal{B}(e[s,t])$. If $i \notin \mathcal{B}(e[s,t])$, i is said to be good for $e[s,t]$. Intuitively, an index is bad if the free-space inside the cell is a diagonal from the top left to the bottom right. For Lemma 21 to hold, the definition a bad index is stronger and depends on s and t.

▶ **Observation 17.** *Let i be good for $e[s,t]$ and P. If $l_i(s) \neq \infty \neq r_i(t)$ then $l_i(s) \leq r_i(t)$.*

We now turn to defining our central tool, the proxy coverage, which approximates the Δ-coverage and is easy to maintain. It is defined via a reduced combinatorial representation.

▶ **Definition 18** (Reduced global group). *Let e be an edge of S and let $0 \leq s \leq t \leq 1$ be given. Based on the global group $\mathcal{G}(e[s,t])$ we define the reduced global group $\widetilde{\mathcal{G}}(e[s,t])$ which results from $\mathcal{G}(e[s,t])$ after merging all pairs of index pairs (a,b) and (c,d) if either $a < c < b < d$, or $b = c$ and $b = c$ is good for e (see Figure 5).*

▶ **Definition 19** (Proxy coverage). *For edge e of S, $0 \leq s \leq t \leq 1$ and $1 \leq i,j < n$ define*

$$\hat{l}_{i,e[s,t]}(s) = \begin{cases} l_i(s), & \text{if } i \text{ is good for } e[s,t] \\ r_i(s), & \text{if } i \text{ is bad for } e[s,t] \end{cases} \qquad \hat{r}_{j,e[s,t]}(t) = \begin{cases} l_j(t), & \text{if } j \text{ is good for } e[s,t] \\ r_j(t), & \text{if } j \text{ is bad for } e[s,t]. \end{cases}$$

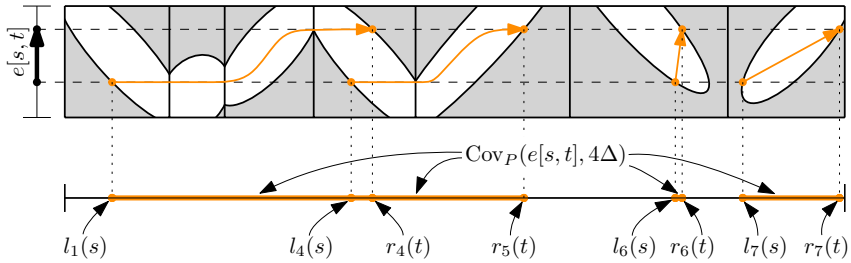


Figure 5 Illustration of the 4Δ-coverage of $e[s,t]$. The global group of $e[s,t]$ is $\{(1,4),(4,5)\}$. The reduced global group of $e[s,t]$ is $\{(1,4),(4,5)\}$ also, as 4 is bad for $e[s,t]$.

With these at hand, define the ***proxy coverage*** *of subedges of S as the union*

$$\widehat{\mathrm{Cov}}_P(e[s,t]) = \left(\bigcup_{(i,i)\in\mathcal{L}(e[s,t])\setminus\mathcal{B}(e[s,t])} [l_i(s), r_i(t)] \right) \cup \left(\bigcup_{(i,j)\in\widetilde{\mathcal{G}}(e[s,t])} [\hat{l}_{i,e[s,t]}(s), \hat{r}_{j,e[s,t]}(t)] \right),$$

where by a slight abuse of notation let $\mathcal{L}(e[s,t]) \setminus \mathcal{B}(e[s,t])$ be all index-pairs (i,i) in $\mathcal{L}(e[s,t])$ such that i is not in $\mathcal{B}(e[s,t])$.

We observe that the proxy coverage can be expressed as a disjoint union via $\widetilde{\mathcal{G}}(e[s,t])$.

▶ **Lemma 20.** *Let e be an edge and let $0 \le s \le t \le 1$. Then $\widehat{\mathrm{Cov}}_P(e[s,t])$ is the disjoint union (refer to Figure 6)*

$$\widehat{\mathrm{Cov}}_P(e[s,t]) = \left(\bigsqcup_{i\in\mathcal{L}(e[s,t])\setminus\mathcal{B}(e[s,t])} [l_i(s), r_i(t)] \right) \sqcup \left(\bigsqcup_{(i,j)\in\widetilde{\mathcal{G}}(e[s,t])} [\hat{l}_{i,e[s,t]}(s), \hat{r}_{j,e[s,t]}(t)] \right).$$

▶ **Lemma 21.** *If i is bad for $e[s,t]$, then i is good for $\mathrm{rev}(e[s,t])$ and $l_i(s), r_i(t) \in [l_i(t), r_i(s)]$.*

▶ **Lemma 22.** *Let e be an edge of S and let $0 \le s \le t \le 1$ be given. Then (refer to Figure 6)*

$$\mathrm{Cov}_P(e[s,t], 4\Delta) \subset \widehat{\mathrm{Cov}}_P(e[s,t]) \cup \widehat{\mathrm{Cov}}_P(\mathrm{rev}(e[s,t])) \subset \mathrm{Cov}_P(\{e[s,t], \mathrm{rev}(e[s,t])\}, 4\Delta).$$

We extend the proxy coverage $\widehat{\mathrm{Cov}}$ to also be defined for Type (I)-subcurves of S, where we set $\widehat{\mathrm{Cov}}_P(S[s,t]) = \mathrm{Cov}_P(S[s,t], 4\Delta)$. Overall, we see that for any $\pi \in \mathcal{C}_S(P)$ – not just Type (II)- and (III)-subcurves of S – there are at most two elements $\pi_1, \pi_2 \in \mathcal{C}_S(P)$ with

$$\mathrm{Cov}_P(\pi, 4\Delta) \subset \widehat{\mathrm{Cov}}_P(\pi_1) \cup \widehat{\mathrm{Cov}}_P(\pi_2) \subset \mathrm{Cov}_P(\{\pi_1, \pi_2\}, 4\Delta).$$

▶ **Theorem 23.** *Let P be a polygonal curve, and let S be a simplification of P. Then for any curve $\pi \in \mathbb{X}_\ell^d$ there is a set $S_\pi \subset \mathcal{C}_S(P)$ of size at most 16 such that*

$$\mathrm{Cov}_P(\pi, \Delta) \subset \bigcup_{s\in S_\pi} \widehat{\mathrm{Cov}}_P(s).$$

Proof. This follows from the definition of the proxy coverage, Lemma 22 and Theorem 11. ◀

5 Maintaining the proxy coverage along a sweep-sequence

In this section we present an algorithm that given $\mathfrak{s} \in \mathfrak{S}_e$ maintains a symbolic representation of $\widehat{\mathrm{Cov}}_P(e[s,t])$ in (roughly) logarithmic time per element in $\mathfrak{s}$. To this end we maintain the set $\mathcal{U}(e[s,t])$ of inclusion-wise maximal index-pairs (i,j) such that there is a path from cell i at height s to cell j at some height $\hat{t} \leq t$, which helps maintain $\widetilde{\mathcal{G}}(e[s,t])$ and $\mathcal{L}(e[s,t]) \setminus \mathcal{B}(e[s,t])$. We want to highlight that maintaining $\mathcal{G}(e[s,t])$ and $\mathcal{L}(e[s,t])$ as a symbolic representation of the 4Δ-Coverage can take total time in $\Theta(n|\mathfrak{S}_e|)$ (Figure 7) motivating our techniques.

▶ **Theorem 24.** *At the beginning and end of the loop in Lines 7 – 15 of* MAINTAIN *of Algorithm 1, the set of index-pairs $\widetilde{\mathcal{G}}$ coincides with $\widetilde{\mathcal{G}}(e[s,t])$, $\mathcal{L}$ coincides with $\mathcal{L}(e[s,t]) \setminus \mathcal{B}(e[s,t])$ and $\mathcal{U}$ coincides with $\mathcal{U}(e[s,t])$. Further, executing* MAINTAIN *takes $\mathcal{O}(n \log n)$ time.*

▶ **Corollary 25.** *Let $\mathfrak{s} \in \mathfrak{S}_e$ be a sweep-sequence. There are $m = \mathcal{O}(n)$ (not necessarily distinct) index-pairs $p_1 = (i_1, j_1), \ldots, p_m = (i_m, j_m)$ together with m contiguous subsets $I_i = \{(s_{a_i}, t_{a_i}), \ldots, (s_{b_i}, t_{b_i})\} \subset \mathfrak{s}$ such that for every $(s,t) \in \mathfrak{s}$ it holds that*

$$\widetilde{\mathcal{G}}(e[s,t]) \cup (\mathcal{L}(e[s,t]) \setminus \mathcal{B}(e[s,t])) = \bigcup_{1 \leq i \leq m, (s,t) \in I_i} \{p_i\}.$$

Further for $p_k = (i_k, j_k)$ the index i_k is either always good or always bad for $e[s,t]$ for $(s,t) \in I_k$, and the index j_k is either always good or always bad for $e[s,t]$ for $(s,t) \in I_k$. The index-pair p_i as well as the values a_i and b_i can be computed for all i in total time $\mathcal{O}(n \log n)$.

Overall, we can maintain a symbolic representation of $\widehat{\mathrm{Cov}}_P(\cdot)$ during a sweep-sequence in total time $\mathcal{O}(n \log n)$ which enables the use of the maintained sets for batch point-queries.

▶ **Theorem 26.** *Let $\mathfrak{s} \in \mathfrak{S}_e$, and let $Q \subset [0,1]$ together with $w_Q : Q \to \mathbb{N}$ be a weighted point-set. For any $Q' \subset Q$ let $w_Q(Q') = \sum_{q \in Q'} w_q(q)$ denote its weight. There is an algorithm which computes $w_Q(Q \cap \widehat{\mathrm{Cov}}_{A_S}(e[s,t]))$ for every $(s,t) \in \mathfrak{s}$ in $\mathcal{O}(|Q| \log |Q| + n \log n)$ time.*

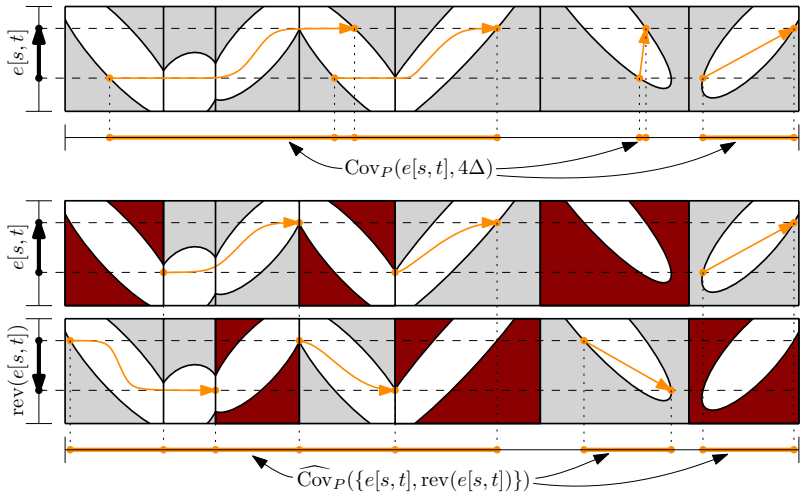


Figure 6 Illustration of the proxy coverage of $e[s,t]$ and $\mathrm{rev}(e[s,t])$ compared to the 4Δ-coverage of $e[s,t]$. Cells with bad index for $e[s,t]$ and $\mathrm{rev}(e[s,t])$ are highlighted in red.

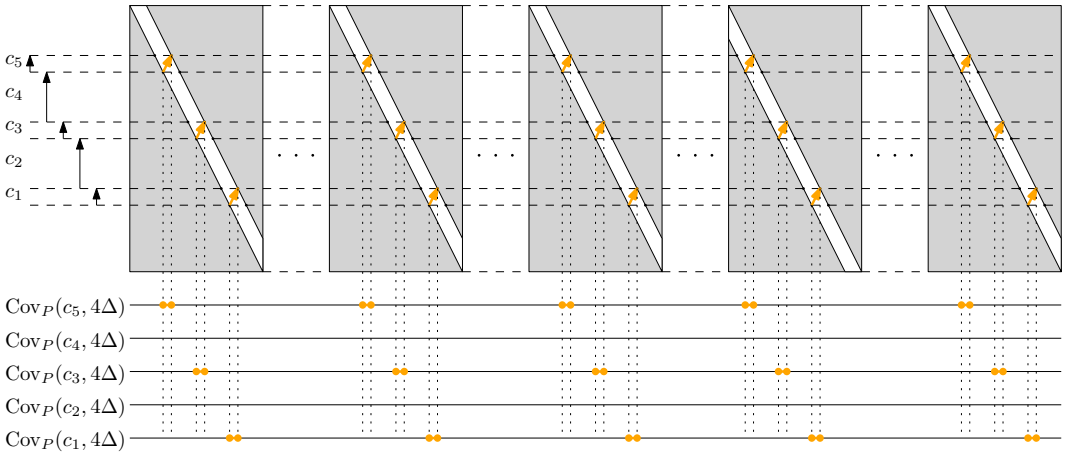


Figure 7 Illustration of how the combinatorial description of the 4Δ-coverage of two neighbouring elements in a sweep-sequence may differ by up to n index-pairs. Instead, $\widehat{\mathrm{Cov}}(c_i) = \emptyset$ for all i.

Algorithm 1 Maintenance of the reduced global group.

procedure MAINTAIN($\mathcal{D}_{4\Delta}(e, P)$, sweep-sequence $\mathfrak{s} = \{(s_1, t_1), \ldots, (s_m, t_m)\}$ of $\mathcal{E}(e, P)$)
  for $i < n$ **do** compute the contiguous sets $B_i = \{(s, t) \in \mathfrak{s} \mid i \text{ is bad for } e[s, t]\}$
  Compute coverage $\mathrm{Cov}_P(e[s_m, t_m], 4\Delta)$ represented as $\mathcal{O}(n)$ disjoint intervals
  Compute $\mathcal{L}(e[s_m, t_m])$ and $\mathcal{G}(e[s_m, t_m])$ from $\mathrm{Cov}_P(e[s_m, t_m], 4\Delta)$
  $\mathcal{U}, \mathcal{L}, \mathcal{G}, (s, t) \leftarrow \mathcal{L}(e[s_m, t_m]) \cup \mathcal{G}(e[s_m, t_m]), \mathcal{L}(e[s_m, t_m]), \mathcal{G}(e[s_m, t_m]), (s_m, t_m)$
  Compute $\mathcal{B}(e[s, t])$ from B_i and with it $\widetilde{\mathcal{G}} \leftarrow \widetilde{\mathcal{G}}(e[s_m, t_m])$
  for (s', t') in REVERSED(S) **do** ▷ sweeping window events
    if $s' \neq s$ **then** ▷ $s' = s - 1$
      Compute $\mathcal{B}(e[s', t]) \setminus \mathcal{B}(e[s, t])$ and $\mathcal{B}(e[s, t]) \setminus \mathcal{B}(e[s', t])$ via the sets B_i
      Update $\mathcal{U}, \mathcal{L}, \mathcal{G}$ and $\widetilde{\mathcal{G}}$ via ADVANCESTART(s, s', t) from Algorithm 2
      Update $\mathcal{B}(e[s, t])$ to $\mathcal{B}(e[s', t])$ via the sets B_i, and $s \leftarrow s'$
    if $t' \neq t$ **then** ▷ $t' = t - 1$
      Compute $\mathcal{B}(e[s, t']) \setminus \mathcal{B}(e[s, t])$ and $\mathcal{B}(e[s, t]) \setminus \mathcal{B}(e[s, t'])$ via the sets B_i
      Update $\mathcal{U}, \mathcal{L}, \mathcal{G}$ and $\widetilde{\mathcal{G}}$ via ADVANCEEND(s, t, t') from Algorithm 2
      Update $\mathcal{B}(e[s, t])$ to $\mathcal{B}(e[s', t])$ via the sets B_i, and $t \leftarrow t'$

Proof. Define following values for all i, j along the sweep-sequence $\mathfrak{s}$ where for $(s, t) \in \mathfrak{s}$:

$$L_i((s,t)) = \sum_{\substack{q \in Q,\ q \text{ in cell } i \\ \hat{l}_{i,e[s,t]}(s) \neq \infty,\ \hat{l}_{i,e[s,t]}(s) \leq q}} w_Q(q), \qquad R_j((s,t)) = \sum_{\substack{q \in Q,\ q \text{ in cell } j \\ \hat{r}_{j,e[s,t]}(t) \neq \infty,\ q \leq \hat{r}_{j,e[s,t]}(t)}} w_Q(q),$$

$$M_i((s,t)) = \sum_{\substack{q \in Q,\ i \notin \mathcal{B}(e[s,t]),\ q \text{ in cell } i \\ l_i(s) \neq \infty,\ r_i(t) \neq \infty,\ l_i(s) \leq q \leq r_i(t)}} w_Q(q), \qquad D(i,j) = \sum_{\substack{q \in Q,\ q \text{ in cell } m \\ i \leq m \leq j}} w_Q(q).$$

The value $L_i((s, t))$ corresponds to the weight of points in cell i that lie right of $\hat{l}_{i,e[s,t]}(s)$. Similarly, $R_i((s, t))$ corresponds to the weight of points in cell i that lie left of $\hat{r}_{i,e[s,t]}(t)$. The value $M_i((s, t))$ corresponds to the weight of points in cell i that lie in between $\hat{l}_{i,e[s,t]}(s)$ and $\hat{r}_{i,e[s,t]}(t)$ if i is good for $e[s, t]$. Lastly $D(i, j)$ corresponds to the total weight of points that lie on edge $i, \ldots, j$. Then for any $(s, t) \in \mathfrak{s}$ and any $(i, i) \in \mathcal{L}(e[s, t]) \setminus \mathcal{B}(e[s, t])$ it holds that $M_i((s, t)) = w_Q(Q \cap [l_i(s), r_i(t)])$. Similarly, for any $(i, j) \in \widetilde{\mathcal{G}}(e[s, t])$ it holds that

Algorithm 2 Advancing the start/end point of the sweeping window.

procedure AdvanceStart(s,s',t)
  Get $\mathcal{U}$, $\mathcal{L}$, $\mathcal{G}$, $\widetilde{\mathcal{G}}$, $\mathcal{B}(e[s,t])$, $\mathcal{B}(e[s',t]) \setminus \mathcal{B}(e[s,t])$, $\mathcal{B}(e[s,t]) \setminus \mathcal{B}(e[s',t])$ and $\mathcal{D}_{4\Delta}(e,P)$
  if s is induced by a lowest point in cell i of $\mathcal{D}_{4\Delta}(e,P)$ **then**
    Identify smallest $j > i$ such that $l_j(s) \neq \infty$
    Update index-pairs in $\mathcal{U}$ and in $\mathcal{G}$ starting at i to start at j
  if s' is induced by a upper boundary of free space interval of cell i of $\mathcal{D}_{4\Delta}(e,P)$ **then**
    Identify largest $j < i$ such that $l_j(s)$ is in the interior of cell j
    Update index-pairs in $\mathcal{U}$ and in $\mathcal{G}$ starting at i to start at j
  if s' is induced by a highest point in cell i of $\mathcal{D}_{4\Delta}(e,P)$ **then** add (i,i) to $\mathcal{U}$
  for changed index-pair (i,j) in $\mathcal{U}$ and $\mathcal{G}$ **do**
    if $i \neq j$ and $r_j(t) \neq \infty$ **then** move index-pair to $\mathcal{G}$, else move it to $\mathcal{U}$
    Remove any dominated index-pairs involving (i,j) from $\mathcal{G}$ and $\mathcal{U}$
  Update $\widetilde{\mathcal{G}}$ from $\mathcal{G}$ after merging overlapping intervals via $\mathcal{B}(e[s,t])$ and $\mathcal{B}(e[s',t])$
  Update $\mathcal{L}$ via the changes to $\mathcal{U}$, $\mathcal{B}(e[s,t])$ and $\mathcal{B}(e[s,t'])$
procedure AdvanceEnd(s,t,t')
  dual to AdvanceStart; refer to full version in appendix

$L_i((s,t)) + D(i+1,j-1) + R_j((s,t)) = w_Q(Q \cap [\hat{l}_{i,e[s,t]}(s), \hat{r}_{i,e[s,t]}(t)])$. Then, by Lemma 20

$$
\begin{aligned}
&w_Q(Q \cap \widehat{\mathrm{Cov}}_{A_S}(e[s,t])) = \\
&= \left(\sum_{i \in \mathcal{L}(e[s,t]) \setminus \mathcal{B}(e[s,t])} w_Q(Q \cap [l_i(s), r_i(t)]) \right) + \left(\sum_{(i,j) \in \widetilde{\mathcal{G}}(e[s,t])} w_Q(Q \cap [\hat{l}_{i,e[s,t]}(s), \hat{r}_{j,e[s,t]}(t)]) \right) \\
&= \left(\sum_{i \in \mathcal{L}(e[s,t]) \setminus \mathcal{B}(e[s,t])} M_i((s,t)) \right) + \left(\sum_{(i,j) \in \widetilde{\mathcal{G}}(e[s,t])} L_i((s,t)) + D(i+1,j-1) + R_j((s,t)) \right).
\end{aligned}
$$

Observe that $D(i,j)$ can be provided via a data-structure that first computes $d_i = \sum_{q \in Q, q \text{ in cell } i} w_Q(q)$ for every i in total time $\mathcal{O}(|Q| \log n)$ and stores them in a balanced binary tree as leaves, where every inner node stores the sum of the values of its children. For every $i \leq j$ the value $D(i,j) = \sum_{i \leq m \leq j} d_m$ can then be returned in $\mathcal{O}(\log n)$ time by identifying in $\mathcal{O}(\log n)$ time all $\mathcal{O}(\log n)$ maximal subtrees whose children lie in the interval $[i,j]$ and then returning the sum of the stored values in the root of each maximal subtree.

Next we show that $L_i(\cdot)$ (resp. $R_j(\cdot)$ and $M_i(\cdot)$) can correctly be maintained when performing the sweep of $\mathfrak{s}$. To this end let

$$I_q = \{(s,t) \in \mathfrak{s} \mid q \text{ is in cell } i \text{ and } \hat{l}_{i,e[s,t]}(s) \neq \infty, \hat{l}_{i,e[s,t]} \leq q\}.$$

Refer to Figure 8. As the free space in every cell is convex, throughout $\mathfrak{s}$ the first indices are monotone and the y-coordinates of the left-most and right-most points are stored in $\mathcal{E}(A_e)$, for every $q \in Q$ the contiguous subsets

$$L_q = \{(s,t) \in \mathfrak{s} \mid q \text{ is in cell } i \text{ and } l_i(s) \neq \infty, l_i(s) \leq q\} \text{ and}$$
$$R_q = \{(s,t) \in \mathfrak{s} \mid q \text{ is in cell } i \text{ and } l_i(s) \neq \infty, r_i(s) \geq q\}$$

can be computed in $\mathcal{O}(\log n)$ time. Similarly the set $B_i = \{(s,t) \in \mathfrak{s} \mid i \text{ is bad for } e[s,t]\}$ is also a contiguous subset of $\mathfrak{s}$ and can be computed in $\mathcal{O}(\log n)$ time. Let Q_i denote all $q \in Q$ that lie on the ith edge of P. Then for any $q \in Q_i$

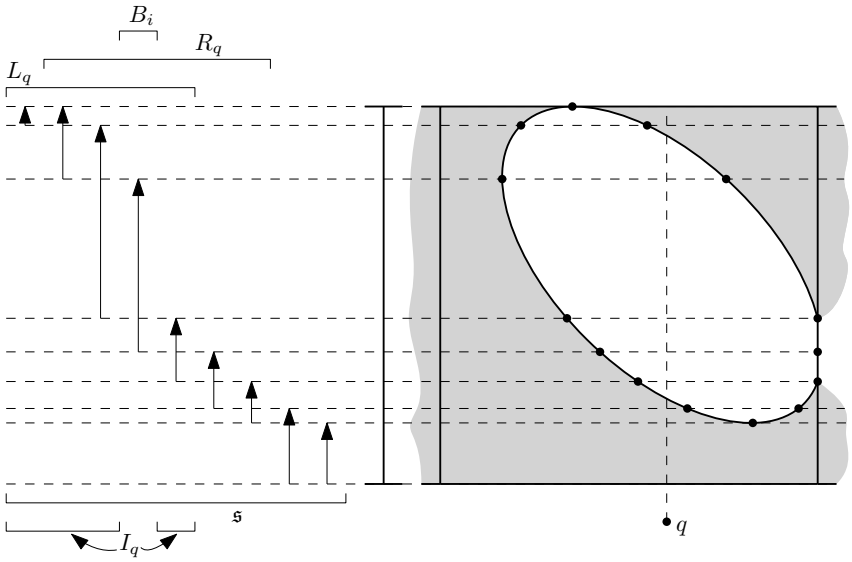


Figure 8 Construction of the set I_q encoding when $\hat{l}_{i,e[s,t]}(s) \leq q$ from Proof of Theorem 26 via the sets L_q and R_q encoding when $l_i(s) \leq q$ and $r_i(s) \geq q$ and the set B_i for all $(s,t) \in \mathfrak{s}$.

$$I_q = (L_q \cap (\mathfrak{s} \setminus B_i)) \cup ((\mathfrak{s} \setminus R_q) \cap B_i),$$

and thus I_q consists of $\mathcal{O}(1)$ contiguous disjoint subsets of $\mathfrak{s}$. Thus all sets I_q (represented as $\mathcal{O}(1)$ contiguous subsets of $\mathfrak{s}$) can be computed in time $\mathcal{O}(|Q| \log n + n \log n)$. Further,

$$L_i((s,t)) = \sum_{q \in Q_i} \mathbb{1}_{q \in I_q} w_Q(q),$$

and hence $L_i(\cdot)$ can be maintained in total time $\mathcal{O}(|Q| + n \log n)$ after one initial computation of all I_q, by adding $w_Q(q)$ for $q \in Q_i$ whenever (s,t) enters the $\mathcal{O}(1)$ contiguous disjoint subsets of I_q, and subtracting $w_Q(q)$ for $q \in Q_i$ whenever (s,t) exits the $\mathcal{O}(1)$ contiguous disjoint subsets of I_q. Sorting the boundaries of all I_q preparing them for the maintenance of $L_i(\cdot)$ takes $\mathcal{O}(|Q| \log |Q|)$ time. Similarly $M_i(\cdot)$ and $R_j(\cdot)$ can be maintained.

Overall, the values $L_i(\cdot)$, $R_j(\cdot)$, $M_i(\cdot)$ and $D(i,j)$ are correctly maintained in total time $\mathcal{O}(|Q| \log |Q|)$ time such that they can be evaluated in $\mathcal{O}(\log n)$. By Corollary 25 there are only $\mathcal{O}(n)$ total updates to $\widetilde{\mathcal{G}}(\cdot)$ and $\mathcal{L}(\cdot) \setminus \mathcal{B}(\cdot)$. Hence, $w_Q(\cdot)$ can be correctly maintained along $\mathfrak{s}$ by updating it whenever $L_i(\cdot)$, $R_j(\cdot)$, $M_i(\cdot)$, $\widetilde{\mathcal{G}}(\cdot)$, $\mathcal{L}(\cdot) \setminus \mathcal{B}(\cdot)$ or $\mathcal{B}(\cdot)$ change. Thus computing $w_Q(e[s,t])$ for all $(s,t) \in S$ takes $\mathcal{O}(|Q| \log |Q| + n \log n)$ time. ◀

6 Ground-set discretization and sampling

We now present two $(\mathcal{O}(\log n), 4)$-approximation algorithm for SC that uses the efficient batch point queries from Theorem 26.

▶ **Definition 27** (Atomic intervals). *Let P be a polygonal curve, and let $\Delta > 0$ and ℓ be given. Let S be a simplification of P. Let G be the set of all intersection-points of horizontal lines at height h for y-coordinates $h \in \mathcal{E}(S,P)$ of extremal points of $\mathcal{D}_{4\Delta}(S,P)$ with the boundary of the free space $\mathcal{D}_{4\Delta}(S,P)$. From this, define the set of atomic intervals $\mathcal{A}(S,P)$ as the set of intervals describing the arrangement of $[0,1]$ defined by the set $\{x \in [0,1] \mid \exists y \in [0,1] : (x,y) \in G\}$ (Figure 9).*

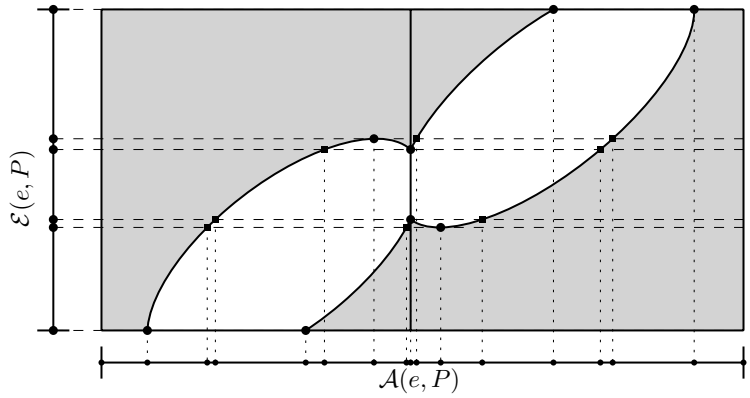


Figure 9 Illustration of atomic intervals $\mathcal{A}(e,P)$ induced by $\mathcal{E}(e,P)$.

▶ **Observation 28.** *As $|\mathcal{E}(S,P)| \leq 8n^2$, and each horizontal line intersects at most n cells, and the free space in every cell is convex it follows, the set of all midpoints of atomic intervals $\mathcal{A}(S,P)$ is a point set $A \subset [0,1]$ of size $16n^3$ such that for any $C \subset \mathcal{C}_S(P)$ it holds that*

$$\widehat{\mathrm{Cov}}_P(C) = [0,1] \iff A \subset \widehat{\mathrm{Cov}}_P(C).$$

▶ **Lemma 29.** *Let P be a polygonal curve of complexity n and let $\Delta > 0$ and $\ell \leq n$ be given. Let S be a simplification of P. Let $\mathcal{E}(S,P)$ be the extremal points of $\mathcal{D}_{4\Delta}(S,P)$. Let $A \subset [0,1]$ be a given set of size at most cn^3 points. For every edge e of S let W_e be the set of atomic intervals a in $\mathcal{A}(e,P)$ such that $w_A(a) = |A \cap a| \neq 0$. Let k_Δ be the size of the smallest set $\mathcal{C}^* \subset \mathbb{X}_\ell^d$ such that $\mathrm{Cov}_P(\mathcal{C}^*, \Delta) = [0,1]$. There is an algorithm that computes a set $\mathcal{C} \subset \mathcal{C}_S(P)$ of size $16(3\ln(n) + \ln c + 1)k_\Delta$ such that $A \subset \widehat{\mathrm{Cov}}_{A_S}(\mathcal{C})$ in time*

$$\mathcal{O}\left(n^2 \ell \log^2 n + |A| \log n + k_\Delta n^2 \log^3 n + \log n \sum_e |W_e|\right).$$

Proof Sketch. The algorithm operates in rounds. In every round we start with a subset A' of A, where initially $A' = A$, and in every round for the partial solution R of the greedy algorithm it holds that $A' = A \setminus \widehat{\mathrm{Cov}}_P(R)$. Further in every round and for every edge e of S we maintain the set W'_e of atomic intervals a in $\mathcal{A}(e,P)$ such that $w_{A'}(a) = |A' \cap a| \neq 0$. These weights in combination with Theorem 26 allow us to identify the Type (II)- or (III)-subedge r of S maximizing $|A' \cap \widehat{\mathrm{Cov}}_P(r)| = |(A \setminus \widehat{\mathrm{Cov}}_P(R)) \cap \widehat{\mathrm{Cov}}_P(r)|$. We similarly identify the Type (I)-curve r maximizing $|A' \cap \widehat{\mathrm{Cov}}_P(r)|$ by initially computing $\widehat{\mathrm{Cov}}_P(r)$ for every Type (I)-curve and then in every round recomputing $|A' \cap \widehat{\mathrm{Cov}}_P(r)|$. Afterwards we add the subcurve maximizing $|A' \cap \widehat{\mathrm{Cov}}_P(r)|$ to R. Lastly we update $A' \leftarrow A' \setminus \widehat{\mathrm{Cov}}_P(r)$. This results in almost all intervals in W'_e whose weights are updated, to have their weights set to zero and are thus removed from W'_e. The number of updates setting such a weight to zero, together with the initial application of Theorem 26 dominate the running time. Lastly, by standard greedy SetCover arguments [14, 22, 18] and Theorem 23 the number of rounds until the algorithm terminates ($A' = \emptyset$) is bounded by $16(\ln(|A|) + 1)$. ◀

▶ **Corollary 30.** *There is an $(48\ln(n) + 64, 4)$-approximation algorithm for SC. Given a polygonal curve P of complexity n, $\Delta > 0$ and $\ell \leq n$, its running time is in $\mathcal{O}\left(n^3 \log^3 n\right)$.*

Proof. Lemma 29 together with Observation 28 and Theorem 23 yields the claim. ◀

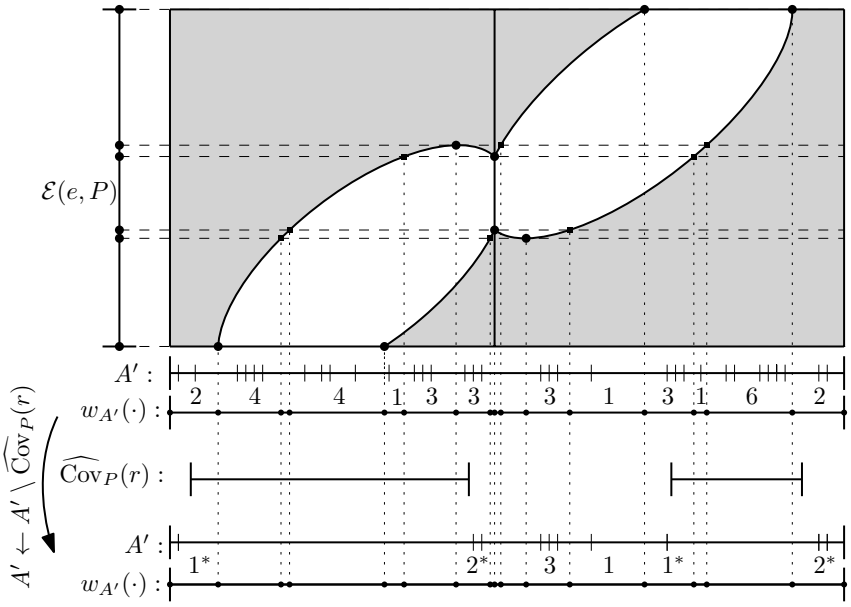


Figure 10 Illustration to Proof Sketch of Lemma 29. Modification of non-zero weights when adding r to R. All weights that are updated but not necessarily set to 0 are highlighted with a *.

The running time of Corollary 30 is dominated by the size of $\mathcal{A}(S,P)$. We now present techniques enabling identification of a representative subset of $\mathcal{A}(S,P)$.

▶ **Theorem 31.** *Let n lists L_i be given, each containing m_i sorted values such that all values are distinct and in every list L_i and for every $j \leq m_i$ identifying the item at position j takes $\mathcal{O}(\log m_i)$ time and also for given v determining the maximal index j such that the jth item is less than v takes $\mathcal{O}(\log m_i)$ time. Then for every $K \leq \sum_i m_i$ in $\mathcal{O}(Kn \log \max_i m_i)$ time one can determine K values $v_1 < \ldots < v_K$ such that*

1. *for every $x \in \bigcup_i L_i$ there is an i such that $x \in [v_i, v_{i+1}]$ and*
2. $|[v_i, v_{i+1}] \cap \bigcup_i L_i| = O\left(\frac{\sum_i m_i}{K}\right)$ *for all i.*

Proof Sketch. We recursively compute a value m such that both a constant fraction of values among all lists is smaller than m and a constant fraction of values among all lists is bigger than m. To find m we compute the middle element of every list in $\mathcal{O}(n \log \max_i m_i)$ time. Then m is (roughly) the middle element of these middle elements. ◀

▶ **Lemma 32.** *For every $\alpha \in [0,3]$ in $\mathcal{O}(n^{1+\alpha} \log n)$ time one can determine $\mathcal{O}(n^\alpha)$ so called α-**coarse** intervals partitioning $[0,1]$, each containing at most $\mathcal{O}(n^{3-\alpha})$ intervals of $\mathcal{A}(S,P)$.*

Proof Sketch. By the structure of $\mathcal{D}_{4\Delta}(e,P)$ we can compute how many atomic intervals of $\mathcal{A}(e,P)$ are contained on any edge of P in $\mathcal{O}(\log n)$ time. Similarly, on any edge of P we can compute the ith atomic interval boundary via $\mathcal{O}(1)$ binary searches of $\mathcal{E}(e,P)$. Hence after $\mathcal{O}(n \log n)$ precomputation time per edge e of S, we can output the ith interval boundary of $\mathcal{A}(e,P)$ in $\mathcal{O}(\log n)$ time for any i. Hence Theorem 31 implies the claim. ◀

We now use Lemma 32 together with Lemma 29 to obtain the following lemma and theorem. The overarching approach for the algorithm is as follows: First use Lemma 32 to compute roughly $\mathcal{O}(n^{3/2})$ points from $\mathcal{A}(S,P)$. These points we cover with a set R_1 of

$\tilde{\mathcal{O}}(k_\Delta)$ elements from $\mathcal{C}_S(P)$ in time $\tilde{\mathcal{O}}(k_\Delta n^{5/2})$ via Lemma 29. Between any two consecutive points of the $\mathcal{O}(n^{3/2})$ points there are at most $\mathcal{O}(n^{3/2})$ other points from $\mathcal{A}(S,P)$, and at most $\tilde{\mathcal{O}}(k_\Delta n)$ of these sets are **not** covered by R_1. This leaves us with $\tilde{\mathcal{O}}(k_\Delta n^{5/2})$ uncovered points in $\mathcal{A}(S,P)$. We again invoke Lemma 29, covering these remaining points with a set R_2 of $\tilde{\mathcal{O}}(k_\Delta)$ elements from $\mathcal{C}_S(P)$ in time $\tilde{\mathcal{O}}(k_\Delta n^{5/2})$.

▶ **Lemma 33.** *Let $\alpha \in [0,3]$ and $K > 0$ be given. Let $C \subset \mathcal{C}_S(P)$ be a set covering all midpoints of α-coarse intervals of size $\mathcal{O}(K \log n)$. Then there are $\mathcal{O}(Kn^{4-\alpha} \log n)$ atomic intervals in $\mathcal{A}(S,P)$ and $\mathcal{O}(Kn^{4-\alpha} \log n + Kn^2 \log n)$ atomic intervals in $\bigcup_e \mathcal{A}(e,P)$ that are not contained in $\widehat{\mathrm{Cov}}_{A_S}(C)$. They can be computed in $\mathcal{O}(Kn^{4-\alpha} \log^2 n + Kn^2 \log^2 n)$ time.*

▶ **Theorem 4.** *There is a $(96\ln(n)+128, 4)$-approximation for SC. Given a polygonal curve P of complexity n, $\Delta > 0$ and $\ell \leq n$, its running time is in $\mathcal{O}\left(\left(n^2\ell + \sqrt{k_\Delta} n^{\frac{5}{2}}\right) \log^2 n\right)$, where k_Δ is the size of the smallest subset $C^* \subset \mathbb{X}_\ell^d$ such that $\mathrm{Cov}_P(C^*, \Delta) = [0,1]$.*

Proof Sketch. The algorithm maintains an internal guess K of k_Δ, which initially is 1. We describe a subroutine that decides, whether $K < k_\Delta$ or outputs a solution of size $\mathcal{O}(K \log n)$. If the subroutine outputs that $K < k_\Delta$ we set $K \leftarrow 2K$, which then implies the claim.

First set $\alpha = \frac{3}{2} + \frac{\log K}{2 \log n} + \frac{\log \log n}{\log n}$. Next compute a set of α-coarse intervals via Lemma 32. From them compute their midpoints M and for every edge e of S compute the subset W_e of $\mathcal{A}(e,P)$ containing such a midpoint. Then W_e is the set of atomic intervals in $\mathcal{A}(e,P)$ with $w_M(\cdot) \neq 0$. The algorithm then attempts to cover the mid-points of these intervals via $\mathcal{O}(K \log n)$ rounds of the algorithm from Lemma 29. If the algorithm does not terminate after $\mathcal{O}(K \log n)$ rounds we return that $K < k_\Delta$. Otherwise we compute the set of atomic intervals $A \subset \mathcal{A}(S,P)$ and for every e the set of atomic intervals in $W'_e \subset \mathcal{A}(e,P)$ that are not fully contained in the solution returned from the first call of the algorithm from Lemma 29 via Lemma 33. We then invoke the algorithm from Lemma 29 on the midpoints M' of A and W'_e which is precisely the set of atomic intervals in $\mathcal{A}(e,P)$ with $w_{M'}(\cdot) \neq 0$. As $K \in \mathcal{O}(k_\Delta)$ and thus $K \in \mathcal{O}(n)$, the running time of the subroutine is

$$\mathcal{O}(n^{1+\alpha} \log n + Kn^{4-\alpha} \log^3 n + Kn^2 \log^2 n) = \mathcal{O}(K^{\frac{1}{2}} n^{\frac{5}{2}} \log^2 n).$$

And thus overall the running time of the algorithm is

$$\mathcal{O}\left(n^2 \ell \log^2 n + \sum_{K=1}^{\log(k_\Delta)} \left(\left(2^K\right)^{\frac{1}{2}} n^{\frac{5}{2}} \log^2 n\right)\right) = \mathcal{O}\left(n^2 \ell \log^2 n + \sqrt{k_\Delta} n^{\frac{5}{2}} \log^2 n\right). \quad \blacktriangleleft$$

7 Subtrajectory Coverage Maximization

The techniques discussed in this paper can also be used to obtain the following result.

▶ **Theorem 5.** *Let $\varepsilon \in \left(0, \frac{1}{5}\right]$. There is a $(\frac{e-1}{16e}, 4+\varepsilon)$-approximation algorithm for SCM, where e is the base of the natural logarithm. Given a polygonal curve P of complexity n, $\Delta > 0$, $\ell \leq n$ and $k > 0$, its running time is in $\mathcal{O}((k+\ell) n^2 \varepsilon^{-2} \log^2 n \log \varepsilon^{-1})$.*

Proof Sketch. We first compute an ε-approximation of the 4Δ-free space, which allows describing $\|\widehat{\mathrm{Cov}}_P(\cdot)\|$ as a sum of linear functions. A sum of linear functions is a linear function itself. As we can maintain a combinatorial description of the proxy coverage we are able to parsimoniously maintain this linear function during a scan of a sweep-sequence in $\mathfrak{S}_e$, which allows evaluating $\|\widehat{\mathrm{Cov}}_P(\cdot)\|$ for all elements in a single sweep-sequence in time roughly $\mathcal{O}(n \log n)$. Using this subroutine once per round and sweep-sequence results in a running time of $\mathcal{O}(kn^2 \log^2 n)$. $\blacktriangleleft$

References

1 Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, Abhinandan Nath, Jiangwei Pan, and Erin Taylor. Subtrajectory clustering: Models and algorithms. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, PODS '18, pages 75–87, 2018. doi:10.1145/3196959.3196972.

2 Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

3 Frederik Brüning, Hugo A. Akitaya, Erin W. Chambers, and Anne Driemel. Subtrajectory clustering: Finding set covers for set systems of subcurves. *Comput. Geom. Topol.*, 2(1):1:1–1:48, February 2023. doi:10.57717/cgt.v2i1.7.

4 Frederik Brüning, Jacobus Conradi, and Anne Driemel. Faster approximate covering of subcurves under the fréchet distance. In *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.28.

5 Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristan, Rodrigo I. Silveira, Frank Staals, and Carola Wenk. Clustering trajectories for map construction. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '17, 2017. doi:10.1145/3139958.3139964.

6 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Jorren Hendriks, Erfan Hosseini Sereshgi, Vera Sacristán, Rodrigo I. Silveira, Jorrick Sleijster, Frank Staals, and Carola Wenk. Improved map construction using subtrajectory clustering. In *LocalRec'20: Proceedings of the 4th ACM SIGSPATIAL Workshop on Location-Based Recommendations, Geosocial Networks, and Geoadvertising, LocalRec@SIGSPATIAL 2020, November 3, 2020, Seattle, WA, USA*, pages 5:1–5:4, 2020. doi:10.1145/3423334.3431451.

7 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry and Applications*, 21(3):253–282, 2011. doi:10.1142/S0218195911003652.

8 Maike Buchin, Bernhard Kilgus, and Andrea Kölzsch. Group diagrams for representing trajectories. *International Journal of Geographical Information Science*, 34(12):2401–2433, 2020. doi:10.1080/13658816.2019.1684498.

9 Jacobus Conradi and Anne Driemel. Finding complex patterns in trajectory data via geometric set cover, 2023. doi:10.48550/arXiv.2308.14865.

10 Mark de Berg, Atlas F. Cook IV, and Joachim Gudmundsson. Fast fréchet queries. *Computational Geometry*, 46(6):747–755, 2013. doi:10.1016/j.comgeo.2012.11.006.

11 Joachim Gudmundsson and Nacho Valladares. A GPU approach to subtrajectory clustering using the fréchet distance. *IEEE Trans. Parallel Distributed Syst.*, 26(4):924–937, 2015. doi:10.1109/TPDS.2014.2317713.

12 Joachim Gudmundsson and Sampson Wong. Cubic upper and lower bounds for subtrajectory clustering under the continuous fréchet distance, 2021. doi:10.48550/arXiv.2110.15554.

13 Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE transactions on pattern analysis and machine intelligence*, 36(7):1325–1339, 2013. doi:10.1109/TPAMI.2013.248.

14 David S Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49, 1973. doi:10.1145/800125.804034.

15 Andreas Krause and Daniel Golovin. Submodular function maximization. In Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli, editors, *Tractability: Practical Approaches to Hard Problems*, pages 71–104. Cambridge University Press, 2014. doi:10.1017/CBO9781139177801.004.

16 Lily Lee and W Eric L Grimson. Gait analysis for recognition and classification. In *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition*, pages 155–162. IEEE, 2002. doi:10.1109/AFGR.2002.1004148.

17 Anqi Liang, Bin Yao, Bo Wang, Yinpei Liu, Zhida Chen, Jiong Xie, and Feifei Li. Subtrajectory clustering with deep reinforcement learning. *VLDB J.*, 33(3):685–702, 2024. doi:10.1007/s00778-023-00833-w.

18 László Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975. doi:10.1016/0012-365X(75)90058-8.

19 G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Mathematical Programming*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

20 Sen Qiao, Y. Wang, and J. Li. Real-time human gesture grading based on OpenPose. *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–6, 2017. doi:10.1109/CISP-BMEI.2017.8301910.

21 Roniel S. De Sousa, Azzedine Boukerche, and Antonio A. F. Loureiro. Vehicle trajectory similarity: Models, methods, and applications. *ACM Comput. Surv.*, 53(5), September 2020. doi:10.1145/3406096.

22 Sherman K Stein. Two combinatorial covering theorems. *Journal of Combinatorial Theory, Series A*, 16(3):391–397, 1974. doi:10.1016/0097-3165(74)90062-4.

23 Ivor van der Hoog, Thijs van der Horst, and Tim Ophelders. Faster and deterministic subtrajectory clustering, 2024. doi:10.48550/arXiv.2402.13117.

24 Erik van Sebille, Stephen M. Griffies, Ryan Abernathey, Thomas P. Adams, Pavel Berloff, Arne Biastoch, Bruno Blanke, Eric P. Chassignet, Yu Cheng, Colin J. Cotter, Eric Deleersnijder, Kristofer Döös, Henri F. Drake, Sybren Drijfhout, Stefan F. Gary, Arnold W. Heemink, Joakim Kjellsson, Inga Monika Koszalka, Michael Lange, Camille Lique, Graeme A. MacGilchrist, Robert Marsh, C. Gabriela Mayorga Adame, Ronan McAdam, Francesco Nencioli, Claire B. Paris, Matthew D. Piggott, Jeff A. Polton, Siren Rühs, Syed H.A.M. Shah, Matthew D. Thomas, Jinbo Wang, Phillip J. Wolfram, Laure Zanna, and Jan D. Zika. Lagrangian ocean analysis: Fundamentals and practices. *Ocean Modelling*, 121:49–75, 2018. doi:10.1016/j.ocemod.2017.11.008.

25 Yiqun Xie, Shashi Shekhar, and Yan Li. Statistically-robust clustering techniques for mapping spatial hotspots: A survey. *ACM Comput. Surv.*, 55(2), January 2022. doi:10.1145/3487893.

Tight Bounds for Some Classical Problems Parameterized by Cutwidth

Narek Bojikian
Humboldt-Universität zu Berlin, Germany

Vera Chekan
Humboldt-Universität zu Berlin, Germany

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany

Abstract

Cutwidth is a widely studied parameter and it quantifies how well a graph can be decomposed along small edge-cuts. It complements pathwidth, which captures decomposition by small vertex separators, and it is well-known that cutwidth upper-bounds pathwidth. The SETH-tight parameterized complexity of problems on graphs of bounded pathwidth (and treewidth) has been actively studied over the past decade while for cutwidth the complexity of many classical problems remained open.

For Hamiltonian Cycle, it is known that a $(2+\sqrt{2})^{\mathrm{pw}} n^{\mathcal{O}(1)}$ algorithm is optimal for pathwidth under SETH [Cygan et al. JACM 2018]. Van Geffen et al. [J. Graph Algorithms Appl. 2020] and Bojikian et al. [STACS 2023] asked which running time is optimal for this problem parameterized by cutwidth. We answer this question with $(1+\sqrt{2})^{\mathrm{ctw}} n^{\mathcal{O}(1)}$ by providing matching upper and lower bounds. Second, as our main technical contribution, we close the gap left by van Heck [2018] for Partition Into Triangles (and Triangle Packing) by improving both upper and lower bound and getting a tight bound of $\sqrt[3]{3}^{\mathrm{ctw}} n^{\mathcal{O}(1)}$, which to our knowledge exhibits the only known tight non-integral basis apart from Hamiltonian Cycle [Cygan et al. JACM 2018] and C_4-Hitting Set [SODA 2025]. We show that the cuts inducing a disjoint union of paths of length three (unions of so-called Z-cuts) lie at the core of the complexity of the problem – usually lower-bound constructions use simpler cuts inducing either a matching or a disjoint union of bicliques. Finally, we determine the optimal running times for Max Cut ($2^{\mathrm{ctw}} n^{\mathcal{O}(1)}$) and Induced Matching ($3^{\mathrm{ctw}} n^{\mathcal{O}(1)}$) by providing matching lower bounds for the existing algorithms – the latter result also answers an open question for treewidth by Chaudhary and Zehavi [WG 2023].

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

Keywords and phrases Parameterized complexity, cutwidth, Hamiltonian cycle, triangle packing, max cut, induced matching

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.13

Related Version *Full Version*: https://arxiv.org/abs/2502.15884 [8]

1 Introduction

In parameterized complexity the (worst-case) complexity of problems is expressed in terms of input size n and one or more parameters, often denoted k. The parameter can, for example, be the size of the sought solution or some measure of the structure of the input. The goal is to understand the influence of solution size or structure on the complexity. A problem is said to be fixed-parameter tractable with parameter k if it admits an algorithm with running time of $f(k) \cdot n^{\mathcal{O}(1)}$ (also denoted by $\mathcal{O}^*(f(k))$) for some computable function f. For NP-hard problems, this function f is usually exponential and it may be doubly exponential or worse.

Due to space constraints, we defer full proofs and technical details to the full version [8].


© Narek Bojikian, Vera Chekan, and Stefan Kratsch;
licensed under Creative Commons License CC-BY 4.0



This motivates a line of research devoted to the study of the smallest possible functions f for various problem-parameter combinations. In this context, one is often interested in NP-hard problems and hence, conjectures stronger than $\mathsf{P} \neq \mathsf{NP}$ are assumed for conditional lower bounds. For example, it has been shown that unless the Exponential Time Hypothesis (ETH) fails, some problems do not admit algorithms with running time $\mathcal{O}^*(c^k)$ for any constant c (e.g., [38]) – algorithms with such a running time are called *single-exponential*. For problems admitting single-exponential algorithms, it is natural to search for the smallest value of c for which such an algorithm exists. An even stronger conjecture called the Strong Exponential Time Hypothesis (SETH) has been assumed to prove for many problems that existing algorithms with some running time $\mathcal{O}^*(c^k)$ are essentially optimal, i.e., for any $\varepsilon > 0$, there is no algorithm for this problem running in time $\mathcal{O}^*((c-\varepsilon)^k)$. This conjecture states, informally speaking, that the SAT problem cannot be solved much more efficiently than brute-forcing all truth-value assignments.

SETH-tight complexity of problems parameterized by treewidth has been actively studied over the last decade. This was initiated by Lokshtanov et al. [37] who showed that for many classical graph problems (e.g., Independent Set or Max Cut) the folklore dynamic-programming (DP) algorithms are essentially optimal under SETH. In parallel, there is also a line of research devoted to accelerating the existing DP algorithms by employing more careful analysis and advanced tools like fast subset convolution (e.g., [4, 48, 5]), Discrete Fourier Transform (e.g., [47]), rank-based approach (e.g., [6]), isolation lemma ([40]), and Cut&Count (e.g., [17]), we refer to the survey by Nederlof [41] for more details.

Such dynamic-programming algorithms on graphs of bounded treewidth employ the fact that those graphs can be decomposed along small vertex separators and therefore, when processing this decomposition in a bottom-up way, one only needs to remember how a partial solution interacts with the current small separator, also called a bag. Thus it is also natural to study parameters based on small edge-cuts (as edge-counterparts of vertex separators). A linear arrangement of a graph places its vertices on a horizontal line so that no two vertices have the same x-coordinate. Now suppose every edge is drawn as an x-monotone curve, then the cutwidth of this linear arrangement is the maximum number of edges crossing any vertical line – observe that the edges crossing such a line separate the vertices on the left side of the vertical line from the vertices on the right side so they form an edge-cut. The cutwidth of the graph, denoted ctw, is then the minimum over all of its linear arrangements. It is well-known that pathwidth, denoted pw, can be defined similarly but instead of the edges crossing the vertical line, one counts its end-vertices on, say, the right side of the cut. In particular, this implies that the cutwidth of a graph upper-bounds its pathwidth.

Due to this relation, every $\mathcal{O}^*(f(\mathrm{pw}))$-time algorithm is also a $\mathcal{O}^*(f(\mathrm{ctw}))$-time algorithm. However, it is possible that a problem admits a more efficient algorithm when parameterized by cutwidth than when parameterized by pathwidth. For example, Edge Disjoint Paths is paraNP-hard for pathwidth [20] but becomes FPT for cutwidth [24]. This difference sparked deeper interest in studying the cutwidth parameter, and in distinguishing problems whose complexity differ between these two parameterizations. On a finer level, for some of the problems for which the SETH-tight complexity parameterized by treewidth is known, the study continued for the parameterization by cutwidth. The SETH-tight bounds parameterized by cutwidth are known for Independent Set and Dominating Set [45], Odd Cycle Transversal [7], Chromatic Number [30], #q-Coloring [26], as well as a list of connectivity problems (e.g., Feedback Vertex Set and Steiner Tree) [7]. The Chromatic Number problem exposes again a particularly interesting behavior: for treewidth, it is known that for any $q \geq 3$, the folklore $\mathcal{O}^*(q^{\mathrm{tw}})$ algorithm is optimal under SETH [37], but for

cutwidth Jansen and Nederlof [30] proved that there is a (randomized) algorithm computing the chromatic number of the graph in time $\mathcal{O}^*(2^{\mathrm{ctw}})$, i.e., independent of the number of colors in question. Recently, a notable progress was also made for H-Homomorphism: Groenland et al. [25] provided a non-algorithmic proof of the existence of so-called representative sets of certain small size on which a dynamic-programming algorithm can rely, and they also provided a lower-bound construction matching the size of these representative sets. They leave it open, though, whether representative sets of small size can also be found efficiently.

The above-mentioned paper by Lokshtanov et al. [37] provided SETH-tight lower bounds for six classical graph problems, namely Independent Set, Dominating Set, Partition Into Triangles, Odd Cycle Transversal, q-Coloring (for any fixed $q \geq 3$), and Max Cut when parameterized by treewidth [37]. The SETH-tight complexity of these problems parameterized by cutwidth was only partially known till now.

Our results. As a first contribution, we resolve the complexity of the two remaining problems from [37], namely Partition Into Triangles (and also Triangle Packing) as well as Max Cut when parameterized by cutwidth. The study of Partition into Triangles parameterized by cutwidth was initiated by van Heck [46]: it was shown that the problem admits a $\mathcal{O}^*(\sqrt[4]{8}^{\mathrm{ctw}})$ algorithm and no $\mathcal{O}^*((\sqrt{2}-\varepsilon)^{\mathrm{ctw}})$ algorithm exists for any $\varepsilon > 0$ unless SETH fails. In this work, we close this gap by providing an algorithm that solves Triangle Packing in time $\mathcal{O}^*(\sqrt[3]{3}^{\mathrm{ctw}})$, together with a matching SETH-based lower bound for the Partition Into Triangles. There is a trivial reduction from Partition Into Triangles to Triangle Packing and hence, $\mathcal{O}^*(\sqrt[3]{3}^{\mathrm{ctw}})$ is the SETH-optimal running time for both problems. Our algorithm is a straightforward dynamic programming over a path decomposition. While a simple analysis yields running time $\mathcal{O}^*(2^k)$ over a path decomposition of width k, we show that given a linear arrangement ℓ of cutwidth ctw, one can construct a specific path decomposition form ℓ, where a careful analysis shows that the number of states in the dynamic programming is bounded by $\mathcal{O}^*(\sqrt[3]{3}^{\mathrm{ctw}})$, resulting in the desired running time. A bottleneck for the running time are the so-called Z-cuts, i.e., bipartite graphs consisting of a disjoint paths of length three. We show that these cuts inherently determine the complexity of this problem, as the lower bound construction also relies on them. For Max Cut we provide a lower bound showing that no $\mathcal{O}^*((2-\varepsilon)^{\mathrm{ctw}})$ algorithm can solve this problem for any $\varepsilon > 0$ implying that the folklore algorithm for tree decompositions is optimal for cutwidth as well.

Apart from these two problems, we resolve two further open questions. We show that SETH-tight complexity of Hamiltonian Cycle is $\mathcal{O}^*((1+\sqrt{2})^{\mathrm{ctw}})$ when parameterized by cutwidth – this was asked by van Geffen et al. [45] and Bojikian et al. [7]. Let us remark, that although for pathwidth it is known that the $\mathcal{O}^*((2+\sqrt{2})^{\mathrm{pw}})$ algorithm is optimal for Hamiltonian Cycle, the complexity relative to treewidth remains a challenging open problem. Finally, Chaudhary and Zehavi [13] developed a $\mathcal{O}^*(3^{\mathrm{tw}})$ algorithm for Induced Matching problem and an SETH-based lower bound excluding $\mathcal{O}^*((\sqrt{6}-\varepsilon)^{\mathrm{ctw}})$ algorithms for any $\varepsilon > 0$. They conjectured that their algorithm is optimal and asked for a matching lower bound. We confirm their conjecture and provide a stronger result, namely that Induced Matching cannot be solved in $\mathcal{O}^*((3-\varepsilon)^{\mathrm{ctw}})$ for any $\varepsilon > 0$ when parameterized by cutwidth – this resolves the complexity of the problem for both treewidth and cutwidth. [1]

[1] Recently, the conjecture was also independently confirmed by Vasilakis and Lampis [36] by providing a matching lower bound for the parameterization by pathwidth. We remark that our lower bound for cutwidth is a stronger result.

Table 1 Tight bounds for parameterizations by treewidth / pathwidth, and cutwidth. For all problems but Hamiltonian Cycle the tight bounds are known to be equal for treewidth and pathwidth. And the tight complexity of Hamiltonian Cycle parameterized by treewidth remains open. The results of this paper are in the right column.

Triangle Packing (TP)	2^{tw}	$\sqrt[3]{3}^{\mathrm{ctw}}$
Partition into Trianlges (PT)	2^{tw}	$\sqrt[3]{3}^{\mathrm{ctw}}$
Hamiltonian Cycle (HC)	$(2+\sqrt{2})^{\mathrm{pw}}$	$(1+\sqrt{2})^{\mathrm{ctw}}$
Max Cut (MC)	2^{tw}	2^{ctw}
Induced Matching (IM)	3^{tw}	3^{ctw}

Related Work. The (S)ETH tight complexity of problems parameterized by structural parameters has been widely studied. Apart from the already mentioned results, there is a long list of papers related to such algorithms on graphs of bounded treewidth (e.g., [10, 14, 15, 18, 19, 21, 22, 32, 43, 44]), treedepth (e.g., [27, 31, 32, 42]), clique-width (e.g., [1, 2, 9, 23, 28, 31, 33]), rank-width (e.g., [3, 11]), and cutwidth (e.g., [7, 39]). There is also a line of work devoted to conjectures weaker than SETH and yielding the same lower bounds for structural parameterizations (e.g., [12, 35, 34]).

Organization. We start by providing a short summary of the used notation. Section 3 is devoted to Triangle Packing, there we provide our algorithm together with the main steps required to justify its running time. We also sketch the lower-bound construction. After that in Section 4 we present the main idea behind the lower bound state the lower bound for Hamiltonian Cycle. In Section 5 we state our lower bounds for Induced Matching and Max Cut. We conclude in Section 6 by providing some open questions.

2 Preliminaries

We use $\mathcal{O}^*$ notation to suppress factors polynomial in the input size. For an integer $i \in \mathbb{N}_0$ by $[i]$ we denote the set $\{1, \ldots, i\}$ (in particular, we have $[0] = \emptyset$) and by $[i]_0$ we denote the set $[i] \cup \{0\}$. For a vector $a = (a_1, \ldots, a_n)$ over a ground set U and a mapping $f : U \to V$ for some set V, we denote by $\big(f(v)\big)_{v \in a}$ the vector $(f(a_1), \ldots, f(a_n))$.

Given a graph $G = (V, E)$ and a vertex $v \in V$, the neighborhood of v in G is defined as $N_G(v) = \{w \in V(G) \colon \{v, w\} \in E(G)\}$. We omit the index G when clear from the context. For an edge set $F \subseteq E$, we define $G[F] = (V, F)$ and we define $V(F)$ as the set of end-points of F. For a vertex set $S \subseteq V$ and a vertex $v \in V$ we define $N_S(v) = N_G(v) \cap S$ and $\deg_S(v) = |N_S(v)|$. We define $\mathrm{cc}(G)$ as the set of all connected components of G, where a connected component of G is a maximal connected subgraph of G.

We base our lower bounds on the Strong Exponential Time Hypothesis (SETH) [29]:

▶ **Conjecture 1** (SETH). *For every $\varepsilon > 0$, there exists a constant $d > 0$ such that d-SAT cannot be solved in time $\mathcal{O}^*((2-\varepsilon)^n)$, where n is the number of variables.*

Cutwidth. A linear arrangement $\ell = v_1, \ldots, v_n$ of a graph $G = (V, E)$ is a linear ordering of V. We define $V_0 = \emptyset$, $V_i = \{v_1, \ldots, v_i\}$ and $\overline{V}_i = V \setminus V_i$ for $i \in [n]_0$. We define the *cut-graph* at $i \in [n]_0$ as the bipartite graph $H_i = G[V_i, \overline{V}_i]$. The set E_i denotes the edge set of H_i. The cutwidth of ℓ is defined as $\mathrm{ctw}(\ell) = \max_{i \in [n]} |E(H_i)|$. The cutwidth of G is defined as $\mathrm{ctw}(G) = \min_\ell \mathrm{ctw}(\ell)$, where the minimum is taken over all linear arrangements of G. We define L_i and R_i as the set of the left and right endpoints of edges in E_i, respectively. Finally, for $i \in [n]$, we define $Y_i = L_{i-1} \cup \{v_i\}$, i.e., Y_i contains all left end-points of the edges of E_{i-1} together with v_i.

Path decompositions. A *path decomposition* of a graph G is a pair $(P, \mathcal{B} : V(P) \to 2^V)$, where P is a simple path and the following properties hold:

1. For every vertex $v \in V(G)$, the set $\{x \in V(P) \colon v \in \mathcal{B}(x)\}$ induces a non-empty connected subgraph of P.
2. For every edge $\{u, v\} \in E(G)$, there exists a node $x \in V(P)$ with $\{u, v\} \subseteq \mathcal{B}(x)$.

Let $x_1, \ldots, x_r$ be the *nodes* of P in the order they occur on P. The sets $\mathcal{B}(x_1), \ldots, \mathcal{B}(x_r)$ are called *bags*. For every $x_i \in V(P)$, we define $B_{x_i} = \mathcal{B}(x_i)$, $V_{x_i} = \cup_{j \leq i} \mathcal{B}(x_i)$, and $G_{x_i} = G[V_{x_i}]$. A path decomposition $(P, \mathcal{B})$ is *nice*, if we have $\mathcal{B}(x_1) = \mathcal{B}(x_r) = \emptyset$ and for any two consecutive nodes x, x' on P, we have $|\mathcal{B}(x) \Delta \mathcal{B}(x')| \leq 1$. Hence, one can define a nice path decomposition by a sequence of introduce- and forget-vertex operations. It is sometimes useful to have designated *introduce-edge* operations as well, in this case, we call the path decomposition *very nice*. For a node x of a very nice path decomposition, by $G_x = (V_x, E_x)$ we denote the (not necessarily induced) subgraph of G whose vertex resp. edge set consists of the vertices resp. edges of G introduced in this decomposition up to the node x.

3 Triangle Packing

A *triangle packing* of a graph $G = (V, E)$ is a subgraph T of G such that each connected component of T is a cycle of length three, the *size* of T is the number of its connected components. In the Triangle Packing problem, given a graph G and a positive integer b^*, we are asked whether there exists a triangle packing of size b^* in G. In the Partition into Triangles problem, we are asked whether a triangle packing of size $|V|/3$ exists in G. We obtain the following result:

▶ **Theorem 2.** *There exists an algorithm that given an instance (G, b^*) of* Triangle Packing *together with a linear arrangement of G of width at most* ctw*, runs in time $\mathcal{O}^*(\sqrt[3]{3}^{\text{ctw}})$ and outputs whether G admits a triangle packing of size b^*.*

We emphasize that in what follows we only provide a sketch of the proof of this theorem and we refer to the full version for all details.

Let $(G = (V, E), b^*)$ be an instance of Triangle Packing and let $\ell = v_1, \ldots, v_n$ be a linear arrangement of G of cutwidth at most ctw. First, we describe a dynamic-programming algorithm over a nice path decomposition $(P, \mathcal{B})$ of G. After that, we construct a nice path decomposition of G from ℓ and show that it has certain useful properties, namely, the number of possible states of our dynamic-programming algorithm is bounded for each bag.

Algorithm over a path decomposition

Let $(P, \mathcal{B})$ be a nice path decomposition of G. For every node x of P, every set $S \subseteq B_x$, and every integer b, we define the family $\mathcal{H}_x^b[S]$ of all triangle packings of G_x of size b whose intersection with B_x is precisely S and furthermore, each triangle in this packing contains at least one vertex of $V_x \setminus B_x$ (i.e., forgotten already). We define the set $\mathcal{S}_x^b$ consisting of all subsets $S \subseteq B_x$ such that $\mathcal{H}_x^b[S]$ is non-empty. We call $\mathcal{S}_x = \bigcup_{b \in \mathbb{N}} \mathcal{S}_x^b$ the set of *realizable states* at x. For a set $Y \subseteq B_x$, let $\mathcal{S}_x[Y] = \{S \cap Y \mid S \in \mathcal{S}_x\}$ be the set of states "induced" by the set Y.

A straightforward algorithm computing all sets $\mathcal{S}_x^b$ can be informally summarized as follows. Let x' be the node preceding x. At every node x forgetting a vertex, say v, we iterate over all triangles containing the vertex v whose other end-points are still in the bag, iterate over all sets in $\mathcal{S}_{x'}^{b-1}$, and "combine" the two if they are vertex-disjoint. For every $S \in \mathcal{S}_{x'}^b$, we also add $S \setminus \{v\}$ to $\mathcal{S}_x^b$ as the corresponding triangle packing of $G_{x'}$ remains "valid" in G_x.

At every introduce-node x we just keep the family $\mathcal{S}^b_{x'}$. Recall that the root-node, say r, of the nice path decomposition $(P, \mathcal{B})$ is an empty bag. Then the graph G admits a triangle packing of size b if and only if $\emptyset \in \mathcal{S}^b_r$ holds. We can show that if α denotes the maximum size of a family $\mathcal{S}^b_x$ over all nodes x and all integers $b \in [\lfloor \frac{n}{3} \rfloor]_0$, then all families $\mathcal{S}^b_x$ can be computed in time $\mathcal{O}^*(\alpha)$. In order to achieve this, we describe a data structure that allows the insertion of a single "state" in time polynomial in n as well as the iteration over all states present in the data structure in time $\mathcal{O}^*(\alpha)$. In the remainder of the proof, we show how to get a nice path decomposition of G from its linear arrangement ℓ so that the value of α is sufficiently small.

From linear arrangement to path decomposition

To obtain the desired path decomposition $(P, \mathcal{B})$ we will proceed as follows. We will first define a set of the so-called *checkpoint* bags $X_0, \ldots, X_n$ corresponding to the cut graphs $H_0, \ldots, H_n$ of ℓ. Later we will show how to add so-called *transition* bags to turn this sequence into a nice path decomposition, while still keeping the number of possible states bounded.

First, for a bipartite graph H (think of $H = H_0, \ldots, H_n$), we define the sets F_H and I_H by an iterative process formally defined next. The set F_H is intended to represent the set of vertices from the left side of the cut that are forgotten already, while the set I_H corresponds to the vertices on the right side of the cut that are introduced already. To ensure that the corresponding "ordering" of the forget- and introduce-operations even yields a valid path decomposition, i.e., no vertex is forgotten before one of its neighbors is introduced, we will ensure that I_H contains all neighbors of F_H.

Clearly, one can safely forget a vertex if all of its neighbors have already been introduced. We additionally apply the following non-trivial rule. We forget a vertex v if (1) it has a single neighbor w in H that is not introduced yet, and (2) the vertex v has degree at most two in H. In this case we introduce the unique missing neighbor of v before forgetting v. We define Q_H as the vertices on the right side that have not been introduced yet, i.e., do not belong to I_H. We provide formal definitions of these sets (see Figure 1 for an illustration):

▶ **Definition 3.** *Let $H = (L, R, E)$ be a bipartite graph. First, we define $F^{(0)} = \emptyset$. For $j \geq 0$, let $I^{(j)} = N_H(F^{(j)})$, and let $Q^{(j)} = R \setminus I^{(j)}$. We define the sets $F^{(j+1)}$ recursively as follows:*

$$F^{(j+1)} = \left\{ v \in L \colon \deg_{Q^{(j)}}(v) = 0 \vee \left(\deg_{Q^{(j)}}(v) = 1 \wedge \deg(v) \leq 2 \right) \right\}.$$

In the full version we show, that there exists an integer k such that $F^{(k')} = F^{(k)}$ for all $k' \geq k$. Let k be the smallest such value. We define $F_H = F^{(k)}$, $I_H = I^{(k)}$, and $Q_H = Q^{(k)}$. We also define the sets L^1_H and L^2_H as the vertices of $L_H \setminus F_H$ that have exactly one or at least two neighbors in Q_H, respectively. Finally, we define the *bag* at H as $X_H = L^1_H \cup L^2_H \cup I_H$.

For every $i \in [n]_0$, we define $L^2_i = L^2_{H_i}$, $L^1_i = L^1_{H_i}$, $F_i = F_{H_i}$, $I_i = I_{H_i}, Q_i = Q_{H_i}$, and $X_i = X_{H_i}$. We call F_i the set of *forgotten vertices* at i, I_i the set of *introduced vertices*, and Q_i the set of *unintroduced vertices*.

It can be shown that X_i is a vertex separator for every $i \in [n]$: this follows from the fact that I_i is the neighborhood of F_i. Furthermore, we show in the full version that if in Definition 3 instead of $F^{(0)} = \emptyset$, we start with the set $F^{(0)} = F_{i-1}$ of the vertices already forgotten at the previous cut, then we obtain the same set F_i of vertices forgotten at the current cut. From this we can then conclude that a nice path decomposition of G can be constructed by using $X_1, \ldots, X_n$ as the main building blocks. We emphasize that the sequence of vertex separators $X_1, \ldots, X_n$ itself does not even necessarily contain every vertex of G. To resolve this, we will turn it into a nice path decomposition by adding the so-called *transition bags*. The bags $X_1, \ldots, X_n$ are called the *checkpoint bags*, and $x_1, \ldots, x_n$ denote the corresponding nodes in the arising path decomposition.

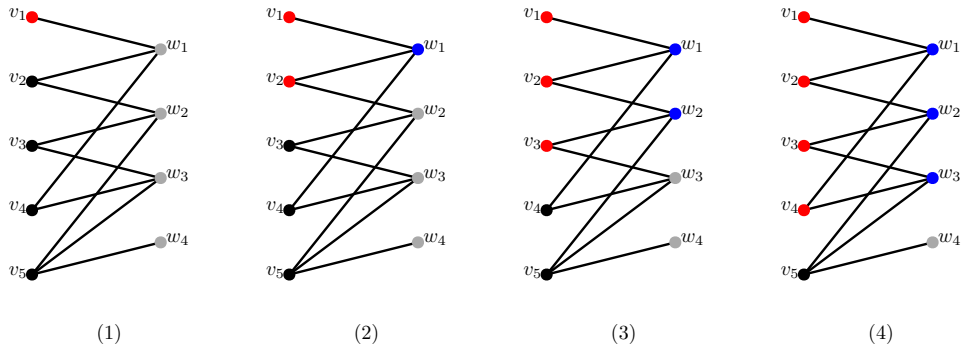


Figure 1 For $i = 1, 2, 3, 4$, the family $F^{(i)}$ is red on the left-hand side of the cut while the families $I^{(i-1)}$ and $Q^{(i-1)}$ are blue and gray, respectively, on the right-hand side. The sets F_H, I_H, Q_H are the red, blue, and gray vertices, respectively, in (4). The black vertex in (4) belongs to L_H^1 as it has a single gray neighbor. The bag X_H is the set of all black and blue vertices in (4).

The reason why we distinguish the sets L_i^1 and L_i^2 is two-fold. First, to bound the number of states in terms of cutwidth, we will "assign" the edges of the cut H_i to certain sets of vertices in the bag, these sets are then called *components*. Since every vertex in L_i^2 has at least two neighbors in Q_i, and since no vertex of Q_i belongs to the bag X_i by definition, we can assign at least two cut-edges to every vertex of L_i^2. Based on this, we show that for some components of a specific structure, enough edges of the cut can be assigned to these components to "allow" all possible state combinations of the vertices of these components. Let us elaborate a bit more on what we mean by this. Our goal is to bound the number of states of the bag X_i in terms of the number of edges in the cut H_i. To prove the desired bound, it suffices to partition the bag into the components, partition the edges of the cut to be assigned to these components, and then show that the bound holds for each component with respect to the number of assigned edges. We will show that if a component contains a vertex from L_i^2, then for the number s of vertices of this component and the number q of edges assigned to it, we have $2^s \leq \sqrt[3]{3}^q$, i.e., the desired bound holds even without a further careful analysis. We will provide more details later. Second, we also need to ensure that along the way between the checkpoint bags, i.e., in transition bags, we do not have too many possible states. This will be achieved by a careful choice of the ordering in which the vertices are forgotten and introduced. The sets L_i^1 and L_i^2 will be used to determine this ordering.

Bounding the number of realizable states

We define the graph $\hat{H}_i$ as $\hat{H}_i = H_i[V_i, I_i]$ to represent the part of the cut restricted to the vertices introduced so far (i.e., we discard the not yet introduced vertices of Q_i from H_i). For a set $S_0 \subseteq R_i$ of vertices on the right side of the cut-graph H_i, we also define the graph $H_i^{S_0} = H_i[V_i, I_i \setminus S_0]$ that additionally removes the vertices S_0 from the right side of the cut. This graph will be crucial to bound the number of the possible states for the transition bags: for example, the graph $H_{i-1}^{\{v_i\}}$ will be useful to analyze the first step of the transition from the cut H_{i-1} to the cut H_i, i.e., removing v_i from the right side of the cut.

▶ **Definition 4.** *For every $i \in [n]$ and $S \subseteq V(H_i)$, we define the set E_i^S of all edges of H_i incident with S and we define $m_i(S) = |E_i^S|$.*

By definition of the bipartite graph $\hat{H}_i$, each edge of H_i either has both end-points in some connected component of $\hat{H}_i$, or it has its left end-point in some connected component of $\hat{H}_i$ and its right end-point in Q_i. First, this implies that we have $E_i^{V(C)} = E(C) \dot{\cup} E_{H_i}(C, Q_i)$ for

every connected component C of $\hat{H}_i$. And second, it shows that every edge of H_i is incident with the vertices of exactly one connected component of $\hat{H}_i$. Therefore, the sets $E_i^{V(C)}$ for $C \in \mathrm{cc}(\hat{H}_i)$ partition $E(H_i)$. This will be crucial to bound the number of states in the checkpoint bags. An analogous argument shows that for any choice of $S_0 \subseteq R_i$, the sets $E_i^{V(C)}$ for $C \in \mathrm{cc}(H_i^{S_0})$ partition the set $E(H_i)$ as well.

Now we aim at showing that for each connected component $C \in \mathrm{cc}(\hat{H}_i)$, the number of possible states of $\mathcal{S}_{x_i}[V(C)]$ is upper-bounded by $\sqrt[3]{3}^{m_i(V(C))}$. The bound on $\mathcal{S}_{x_i}$ then follows by the fact that the sets $E_i^{V(C)}$ are pairwise disjoint. This will actually be the part where it becomes evident, as we shall see in the following proof, that the so-called Z-cuts form the bottleneck of the algorithm: We will distinguish different types of components C of $\hat{H}_i$ and the tight upper bound on the number of possible states is achieved by the Z-cuts.

We actually prove a more general statement. First, we prove that the claimed bound holds not only for $C \in \mathrm{cc}(\hat{H}_i)$, but for any connected component C of $H_i^{S_0}$ where $S_0 \subseteq R_i$ is arbitrary. Here it is important to remember that S_0 only contains vertices from the right side of the cut. Second, we will show that the bound holds even if we allow to add any subset of $L_i \setminus F_i$ to every realizable state. This motivates the next definition of the set $\mathcal{T}_i^b[S]$ which can be considered as a robust generalization of the set of possible states: intuitively, this permits us to say that even if we allow, instead of $G[V_i]$, an arbitrary graph on the left-hand side of the cut (i.e., a triangle packing is allowed to use an arbitrary subset of vertices on the left-hand side), the number of states is still bounded. This will later allow us to prove the bound for the transition bags as well, given that the transitions are carried out in the correct order. For every $i \in [n]$ and every $b \in [\lfloor \frac{n}{3} \rfloor]_0$, we will use $\mathcal{S}_i^b$ as a shorthand for $\mathcal{S}_{x_i}^b$.

▶ **Definition 5.** *For $b \in [\lfloor \frac{n}{3} \rfloor]_0$, $i \in [n]$, and $S \subseteq V(H_i)$, we define*

$$\mathcal{T}_i^b[S] = \{S' \cup T \colon S' \in \mathcal{S}_i^b[S], T \subseteq S \cap (L_i^1 \cup L_i^2)\} \text{ and } \mathcal{T}_i[S] = \cup_{b \in [\lfloor \frac{n}{3} \rfloor]_0} \mathcal{T}_i^b[S].$$

For a connected component C of a subgraph of H_i, we use $\mathcal{T}_i[C]$ and $m_i(C)$ as shorthands for $\mathcal{T}_i[V(C)]$ and $m_i(V(C))$, respectively. Now we prove the main technical lemma.

▶ **Lemma 6.** *For all $i \in [n]$, all $S_0 \subseteq R_i$, and all $C \in \mathrm{cc}(H_i^{S_0})$ it holds that $|\mathcal{T}_i[C]| \leq \sqrt[3]{3}^{m_i(C)}$.*

Sketch. Let $F = F_i \cap V(C)$, $I = I_i \cap V(C)$, $L^1 = L_i^1 \cap V(C)$, $L^2 = L_i^2 \cap V(C)$, and $X = X_i \cap V(C)$. By definition of these sets we thus get $|X| = |L^1| + |L^2| + |I|$. First of all, it can be argued that the claim is true whenever $m_i(C) \leq 2$ so we assume $m_i(C) \geq 3$ in the remainder. The proof is based on two main inequalities. The first follows directly from the definition of the $\mathcal{T}_i^b$: as every element of $\mathcal{T}_i$ is a subset of $V(C) \cap X_i = X$, we have:

$$|\mathcal{T}_i[C]| \leq 2^{|X|}. \tag{1}$$

For the second inequality, recall that $E_i^C = E(C) \dot{\cup} E_{H_i}(C, Q_i)$ holds, i.e., we have $m_i(C) = |E(C)| + |E_{H_i}(C, Q_i)|$. Since C is a connected component of $H_i^{S_0}$, its edge set $E(C)$ contains at least $|V(C)| - 1 = |L^2| + |L^1| + |F| + |I| - 1$ edges. Moreover, we claim that $|F| \geq |I|$ holds. Recall that by definition of the sets $F^{(j)}$ and $I^{(j)}$ for $j \geq 0$ (see Definition 3), we have that $|F^{(j)}| \geq |I^{(j)}|$: this is because a vertex can only be added to I_i (i.e., introduced) if at least one of its neighbors is added to F_i (i.e., forgotten). Furthermore, for any vertex v in $I^{(j)} \cap V(C)$, the unique vertex in $F^{(j)}$ due to which v was introduced belongs to the same connected component C. Thus we have $|E(C)| \geq |L^2| + |L^1| + 2|I| - 1$. Finally, recall that by definition, each vertex of L^1 has exactly one neighbor in Q_i while each vertex of L^2 has at least two neighbors in Q_i. Hence, it holds that $|E_{H_i}(C, Q_i)| \geq |L^1| + 2|L^2|$. It follows that $m_i(C) \geq (|L^2| + |L^1| + 2|I| - 1) + (|L^1| + 2|L^2|) = 2|X| + |L^2| - 1$, and hence we get:

$$|X| \leq \frac{m_i(C) + 1}{2}. \tag{2}$$

First, assume that at least one of the following hold: C contains a cycle, or L^2 is not empty, or $|F| > |I|$, or $V(C) \cap L_i$ has a neighbor in $S_0 \cap I_i$. It is not hard to verify that in this case $m_i(C) \geq 2|X|$ holds and therefore, $|\mathcal{T}_i[C]| \leq 2^{|X|} \leq \sqrt{2}^{m_i(C)} \leq \sqrt[3]{3}^{m_i(C)}$.

In the remainder of the proof we may thus assume that C is a tree, L^2 is empty, $|F| = |I|$, and each neighbor of $V(C) \cap L_i$ in S_0 belongs to Q_i. We can show that in this case, each vertex of F has degree at most 2 in H_i. After that we carry out an extensive case distinction and show that in each case, there exists a constant fraction of all subsets of X that are certainly not elements of $\mathcal{T}[C]$. The main idea behind each of the cases is to find a vertex, say v, in I with certain nice properties, namely there exists a constant-sized subset, say U, of X such that any triangle packing using v has to use at least one of the vertices in X. Thus, every subset of X that contains v and has an empty intersection with U is not a possible state – let us remark that this intuition reflects the proof in an overly simplified manner for space reasons.

The simplest of these cases is when C contains a leaf v in I. Then let w be the unique neighbor of v in L_i. As v was introduced due to forgetting one of its neighbors, this neighbor has to be w, i.e., we have $w \in F$. As $m_i(C) \geq 3$, there exists a vertex $v' \neq v \in I$ adjacent to w. Since every vertex in F has degree at most 2 in H_i, the vertex w does not have neighbors other than v and v'. Recall that the dynamic-programming algorithm only considers triangles where at least one vertex is forgotten already. Since w is the unique forgotten neighbor of v, every such triangle packing containing the vertex v, uses the triangle v, w, v'. Recall that we have $v, v' \in I \subseteq X$. Hence, for every $S \in \mathcal{T}_i[C]$, the property $v \in S$ implies $v' \in S$. This way we exclude one fourth of all subsets of X from $\mathcal{T}_i[C]$ and get:

$$|\mathcal{T}_i[C]| \leq \frac{3}{4}2^{|X|} \leq \frac{3}{4}2^{\frac{m_i(C)+1}{2}} \leq \sqrt[3]{3}^{m_i(C)},$$

where the last inequality holds due to $m_i(C) \geq 3$.

This is the spot where a Z-cut occurs: If C contains exactly three edges, then it consists of the vertices v, v', w as well as a vertex, say $w' \in F$, adjacent to v' only, and C induces a Z-cut. Recall that by definition we have $X = \{v, v'\}$ (i.e., the bag contains v and v') since the vertices w and w' are forgotten, i.e., belong to F. One can verify that in this case all states other than $\{v\}$ are possible and therefore, the above inequality is tight and the equality is achieved by a Z-cut.

If C contains no leaves in I, we can show that $m_i(C) \geq 5$ holds. Analyzing the structure of the cut, we then show the inequality $|\mathcal{T}_i[C]| \leq \frac{49}{64}2^{|X|}$ implying the desired bound. ◀

By applying the above lemma with $S_0 = \emptyset$, we can show that the number of states at each checkpoint bag is upper-bounded by $\sqrt[3]{3}^{\mathrm{ctw}}$:

▶ **Corollary 7.** *For all $i \in [n]$ and $b \in [\lfloor \frac{n}{3} \rfloor]_0$, it holds that $|\mathcal{S}_i^b| \leq \sqrt[3]{3}^{|E_i|}$.*

Due to space constraints, we omit the proof of the following bound for transition bags. It relies on a careful choice of the ordering of the forget- and introduce-operations together with an involved analysis of the relation between the connected components of the graphs $\hat{H}_{i-1}$ and $\hat{H}_i$ as well as the states of these components:

▶ **Lemma 8.** *For every transition node x_i^j and every $b \in [\lfloor \frac{n}{3} \rfloor]_0$, it holds that $|\mathcal{S}_{x_i^j}^b| \leq 4 \cdot \sqrt[3]{3}^{\mathrm{ctw}}$.*

In Figure 2 we sketch the idea behind our lower-bound construction matching the running time of our algorithm from the previous subsection and refer to the full version for a formal description. We prove the following result for Partition into Triangles:

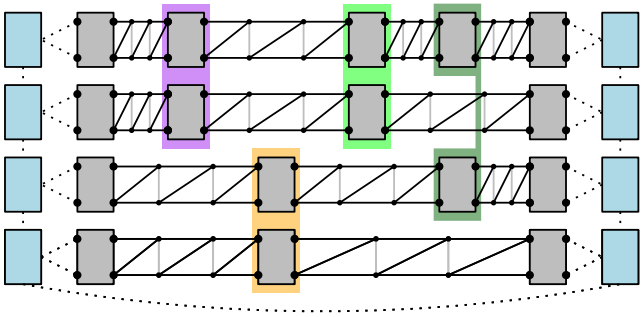


Figure 2 Sketch of the lower-bound construction for $n = 4$ variables and $m = 4$ clauses in an instance of d-CSP-3. Each of the gray boxes is a path gadget which may have one of the three states. Blue boxes are small cliques. Dotted lines reflect that there exist all possible edges between the sets. Every colored box is a constraint gadget. The cutwidth of the construction is essentially upper-bounded by $3n$ as each of the n Z-cuts contributes three edges to the size of a cut.

▶ **Theorem 9.** *Assuming SETH, there exists no algorithm that solves* Partition into Triangles *on graphs given with linear arrangements of cutwidth k in time $\mathcal{O}^*\big((\sqrt[3]{3}-\varepsilon)^k\big)$ for any $\varepsilon > 0$.*

Since there is a trivial reduction from Partition into Triangles to Triangle Packing, this lower bound then also holds for Triangle Packing and so the running time of $\mathcal{O}^*(\sqrt[3]{3}^{\mathrm{ctw}})$ is optimal for both problems under SETH.

4 Hamiltonian Cycle

For a graph $G = (V, E)$, a set $C \subseteq E$ of edges is called a *Hamiltonian cycle* of G if C induces a single cycle visiting all vertices of G. In the Hamiltonian Cycle problem, we are given a graph G and asked if there is a Hamiltonian cycle in G.

Here we provide a randomized algorithm solving Hamiltonian Cycle in time $\mathcal{O}^*((1+\sqrt{2})^k)$ on graphs provided with a linear arrangement of cutwidth k. For this we adapt the $\mathcal{O}^*((2+\sqrt{2})^p)$ algorithm by Cygan et al. [16] working on graphs provided with a path decomposition of pathwidth p. We will transform a linear arrangement of cutwidth k into a path decomposition having a useful algorithmic property, namely, the dynamic-programming table as defined by Cygan et al. has only $\mathcal{O}^*((1+\sqrt{2})^k)$ non-zero entries and there is an efficient way to determine the "certainly zero" entries. Now we provide some details. The algorithm by Cygan et al. [16] strongly relies on their algebraic result about the $\mathbb{F}_2$-rank of a certain "compatibility" matrix reflecting, for each pair of perfect matchings, whether their union is a Hamiltonian cycle. We summarize the necessary parts of this result:

▶ **Theorem 10** ([16])**.** *Let $t \in \mathbb{N}$ be even. Then there exists a set $\mathcal{X}_t$ of perfect matchings of the complete graph K_t with the following properties: (1) $|\mathcal{X}_t| = 2^{t/2-1}$, (2) $\mathcal{X}_t$ can be computed in time $\sqrt{2}^t t^{\mathcal{O}(1)}$, and (3) for every matching $M \in \mathcal{X}_t$, there exists a unique matching $M' \in \mathcal{X}_t$ such that $M \cup M'$ forms a Hamiltonian cycle of K_t.*

Observe that the third property implies that one can partition the set $\mathcal{X}_t$ into pairs of matchings such that the union of two perfect matchings forms a Hamiltonian cycle, if and only if the two matchings are paired in this partition. In other words, the family $\mathcal{X}_t$ induces a permutation submatrix of the compatibility matrix mentioned above. Like Cygan et al., we will sometimes identify some ordered set, say S, of even cardinality t with the vertex set $[t]$ of K_t. Then by $\mathcal{X}(S)$ we denote the set obtained from $\mathcal{X}_t$ by identifying the elements of S with $[t]$. In particular, every element of $\mathcal{X}(S)$ is a perfect matching on the vertex set S.

In the following, we assume that the graph G is provided with a weight function $\omega\colon E(G) \to \mathbb{N}$. As many other algorithms for connectivity problems, the algorithm of Cygan et al. [16] makes use of the classic isolation lemma (see [40]) and samples ω in a certain probabilistic way. Their algorithm works on a very nice path decomposition of the input graph G. Essentially, it processes such a path decomposition and for every bag, counts, modulo 2, the number of subgraphs of the already processed graph such that every vertex in this subgraph has degree 2, except for vertices in the current bag which may have lower degree. The dynamic-programming table refines these counts depending on the weight of the subgraph, its degree sequence on the bag, and whether this subgraph forms a single cycle with a certain matching defined on the vertex set of the bag. A crucial implication of Theorem 10 (as proven in their paper) is that instead of taking all such matchings into account, it suffices to consider only the "base matchings" from $\mathcal{X}_t$ to solve the problem. Now we summarize this more formally:

▶ **Definition 11** ([16]). *A* partial cycle cover *of a graph is a set of edges such that every vertex has at most two incident edges in this set. For every node x of the provided very nice path decomposition, every $s\colon B_x \to \{0,1,2\}$ where $s^{-1}(1)$ has even cardinality, every $M \in \mathcal{X}(s^{-1}(1))$, and every $w \in \mathbb{N}$, the value $T_x[s,M,w]$ is defined as the number, modulo 2, of partial cycle covers C of the graph G_x with the following properties:*

1. *the set $C \cup M$ of edges induces a single cycle,*
2. *the total weight (with respect to ω) of the edges in C is w,*
3. *every vertex $v \in B_x$ has precisely $s(v)$ incident edges in C,*
4. *every vertex $v \in V(G_x) \setminus B_x$ has precisely two incident edges in C.*

We say that a partial cycle cover C of G_x has *footprint* s on x if it satisfies the last two items.

▶ **Theorem 12** ([16]). *Let x be a non-first node of a very nice path decomposition of a graph G and let y denote its predecessor. For any fixed $s\colon B_x \to \{0,1,2\}$ where $s^{-1}(1)$ has even cardinality, every $M \in \mathcal{X}(s^{-1}(1))$, and every $w \in \mathbb{N}$, given the table T_y, the value $T_x[s,M,w]$ can be computed in time $\mathcal{O}^*(1)$ by querying $\mathcal{O}(1)$ entries of T_y.*

Let G be a graph and let $\ell = v_1, \ldots, v_n$ be a linear arrangement of G of cutwidth at most k. The aim now is to compute from ℓ a path decomposition of G with certain nice properties to which we will later apply the algorithm from Theorem 12. We recall that by definition for every $i \in [n]$, the set Y_i consists of all left end-points of the edges in the $(i-1)$st cut E_{i-1} of ℓ together with the vertex v_i. Thus, we have $|E_{i-1}| \leq k$ and $|Y_i| \leq k+1$ for all $i \in [n]$. It is well-known that the sequence $Y_1, \ldots, Y_n$ of bags is a path decomposition of G (see e.g., [26] for the idea and [7] for a formal proof). To reverse the ordering in which these bags are traversed, for every $i \in [n]$, we define the set $X_i = Y_{n+1-i}$.

▶ **Definition 13.** *For a node x of a path decomposition we define the sets*

$$B_1(x) = \{u \in B_x \mid \deg_{G_x}(u) = 1\} \text{ and } B_2(x) = \{u \in B_x \mid \deg_{G_x}(u) \geq 2\}.$$

We call a mapping $s\colon B_x \to \{0,1,2\}$ relevant *if all of the following hold: (1) $|s^{-1}(1)|$ is even, (2) $s^{-1}(2) \subseteq B_2(x)$, and (3) $s^{-1}(1) \subseteq B_1(x) \cup B_2(x)$.*

▶ **Lemma 14.** *Let x be a node of a very nice path decomposition of G and P be a partial cycle cover of G_x. Let $s\colon B_x \to \{0,1,2\}$ be such that P has footprint s. Then s is relevant.*

Furthermore, if $|B_1(x)| + 2 \cdot |B_2(x)| \leq k + \mathcal{O}(1)$ holds, then the number of pairs (s,M) such that $s\colon B_x \to \{0,1,2\}$ is relevant and $M \in \mathcal{X}(s^{-1}(1))$ is upper-bounded by $\mathcal{O}((1+\sqrt{2})^k)$.

Sketch. Let P be a partial cycle cover of G_x and let $s\colon B_x \to \{0,1,2\}$ be such that P has footprint s. By definition of a footprint, all end-vertices of P belong to B_x. Furthermore, every vertex of degree 1 resp. 2 in P has the degree of at least 1 resp. 2 in G_x so the first claim holds. Now let $\ell_1 = |B_1(x)|$ and $\ell_2 = |B_2(x)|$. The number of pairs (s, M) such that $s\colon B_x \to \{0,1,2\}$ is relevant and $M \in \mathcal{X}(s^{-1}(1))$ is at most

$$\sum_{0 \le i_2 \le \ell_2} \binom{\ell_2}{i_2} \sum_{0 \le i_1 \le (\ell_2 - i_2) + \ell_1} \binom{(\ell_2 - i_2) + \ell_1}{i_1} 2^{i_1/2-1}.$$

A careful analysis upper-bounds this by $\mathcal{O}((1+\sqrt{2})^k)$ if $\ell_1 + 2 \cdot \ell_2 \le k + \mathcal{O}(1)$ holds. ◀

▶ **Lemma 15.** *From the linear arrangement $v_1, \ldots, v_n$ of the graph G, in polynomial time we can construct a very nice path decomposition of G in which each node x satisfies $|B_1(x)| + 2 \cdot |B_2(x)| \le k + \mathcal{O}(1)$.*

Sketch. The desired path decomposition is obtained by starting with the nodes $x_1, \ldots, x_n$ corresponding to the bags $X_1, \ldots X_n$, respectively, and making the decomposition very nice by a careful choice of the ordering of introduce-vertex-, introduce-edge-, and forget-vertex-operations. This ordering, in particular, ensures that the graph G_{x_i} contains no edges with both end-points in $X_i \setminus \{v_{n+1-i}\}$. We recall that for every $i \in [n]$, all vertices in the bag X_i (apart from v_{n+1-i}) are left end-points of edges in the cut E_{n-i} (whose size is bounded by k). So every vertex in $B_1(x_i) \setminus \{v_{n+1-i}\}$ resp. $B_2(x_i) \setminus \{v_{n+1-i}\}$ contributes 1 resp. at least 2 to the size of E_{n-i}. And therefore $|B_1(x_i)| + 2 \cdot |B_2(x_i)|$ is upper-bounded by $k+2$. We also show that the intermediate bags added to make the decomposition very nice also have this property as they can be upper-bounded using x_i for some $i \in [n]$. ◀

We can then run the dynamic-programming algorithm from Theorem 12 restricted to relevant footprints only to compute the values $T_r[\emptyset, \emptyset, w]$ for every "reasonable" integer w where r denotes the root of a path decomposition satisfying the above lemma. Cygan et al. [16] show that this information suffices to find out, with high probability, if the graph G admits a Hamiltonian cycle. We refer to the full version for all details.

▶ **Theorem 16.** *There exists a one-sided error Monte-Carlo algorithm that takes a graph G together with a linear arrangement of G of cutwidth at most* ctw*, runs in time $\mathcal{O}^*((1+\sqrt{2})^{\mathrm{ctw}})$, and solves the* Hamiltonian Cycle *problem. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Cygan et al. [16] showed that unless SETH fails, no algorithm working on path decompositions of pathwidth pw can solve the problem in time $\mathcal{O}^*((2+\sqrt{2}-\varepsilon)^{\mathrm{pw}})$ for any $\varepsilon > 0$. We show that their ideas are also useful to exclude the $\mathcal{O}^*((1+\sqrt{2}-\varepsilon)^{\mathrm{ctw}})$ algorithms for any $\varepsilon > 0$. To ensure that the construction has bounded cutwidth (and not only pathwidth), we modify the connections between path gadgets and employ new clause gadgets:

▶ **Theorem 17.** *Assuming SETH, there is no algorithm that solves* Hamiltonian Cycle *on graphs given with linear arrangements of cutwidth k in time $\mathcal{O}^*((1+\sqrt{2}-\varepsilon)^k)$ for any $\varepsilon > 0$.*

5 Max Cut and Induced Matching

In Induced Matching, given a graph G and an integer b, we are asked whether G contains a matching M of cardinality b such that $(V(M), M)$ is an induced subgraph of G, i.e., there are no edges other than M with both endpoints in $V(M)$. We prove the following result:

▶ **Theorem 18.** *Assuming SETH, there is no algorithm that solves* Induced Matching *on graphs given with linear arrangements of cutwidth k in time $\mathcal{O}^*\big((3-\varepsilon)^k\big)$ for any $\varepsilon > 0$.*

We recall that the treewidth of a graph is upper-bounded by its cutwidth. This shows that the $\mathcal{O}^*\big(3^k\big)$ algorithm by Chaudhary and Zehavi [13] working on graphs provided with tree decompositions of treewidth at most k is optimal for both treewidth and cutwidth and thus answers their open question. In Max Cut we are given a graph and asked to partition its vertex set into two sets to maximize the number of edges between these sets. We prove that the folklore $\mathcal{O}^*\big(2^k\big)$ algorithm working on graphs provided with tree decompositions of treewidth at most k is optimal for both treewidth and cutwidth:

▶ **Theorem 19.** *Assuming SETH, there is no algorithm that solves* Max Cut *on graphs given with linear arrangements of cutwidth k in time $\mathcal{O}^*\big((2-\varepsilon)^k\big)$ for any $\varepsilon > 0$.*

6 Conclusion and Future Work

Our results together with previous work [7, 30] show an interesting variety of behaviors for the complexity of problems relative to treewidth/pathwidth vs. relative to cutwidth: We may get the same tight bound, a small decrease in complexity, or a substantial decrease. We also discovered two more rare examples of tight exponential bounds with non-integral bases, especially Triangle Packing, which has an integral base relative to treewidth. In our opinion, this makes parameterization by cutwidth a very good test bed for deepening the understanding of dynamic programming and width parameters.

There are several avenues for future work: (i) Some edge-disjoint packing problems are intractable for treewidth but we may be able to dertermine tight bounds for cutwidth. (ii) Going beyond classical problems, it would be interesting to determine the tight complexity of (σ, ρ)-domination problems for cutwidth which is known for treewidth [21]. (iii) Closing the gap for H-Homomorphism left by Groenland et al. [25] is a challenging open question.

References

1 Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parameterized by clique-width. *Theor. Comput. Sci.*, 782:30–53, 2019. doi:10.1016/j.tcs.2019.02.030.

2 Benjamin Bergougnoux and Mamadou Moustapha Kanté. More Applications of the d-Neighbor Equivalence: Acyclicity and Connectivity Constraints. *SIAM J. Discret. Math.*, 35(3):1881–1926, 2021. doi:10.1137/20M1350571.

3 Benjamin Bergougnoux, Tuukka Korhonen, and Jesper Nederlof. Tight Lower Bounds for Problems Parameterized by Rank-Width. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.11.

4 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.

5 Andreas Björklund, Thore Husfeldt, Petteri Kaski, Mikko Koivisto, Jesper Nederlof, and Pekka Parviainen. Fast Zeta Transforms for Lattices with Few Irreducibles. *ACM Trans. Algorithms*, 12(1):4:1–4:19, 2016. doi:10.1145/2629429.

6 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

7 Narek Bojikian, Vera Chekan, Falko Hegerfeld, and Stefan Kratsch. Tight Bounds for Connectivity Problems Parameterized by Cutwidth. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.14.

8 Narek Bojikian, Vera Chekan, and Stefan Kratsch. Tight Bounds for some Classical Problems Parameterized by Cutwidth. *CoRR*, abs/2502.15884, 2025. doi:10.48550/arXiv.2502.15884.

9 Narek Bojikian and Stefan Kratsch. A tight Monte-Carlo algorithm for Steiner Tree parameterized by clique-width. *CoRR*, abs/2307.14264, 2023. doi:10.48550/arXiv.2307.14264.

10 Glencora Borradaile and Hung Le. Optimal Dynamic Program for r-Domination Problems over Tree Decompositions. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.8.

11 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discret. Appl. Math.*, 158(7):809–819, 2010. doi:10.1016/j.dam.2009.09.009.

12 Barış Can Esmer, Jacob Focke, Dániel Marx, and Paweł Rzążewski. Fundamental Problems on Bounded-Treewidth Graphs: The Real Source of Hardness. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:17, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2024.34.

13 Juhi Chaudhary and Meirav Zehavi. P-Matchings Parameterized by Treewidth. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science – 49th International Workshop, WG 2023, Fribourg, Switzerland, June 28-30, 2023, Revised Selected Papers*, volume 14093 of *Lecture Notes in Computer Science*, pages 217–231. Springer, 2023. doi:10.1007/978-3-031-43380-1_16.

14 Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A Tight Lower Bound for Counting Hamiltonian Cycles via Matrix Rank. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1080–1099. SIAM, 2018. doi:10.1137/1.9781611975031.70.

15 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.

16 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity Checking Via Bases of Perfect Matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

17 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *CoRR*, abs/1103.0534, 2011. arXiv:1103.0534.

18 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

19 Baris Can Esmer, Jacob Focke, Dániel Marx, and Pawel Rzazewski. List homomorphisms by deleting edges and vertices: tight complexity bounds for bounded-treewidth graphs. *CoRR*, abs/2210.10677, 2022. doi:10.48550/arXiv.2210.10677.

20 Krzysztof Fleszar, Matthias Mnich, and Joachim Spoerhase. New algorithms for maximum disjoint paths based on tree-likeness. *Math. Program.*, 171(1-2):433–461, 2018. doi:10.1007/S10107-017-1199-3.

21 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight Complexity Bounds for Counting Generalized Dominating Sets in Bounded-Treewidth Graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3664–3683. SIAM, 2023. doi:10.1137/1.9781611977554.ch140.

22 Jacob Focke, Dániel Marx, and Pawel Rzazewski. Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 – 12, 2022*, pages 431–458. SIAM, 2022. doi:10.1137/1.9781611977073.22.

23 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The Fine-Grained Complexity of Graph Homomorphism Parameterized by Clique-Width. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 66:1–66:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.66.

24 Robert Ganian and Sebastian Ordyniak. The Power of Cut-Based Parameters for Computing Edge-Disjoint Paths. *Algorithmica*, 83(2):726–752, 2021. doi:10.1007/S00453-020-00772-W.

25 Carla Groenland, Isja Mannens, Jesper Nederlof, Marta Piecyk, and Pawel Rzazewski. Towards Tight Bounds for the Graph Homomorphism Problem Parameterized by Cutwidth via Asymptotic Matrix Parameters. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 77:1–77:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.77.

26 Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. Tight Bounds for Counting Colorings and Connected Edge Sets Parameterized by Cutwidth. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.36.

27 Falko Hegerfeld and Stefan Kratsch. Solving Connectivity Problems Parameterized by Treedepth in Single-Exponential Time and Polynomial Space. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.STACS.2020.29.

28 Falko Hegerfeld and Stefan Kratsch. Tight Algorithms for Connectivity Problems Parameterized by Clique-Width. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 59:1–59:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.59.

29 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

30 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.*, 795:520–539, 2019. doi:10.1016/j.tcs.2019.08.006.

31 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

32 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. *Discret. Appl. Math.*, 308:168–186, 2022. doi:10.1016/j.dam.2020.03.052.

33 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

34 Michael Lampis. Circuits and Backdoors: Five Shades of the SETH, 2024. doi:10.48550/arXiv.2407.09683.

35 Michael Lampis. The Primal Pathwidth SETH. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 1494–1564. SIAM, 2025. doi:10.1137/1.9781611978322.47.

36 Michael Lampis and Manolis Vasilakis. Structural Parameterizations for Induced and Acyclic Matching. *CoRR*, abs/2502.14161, 2025. doi:10.48550/arXiv.2502.14161.

37 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known Algorithms on Graphs of Bounded Treewidth Are Probably Optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

38 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly Superexponential Parameterized Problems. *SIAM Journal on Computing*, 47(3):675–702, 2018. doi:10.1137/16M1104834.

39 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and Gaps: Tight Complexity Results of General Factor Problems Parameterized by Treewidth and Cutwidth. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 95:1–95:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.95.

40 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Comb.*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

41 Jesper Nederlof. Algorithms for NP-Hard Problems via Rank-Related Parameters of Matrices. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms – Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 145–164. Springer, 2020. doi:10.1007/978-3-030-42071-0_11.

42 Jesper Nederlof, Michał Pilipczuk, Céline M. F. Swennenhuis, and Karol Węgrzycki. Hamiltonian Cycle Parameterized by Treedepth in Single Exponential Time and Polynomial Space. In *Proc. WG 2020*, volume 12301 of *Lecture Notes Comput. Sci.*, pages 27–39, 2020. doi:10.1007/978-3-030-60440-0_3.

43 Karolina Okrasa, Marta Piecyk, and Pawel Rzazewski. Full Complexity Classification of the List Homomorphism Problem for Bounded-Treewidth Graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.74.

44 Karolina Okrasa and Pawel Rzazewski. Fine-Grained Complexity of the Graph Homomorphism Problem for Bounded-Treewidth Graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. doi:10.1137/20M1320146.

45 Bas A. M. van Geffen, Bart M. P. Jansen, Arnoud A. W. M. de Kroon, and Rolf Morel. Lower Bounds for Dynamic Programming on Planar Graphs of Bounded Cutwidth. *J. Graph Algorithms Appl.*, 24(3):461–482, 2020. doi:10.7155/jgaa.00542.

46 Ivo van Heck. Triangle partition on graphs of bounded cutwidth upper- and lower bounds on algorithmic and communication complexity. Master's thesis, Eindhoven University of Technology, Eindhoven, June 2018. Available at `https://pure.tue.nl/ws/portalfiles/portal/109480417/Thesis0775551_Ivo_van_Heck.pdf`.

47 Johan M. M. van Rooij. Fast Algorithms for Join Operations on Tree Decompositions. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms – Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 262–297. Springer, 2020. `doi:10.1007/978-3-030-42071-0_18`.

48 Johan M. M. van Rooij. A Generic Convolution Algorithm for Join Operations on Tree Decompositions. In Rahul Santhanam and Daniil Musatov, editors, *Computer Science – Theory and Applications – 16th International Computer Science Symposium in Russia, CSR 2021, Sochi, Russia, June 28 – July 2, 2021, Proceedings*, volume 12730 of *Lecture Notes in Computer Science*, pages 435–459. Springer, 2021. `doi:10.1007/978-3-030-79416-3_27`.

Testing Sumsets Is Hard

Xi Chen
Department of Computer Science, Columbia University, New York, NY, USA

Shivam Nadimpalli
Department of Mathematics, MIT, Cambridge, MA, USA

Tim Randolph
Department of Computer Science, Harvey Mudd College, Claremont, CA, USA

Rocco A. Servedio
Department of Computer Science, Columbia University, New York, NY, USA

Or Zamir
Blavatnik School of Computer Science, Tel Aviv University, Israel

Abstract

A subset S of the Boolean hypercube $\mathbb{F}_2^n$ is a *sumset* if $S = \{a + b : a, b \in A\}$ for some $A \subseteq \mathbb{F}_2^n$. Sumsets are central objects of study in additive combinatorics, where they play a role in several of the field's most important results. We prove a lower bound of $\Omega(2^{n/2})$ for the number of queries needed to test whether a Boolean function $f : \mathbb{F}_2^n \to \{0, 1\}$ is the indicator function of a sumset, ruling out an efficient testing algorithm for sumsets.

Our lower bound for testing sumsets follows from sharp bounds on the related problem of *shift testing*, which may be of independent interest. We also give a near-optimal $2^{n/2} \cdot \mathrm{poly}(n)$-query algorithm for a smoothed analysis formulation of the sumset *refutation* problem. Finally, we include a simple proof that the number of different sumsets in $\mathbb{F}_2^n$ is $2^{(1\pm o(1))2^{n-1}}$.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Sumsets, additive combinatorics, property testing, Boolean functions

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.14

Related Version *Full Version*: https://arxiv.org/abs/2401.07242

Funding *Xi Chen*: Supported in part by NSF awards CCF-2106429 and CCF-2107187.
Shivam Nadimpalli: Supported in part by NSF grants CCF-2106429, CCF-2211238, CCF-1763970, and CCF-210718.
Rocco A. Servedio: Supported in part by NSF awards CCF-2211238 and CCF-2106429.
Or Zamir: Supported in part by the Israel Science Foundation, Grant No. 1593/24, and in part by the Blavatnik Family Foundation.

1 Introduction

In recent years, theoretical computer science has increasingly been influenced by ideas and techniques from *additive combinatorics*, a field sitting at the intersection of combinatorics, number theory, Fourier analysis, and ergodic theory. Notable examples of this connection include communication complexity [16, 9, 14], constructions of randomness extractors [13, 7, 31, 21], and property testing [27, 32]; we also refer the reader to various surveys on additive combinatorics from the vantage point of theoretical computer science [8, 35, 36, 10, 30].

Among the simplest objects of study in additive combinatorics are *sumsets*: A subset S of an abelian group G (with group operation "+") is said to be a *sumset* if $S = A + A$ for some $A \subseteq G$, where for sets $A, B \subseteq G$ we write $A + B$ to denote the set $\{a + b : a \in A, b \in B\}$. Sumsets play a major role in additive combinatorics, where their study has led to many

© Xi Chen, Shivam Nadimpalli, Tim Randolph, Rocco A. Servedio, and Or Zamir;
licensed under Creative Commons License CC-BY 4.0

questions and insights about the additive structure of subsets of abelian groups. They are the subject of touchstone results in the field such as Freiman's theorem [23], which (roughly speaking) says that if $|A+A|$ is "not too much larger" than $|A|$ then A must be contained in a generalized arithmetic progression which is "not too large."

Our main interest in this paper is in *algorithmic* questions related to sumsets. In [22] Fagnot, Fertin, and Vialette considered the 2-Sumset Cover problem: given a set S of integers, does there exist a set A of cardinality at most k such that $S \subseteq A+A$? They proved APX-hardness for this problem and presented a $\text{poly}(k) \cdot 5^{k^2(k+3)/2}$-time algorithm. The latter was improved to $\text{poly}(k) \cdot 2^{(3\log k - 1.4)k}$ by Bulteau, Fertin, Rizzi, and Vialette [15]. 2-Sumset Cover itself specializes Generating Set, in which the goal is to find a minimal set A such that $S \subseteq \{\sum_{i \in I} i \; ; \; I \subseteq A\}$ and which was studied in [17]. Given S and k, finding a set A of size $|A| \geq k$ with $A+A \subseteq S$ is equivalent to finding a k-Clique on the Cayley sum graph of S; this problem remains NP-hard, but can be solved with existing algorithms for k-clique [25]. Recently, Abboud, Fischer, Safier, and Wallheimer proved that recognizing sumsets is NP-complete [1], settling a question raised by Granville [18].

1.1 This Work

In this paper we restrict our attention to the case in which the ambient abelian group G is $\mathbb{F}_2^n$. We do this for several reasons: first, given that our focus is on algorithmic problems, $\mathbb{F}_2^n$ is a very natural domain to consider from a theoretical computer science perspective. Another motivation is that $\mathbb{F}_2^n$ is in some sense the simplest setting for many problems involving sumsets; as Green stated in [27], "the reason that finite field models are nice to work with is that one has the tools of linear algebra, including such notions as subspace and linear independence, which are unavailable in general abelian groups." Indeed, several of our arguments use these linear-algebraic tools.

Since $\mathbb{F}_2^n$ is an exponentially large domain, it is natural to approach the study of sumsets over $\mathbb{F}_2^n$ from the vantage point of *sublinear* algorithms. Thus, we will be interested in algorithms for which either the running time or the number of calls to an oracle for the input set S (i.e. queries of the form "does element x belong to the set S?" is less than 2^n. The recent work [19] took such a sublinear-algorithms perspective; it studied a problem which was closely related to the problem of approximating the size of the sumset $A+A$, given access to an oracle for the unknown set $A \subseteq \mathbb{F}_2^n$. The main result of [19] was that in fact $O_\varepsilon(1)$ queries – in particular, with no dependence on the dimension parameter n – are sufficient for a $\pm\varepsilon \cdot 2^n$-accurate approximation of the quantity that they consider. This naturally motivates the following broad question: What other algorithmic problems involving sumsets may be solvable with "constant" (depending only on ε) or very low query complexity?

Motivated by this general question, in the current work we study a number of algorithmic questions related to sumsets. The main problems we consider are described below:

1. We study (approximate) sumset recognition from a property testing perspective. In more detail, given access to a membership oracle for an unknown set $S \subseteq \mathbb{F}_2^n$, in the *sumset testing problem* the goal is to output "yes" with high probability (say, at least 9/10) if S is a sumset and "no" with high probability if S is ε-far from every sumset (i.e. $|S \triangle (A+A)| \geq \varepsilon 2^n$ for every set $A \subseteq \mathbb{F}_2^n$), while making as few queries to the oracle as possible.
2. The above-described sumset testing problem turns out to be closely related to the problem of *shift testing*, which is defined as follows: A shift testing algorithm is given black-box access to two oracles $\mathcal{O}_A, \mathcal{O}_B : \mathbb{F}_2^n \to \{0,1\}$, which should be viewed as membership oracles for two subsets $A, B \subseteq \mathbb{F}_2^n$. The algorithm must output "yes" with probability at

least 9/10 if $B = A + \{z\}$ for some string $z \in \mathbb{F}_2^n$ and must output "no" with probability at least 9/10 if the symmetric difference $B \triangle (A + \{z\})$ has size at least $\varepsilon 2^n$ for every $z \in \mathbb{F}_2^n$.

3. For $S \subseteq \mathbb{F}_2^n$, let $\boldsymbol{N}_\varepsilon(S)$ denote a random set which is an "ε-noisy" version of S, obtained by flipping the membership / non-membership of each $x \in \mathbb{F}_2^n$ in S with probability ε. It can be shown that for every $S \subseteq \mathbb{F}_2^n$ and every constant $0 < \varepsilon < 1$, the noisy set $\boldsymbol{N}_\varepsilon(S)$ is with high probability not a sumset. We study the problem of algorithmically *certifying* that $\boldsymbol{N}_\varepsilon(S)$ is not a sumset; i.e. we are given access to a membership oracle for $\boldsymbol{N}_\varepsilon(S)$, where S is an arbitrary and unknown subset of $\mathbb{F}_2^n$, and the goal is to output a set $C \subseteq \mathbb{F}_2^n$ of points such that there is no sumset $A + A$ for which $(A + A) \cap C = \boldsymbol{N}_\varepsilon(S) \cap C$. We refer to this problem as the *smoothed sumset refutation problem,* since it aligns with the well-studied framework of smoothed analysis [34] in which an arbitrary worst-case instance is subjected to a mild perturbation.

The latter is also of non-algorithmic interest, and for completeness we also present (in Section 3) a short proof that the number of different sumsets in $\mathbb{F}_2^n$ is between $2^{2^{n-1}}$ and $2^{2^{n-1}+O(n^2)}$. While an upper bound on the number of such sumsets could have been previously deduced from Theorem 3 in [33], our proof is simpler and gives tighter bounds. Following this work, Alon and Zamir recently further improved these bounds [6].

Our main results are as follows.

Sumset Testing Lower Bound

We give an $\Omega(2^{n/2})$ lower bound on the query complexity of sumset testing:

▶ **Theorem 1.** *There is a constant $\varepsilon > 0$ (independent of n) such that any algorithm $\mathcal{A}$ for the ε-sumset testing problem must make $\Omega(2^{n/2})$ oracle calls.*

Theorem 1 holds even for adaptive testers which may make two-sided error. In particular, note that Theorem 1 rules out the possibility of an efficient tester for the property of being a sumset. Recall that in the property testing literature, "efficient" testers are often defined as algorithms that make a number of queries that depend only on the distance parameter ϵ, or occasionally also polylogarithmic in the problem size. For example, the seminal Blum-Luby-Rubinfeld linearity tester [12] determines if a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is linear or ε-far from all linear functions by making $O_\varepsilon(1)$ queries to the function. Other examples include testing for juntas [11], low-degree functions [20], and testing monotonicity [29].

Tight Bounds for Shift Testing

We show that the query complexity of shift testing is $\Theta^*(2^{n/2})$.

▶ **Theorem 2.** *(1) There is an algorithm for the shift testing problem which makes $O(n2^{n/2}/\varepsilon)$ oracle calls and runs in time $\mathrm{poly}(n) \cdot 2^n/\varepsilon$. Moreover, (2) For any constant $0 < c < 1/2$, any algorithm for the shift testing problem must make $\Omega(2^{n/2})$ oracle calls, even for $\varepsilon = (1/2 - 1/2^{cn})$. This lower bound holds even for distinguishing the following two cases: (i) A is a uniform random subset of $\mathbb{F}_2^n$ and $B = A + \{z\}$ for a uniform random $z \in \mathbb{F}_2^n$; versus (ii) A and B are independent uniform random subsets of $\mathbb{F}_2^n$.*

Like Theorem 1, the lower bound, i.e. Part (2), of Theorem 2 holds even for adaptive testers which may make two-sided error.

A Near-Optimal Algorithm for "Smoothed" Sumset Refutation

Our final result is a near-optimal algorithm which certifies that any noisy set $\boldsymbol{N}_\varepsilon(S)$ is not a sumset:

▶ **Theorem 3.** *(1) There is an algorithm for the ε-smoothed sumset refutation problem that makes $2^{n/2} \cdot O(n^{1.5}/\epsilon^{1.5})$ oracle calls and succeeds in certifying that $\boldsymbol{N}_\epsilon(S)$ is not a sumset with probability $1 - o_n(1)$. Moreover, (2) for any constant $\varepsilon > 0$, any algorithm that certifies that $\boldsymbol{N}_\epsilon(S)$ is not a sumset must make $\Omega(2^{n/2}/\sqrt{n})$ many oracle calls.*

1.2 Technical Overview

The $\mathbb{F}_2^n$ testing setting allows us to employ algorithms that are conceptually straightforward, even if proving correctness requires some care.

The main idea of our algorithm for shift testing (Part (1) of Theorem 2) is to query one oracle with all shifts of a random point r by a subspace V, and query the other oracle by all shifts of the same point r by the orthogonal complement $V^\perp$. This requires only $O(2^{n/2})$ queries, while providing information about the relationship between the two oracles vis-a-vis any possible shift $z \in \mathbb{F}_2^n$, since every possible shift has a decomposition into $z = z_1 + z_2$ for some $z_1 \in V, z_2 \in V^\perp$.

The optimality of this general approach is witnessed by the lower bound in Part (2) of Theorem 2. The proof is by a "deferred decisions" argument which analyzes the knowledge transcript of a query algorithm which may be interacting either with the "yes"-pair of oracles or the "no"-pair of oracles. We describe a coupling of the knowledge transcripts between these two cases, and use it to argue that if fewer than $0.1 \cdot 2^{n/2}$ queries have been made, then with high probability the transcripts are identically distributed across these two cases. (See [26] for a similar high-level argument, though in an entirely different technical setting.)

The $\Omega(2^{n/2})$ lower bound of Theorem 1 for sumset testing is by a reduction to the lower bound for shift testing. We give a straightforward embedding of the "A is random, $B = A + \{z\}$"-versus-"A, B are independent random" shift testing problem over $\mathbb{F}_2^n$ into the problem of sumset testing over $\mathbb{F}_2^{n+2}$. The most challenging part of the argument is to prove that in fact the "no" instances of shift testing (when A, B are independent random sets) give rise to instances which are far from sumsets over $\mathbb{F}_2^{n+2}$. This requires us to argue that a subset of $\mathbb{F}_2^{n+2}$ which is constant on two n-dimensional cosets and is uniform random on the other two n-dimensional cosets, is likely to be far from every sumset, which we prove using a linear algebraic argument.

For Theorem 3, a result due to Alon establishes that every subset of the Boolean cube of size $2^n - c2^{n/2}/\sqrt{n}$ is a sumset for a small constant c, which implies that any sumset "0-certificate" has size $\Omega^*(2^{n/2})$ [3]. To find such a certificate, we show that few noisy sumsets are likely to be consistent with an arbitrarily chosen subspace of dimension $n/2$, and then use a small random sample to rule out these sumsets with high probability.

1.3 Discussion

Our results suggest many questions and goals for future work; we record two such directions here.

The first direction is to obtain stronger results on sumset refutation. Is it possible to strengthen our sumset refutation result by eliminating the "smoothed analysis" aspect, i.e. is it the case that any $S \subseteq \mathbb{F}_2^n$ that is ε-far from every sumset has a "0-certificate" of size $2^{n/2} \cdot \mathrm{poly}\left(n, \frac{1}{\varepsilon}\right)$? If so, can such certificates be found efficiently given query access to S?

The second, and perhaps most compelling, direction is to either strengthen our $\Omega(2^{n/2})$-query lower bound, or prove an upper bound, for the sumset testing problem. We are cautiously optimistic that the true query complexity of sumset testing may be closer to $2^{n/2}$ queries than to 2^n queries, but any nontrivial ($o(2^n)$-query) algorithm would be an interesting result. One potentially relevant intermediate problem towards sumset testing is the problem of *k-shift testing*, in which the goal is to determine whether oracles $\mathcal{O}_A, \mathcal{O}_B : \mathbb{F}_2^n \to \{0,1\}$ correspond to $B = A + \{s_1, \dots, s_k\}$ for some k "shift" vectors $(s_i)_{i\in[k]}$ versus B being ε-far from every union of k shifts of A.

2 Preliminaries

All probabilities and expectations will be with respect to the uniform distribution, unless otherwise indicated. We use boldfaced characters such as $\boldsymbol{x}, \boldsymbol{f}$, and $\boldsymbol{A}$ to denote random variables (which may be real-valued, vector-valued, function-valued, or set-valued; the intended type will be clear from the context). We write $\boldsymbol{x} \sim \mathcal{D}$ to indicate that the random variable $\boldsymbol{x}$ is distributed according to the probability distribution $\mathcal{D}$. We write $\mathrm{dist}_{\mathrm{TV}}(\mathcal{D}_1, \mathcal{D}_2)$ to denote the *total variation distance* or *statistical distance* between the distributions $\mathcal{D}_1$ and $\mathcal{D}_2$.

For $\epsilon \in [0,1]$, we write $\boldsymbol{R}_\varepsilon$ to denote a random subset of $\mathbb{F}_2^n$ obtained by selecting each element with probability ϵ, so the "ε-noisy version" of a set $S \subseteq \mathbb{F}_2^n$, denoted $\boldsymbol{N}_\varepsilon(S)$, is equivalent to $S \triangle \boldsymbol{R}_\varepsilon$, where $A \triangle B := (A \setminus B) \cup (B \setminus A)$ denotes the symmetric difference of A and B.

Given a set $A \subseteq \mathbb{F}_2^n$, we will write $\mathcal{O}_A : \mathbb{F}_2^n \to \{0,1\}$ to denote the membership oracle for A, i.e.

$$\mathcal{O}_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}$$

for $x \in \mathbb{F}_2^n$. Given $A, B \subseteq \mathbb{F}_2^n$, we write $\mathrm{dist}(A,B)$ for the normalized Hamming distance between the sets A and B, i.e.

$$\mathrm{dist}(A,B) := \frac{|A \triangle B|}{2^n} = \mathop{\mathbf{Pr}}_{\boldsymbol{x} \sim \mathbb{F}_2^n}[\mathcal{O}_A(\boldsymbol{x}) \neq \mathcal{O}_B(\boldsymbol{x})].$$

We will also write $A + B := \{a + b : a \in A, b \in B\}$. If one of the sets is a singleton, e.g. if $A = \{a\}$, we will sometimes write $a + B := \{a\} + B$ instead.

We write $H(x)$ to denote the binary entropy function $-x \log_2 x - (1-x)\log_2(1-x)$. Stirling's approximation gives us the following helpful identity:

$$\binom{n}{\alpha n} = \Theta^*(2^{H(\alpha)n}); \text{ or, equivalently, } \binom{2^n}{\alpha 2^n} = 2^{H(\alpha)2^n} \cdot 2^{\Theta(n)}. \tag{1}$$

Given a subset D of an Abelian group G, we write $\Gamma_G(D)$ to denote the *Cayley sum graph* of G with respect to the generator set D; that is, the graph on the vertex set G that contains the edge (x,y) if and only if $x + y \in D$. (Since the group we consider is $\mathbb{F}_2^n$, for us this is the same as the regular Cayley graph of G with respect to generator set D.) When $D = \{x\}$ is a singleton for some $x \in G$, we abuse notation slightly and write $\Gamma_G(x)$ for $\Gamma_G(\{x\})$.

3 The Number of Sumsets in $\mathbb{F}_2^n$

▶ **Proposition 4.** *The number of sumsets in $\mathbb{F}_2^n$ is at most*

$2^{2^{n-1}+O(n^2)}$.

Proof. Consider a sumset $S = A + A$; we consider two cases depending on the linear rank of the set $\mathbb{F}_2^n \setminus S$.

Case 1: $\mathbb{F}_2^n \setminus S$ does not have full rank. In other words, there exists a vector $v \in \mathbb{F}_2^n$ such that

$$\langle x, v\rangle = 1 \qquad \text{implies that} \qquad x \in S.$$

In particular, we have that $S = S' \cup \left(\mathbb{F}_2^n \setminus v^\perp\right)$ for some $S' \subseteq v^\perp = \{x \in \mathbb{F}_2^n : \langle x, v\rangle = 0\}$. As there are at most 2^n choices for v, and for each choice of v there are at most $2^{2^{n-1}}$ choices for S', we have that there are at most $2^{2^{n-1}+n}$ many sumsets of this form.

Case 2: $\mathbb{F}_2^n \setminus S$ has full rank. In particular, there are n linearly independent vectors *not* in S. For $v \in \mathbb{F}_2^n$, observe that the Cayley graph $\Gamma_{\mathbb{F}_2^n}(v)$ is a perfect matching. Next, note that if $v \notin S = A + A$, then A must be an independent set in $\Gamma_{\mathbb{F}_2^n}(v)$. This is because, if $x, y \in A$ with $x + v = y$ then

$$A + A \ni x + y = x + x + v = v \notin A + A,$$

which is a contradiction. As we have n linearly independent vectors not in S, it follows that there exists an orthogonal transformation of $\mathbb{F}_2^n$ such that A must be an independent set in the hypercube (where edges are incident to elements of $\mathbb{F}_2^n$ that differ in a single coordinate). As the number of independent sets in the hypercube Q_n is at most $2^{2^{n-1}+O(1)}$ (see for example [24]), and as the number of orthogonal transformations of the hypercube is at most 2^{n^2}, it follows that the total number of sumsets of this form is at most

$$2^{2^{n-1}+n^2+O(1)}.$$

Both cases together complete the proof. ◀

▶ **Proposition 5.** *The number of sumsets in $\mathbb{F}_2^n$ is at least $2^{2^{n-1}}$.*

Proof. For any subset $A \subseteq \mathbb{F}_2^{n-1}$ of the $(n-1)$-th dimensional hypercube, we define a subset $A' \subseteq \mathbb{F}_2^n$ as

$$A' := \{\vec{0}\} \cup \{(1, a) \mid a \in A\},$$

where for a $(n-1)$-dimensional vector a, the concatenation $(1, a)$ is defined as the n-dimensional vector where the first coordinate is 1 and the other $(n-1)$ coordinates are equal to a. We observe that $(A' + A') \cap \left(\mathbb{F}_2^n \setminus e_1^\perp\right) = \{(1, a) \mid a \in A\}$. That is, in the sumset $(A' + A')$ all vectors in which the first coordinate is 1 exactly correspond to the set A. In particular, for any $A_1 \neq A_2 \in \mathbb{F}_2^{n-1}$, we have $(A_1' + A_1') \neq (A_2' + A_2')$. ◀

4 Optimally Testing Shifts

Given $A, B \subseteq \mathbb{F}_2^n$, we say that B is a *shift* of A if there exists $z \in \mathbb{F}_2^n$ such that $A + z = B$. We obtain the following upper and lower bounds for the *shift testing* problem:

▶ **Theorem 6.** *Let $\mathcal{O}_A, \mathcal{O}_B : \mathbb{F}_2^n \to \{0,1\}$ be membership oracles for $A, B \subseteq \mathbb{F}_2^n$. Then:*

1. *The algorithm* Shift-Tester *(Algorithm 1) makes $O(n2^{n/2}/\varepsilon)$ oracle calls, runs in time $\mathrm{poly}(n) \cdot 2^n/\varepsilon$, and guarantees that:*
   - **a.** *If $B = A + z$ for some $z \in \mathbb{F}_2^n$, the algorithm outputs "shift" with probability $9/10$;*
   - **b.** *If for every $z \in \mathbb{F}_2^n$ we have $\mathrm{dist}(A + z, B) \geq \varepsilon$, the algorithm outputs "ε-far from shift" with probability $9/10$.*

2. *Fix c to be a constant that is less than $1/2$. Any (adaptive, randomized) algorithm with the performance guarantee in the previous item makes $\Omega(2^{n/2})$ oracle calls, even for $\varepsilon = 1/2 - 1/2^{cn}$.*

In fact, the lower bound holds even for distinguishing the following two cases: (i) A is a uniform random subset of $\mathbb{F}_2^n$ and $B = A + s$ for a uniform random $s \in \mathbb{F}_2^n$; versus (ii) A and B are independent uniform random subsets of $\mathbb{F}_2^n$.

4.1 Upper Bound

In this section, we prove Item 1 of Theorem 6. Note that since

$$\operatorname{dist}(B, A + z) = \mathop{\mathbf{Pr}}_{\boldsymbol{x}}\left[\mathcal{O}_A(\boldsymbol{x}) \neq \mathcal{O}_B(\boldsymbol{x} + z)\right],$$

if B is a shift of A (i.e. $B = A + z_*$ for some z_*), we then have for that z_* that

$$\mathop{\mathbf{Pr}}_{\boldsymbol{x}}\left[\mathcal{O}_B(\boldsymbol{x}) = \mathcal{O}_A(\boldsymbol{x} + z_*)\right] = 1.$$

On the other hand, if $\operatorname{dist}(B, A + z) \geq \varepsilon$ for every z, then for every z we have

$$\mathop{\mathbf{Pr}}_{\boldsymbol{x}}\left[\mathcal{O}_B(\boldsymbol{x}) = \mathcal{O}_A(\boldsymbol{x} + z)\right] \leq 1 - \varepsilon.$$

These simple observations suggest that in order to estimate $\Pr[\mathcal{O}_B(\boldsymbol{x}) = \mathcal{O}_A(\boldsymbol{x} + z)]$ for a particular z, we would like to make queries $\mathcal{O}_B(\boldsymbol{x}), \mathcal{O}_A(\boldsymbol{x} + z)$ for uniform random $\boldsymbol{x}$. The fact that we need to do this for all z motivates the following approach; before proceeding, we introduce some notation.

▶ **Notation 7.** *We define the subsets $D_1, D_2 \subset \mathbb{F}_2^n$, where D_1 is the set of all $2^{\lfloor n/2 \rfloor}$ vectors whose last $\lfloor n/2 \rfloor$ coordinates are all-0 and $D_2 \subset \mathbb{F}_2^n$ is the set of $2^{\lceil n/2 \rceil}$ vectors whose first $\lceil n/2 \rceil$ coordinates are all-0. Note that every $z \in \mathbb{F}_2^n$ has a unique expression as*

$$z := z^{(1)} + z^{(2)}, \qquad \text{for } z^{(1)} \in D_1 \text{ and } z^{(2)} \in D_2.$$

Fix a particular string $z = z^{(1)} + z^{(2)}$ as above. We write $\boldsymbol{x} = \boldsymbol{r} + z^{(1)}$, and we observe that if $\boldsymbol{r}$ is uniform random then so is $\boldsymbol{x}$. As alluded to earlier we would like to query B on $\boldsymbol{x}$ and A on $\boldsymbol{x} + z = \boldsymbol{r} + z^{(1)} + z = \boldsymbol{r} + z^{(2)}$. The main observation is that if we query B on all strings in $D_1 + \boldsymbol{r}$ and query A on all strings in $D_2 + \boldsymbol{r}$, then no matter what z is we will have made the queries $\mathcal{O}_B(\boldsymbol{x}) = \mathcal{O}_B(\boldsymbol{r} + z^{(1)})$ and $\mathcal{O}_A(\boldsymbol{x} + z) = \mathcal{O}_A(\boldsymbol{r} + z^{(2)})$, so we will have obtained a sample towards estimating $\Pr_{\boldsymbol{x}}[\mathcal{O}_B(\boldsymbol{x}) = \mathcal{O}_A(\boldsymbol{x} + z)]$. Since this is true for every z, we can reuse the above queries towards all possibilities for z. (Of course one sample is not enough to estimate a probability, so we will repeat the above with n/ε different choices of $\boldsymbol{r}$.)

Proof of Item 1 of Theorem 6. Our algorithm, Shift-Tester, is presented in Algorithm 1. Note that if B is a shift of A, i.e. if there exists a $z_* \in \mathbb{F}_2^n$ for which $B = A + z_*$ then

$$\boldsymbol{r} + z_*^{(1)} \in A \qquad \text{if and only if} \qquad \boldsymbol{r} + z_*^{(1)} + z_* = \boldsymbol{r} + z_*^{(2)} \in B,$$

where we used the fact that $z_*^{(1)} + z_* = z_*^{(1)} + z_*^{(1)} + z_*^{(2)} = z_*^{(2)}$. In particular, we will have $p_{z_*} = 1$ and so the algorithm will return "shift" with probability 1. On the other hand, suppose B is ε-far from $A + z$ for every $z \in \mathbb{F}_2^n$; fix any such z. Then the probability that all n/ε repetitions in Algorithm 1 will have $\mathcal{O}_A(\boldsymbol{r} + z^{(1)}) = \mathcal{O}_B(\boldsymbol{r} + z^{(2)})$ is at most

$$(1 - \varepsilon)^{n/\varepsilon} \leq e^{-n}.$$

Taking a union bound over all $z \in \mathbb{F}_2^n$ implies that the probability that Algorithm 1 will output "ε-far from shift" is at least $1 - (2/e)^n$, completing the proof. ◀

Algorithm 1 An algorithm for shift testing.

Input: Oracles $\mathcal{O}_A, \mathcal{O}_B : \mathbb{F}_2^n \to \{0,1\}$ and $\epsilon > 0$
Output: "Shift" or "ε-far from shift"

SHIFT-TESTER$(\mathcal{O}_A, \mathcal{O}_B, \epsilon)$:

1. Repeat the following n/ϵ times:
   a. Draw a uniformly random $\boldsymbol{r} \in \mathbb{F}_2^n$.
   b. Query $\mathcal{O}_A$ on all $x \in D_1 + \boldsymbol{r}$, and query $\mathcal{O}_B$ on all $y \in D_2 + \boldsymbol{r}$.
2. For each $z = z^{(1)} + z^{(2)} \in \mathbb{F}_2^n$, let p_z be the fraction of the n/ε iterations for which

$$\mathcal{O}_A(\boldsymbol{r} + z^{(1)}) = \mathcal{O}_B(\boldsymbol{r} + z^{(2)}).$$

3. If $p_z = 1$ for some $z \in \mathbb{F}_2^n$, output "shift"; otherwise output "ε-far from shift".

Note that Algorithm 1 in fact guarantees 1-sided error, stronger than what is required by Theorem 6: The algorithm never outputs "ε-far from shift" if B is a shift of A, and if B is ε-far from every shift of A then the algorithm outputs "shift" with probability at most $(2/e)^n$.

4.2 Lower Bound

To prove Item 2 of Theorem 6 we define two probability distributions, $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$, over instances of the shift testing problem.

▶ **Definition 8.** *A draw $(\boldsymbol{A}, \boldsymbol{B})$ from $\mathcal{D}_{\text{yes}}$ is obtained as follows:*
- *$\boldsymbol{A} \subseteq \mathbb{F}_2^n$ includes each element of $\mathbb{F}_2^n$ independently with probability 1/2.*
- *$\boldsymbol{B} \subseteq \mathbb{F}_2^n$ equals $\boldsymbol{A} + \boldsymbol{s}$ for $\boldsymbol{s}$ sampled uniformly at random from $\mathbb{F}_2^n$.*

Note that for $(\boldsymbol{A}, \boldsymbol{B}) \sim \mathcal{D}_{\text{yes}}$, $\boldsymbol{B}$ is a shift of $\boldsymbol{A}$.

▶ **Definition 9.** *A draw $(\boldsymbol{A}, \boldsymbol{B})$ from $\mathcal{D}_{\text{no}}$ is obtained as follows:*
- *$\boldsymbol{A} \subseteq \mathbb{F}_2^n$ includes each element of $\mathbb{F}_2^n$ independently with probability 1/2.*
- *$\boldsymbol{B} \subseteq \mathbb{F}_2^n$ also includes each element of $\mathbb{F}_2^n$ with probability 1/2 (independently of $\boldsymbol{A}$).*

A straightforward application of the Chernoff bound, combined with a union bound over the 2^n possible shifts, shows that with probability at least 19/20 a draw of $(\boldsymbol{A}, \boldsymbol{B}) \sim \mathcal{D}_{\text{no}}$ is such that $\boldsymbol{B}$ is $(1/2 - 1/2^{cn})$-far from every shift of $\boldsymbol{A}$ (for any constant $c < 1/2$). So to prove Item 2 of Theorem 2, it is enough to establish the following claim for deterministic algorithms. (By Yao's minimax principle, this is sufficient to prove a lower bound for randomized algorithms as well.)

▷ Claim 10. Let `Test` be any deterministic, adaptive algorithm that makes $N := 0.1 \cdot 2^{n/2}$ oracle calls to $\mathcal{O}_A$ and $\mathcal{O}_B$. Let $T_{\texttt{test}}(A, B)$ be the "transcript" of its queries to the oracles and received responses, i.e. $T_{\texttt{test}}(A, B)$ consists of

$$\begin{gathered}(\text{first query to one of the oracles, response received})\\ \vdots\\ (N\text{-th query to one of the oracles, response received}).\end{gathered}$$

Then we have

$$\text{dist}_{\text{TV}}\left(T_{\texttt{test}}(\boldsymbol{A}_{\text{yes}}, \boldsymbol{B}_{\text{yes}}), T_{\texttt{test}}(\boldsymbol{A}_{\text{no}}, \boldsymbol{B}_{\text{no}})\right) \leq 0.02,$$

where $(\boldsymbol{A}_{\text{yes}}, \boldsymbol{B}_{\text{yes}}) \sim \mathcal{D}_{\text{yes}}$ and $(\boldsymbol{A}_{\text{no}}, \boldsymbol{B}_{\text{no}}) \sim \mathcal{D}_{\text{no}}$.

The claim follows by analyzing the behavior of the algorithm on an oracle constructed over the course of answering the queries posed by algorithm `Test`, i.e., "deferring" the decision of whether the oracle $(\boldsymbol{A}, \boldsymbol{B})$ is drawn from $\mathcal{D}_{\text{yes}}$ or $\mathcal{D}_{\text{no}}$. See for example Section 7.1 of [26].

Proof. For simplicity, we assume that in each round, `Test` queries one point q and receives *both* $\mathcal{O}_A(q)$ and $\mathcal{O}_B(q)$; this can only make `Test` more powerful.

Consider the following approach to answering queries posed by `Test`: before any queries are made, draw a uniform random $\boldsymbol{s} \sim \mathbb{F}_2^n$. Let $q_1, \ldots, q_{t-1} \in \mathbb{F}_2^n$ be the first $t-1$ queries made by `Test` (we may suppose without loss of generality that all these $t-1$ query strings are different from each other, since any algorithm that repeats a query string can easily be modified so as not to do so). When the t-th query string q_t is provided by `Test`, the answer is generated as follows:

1. If $\boldsymbol{s} \neq q_t + q_{t'}$ for all $t' \leq t$, then two independent uniform random bits $\boldsymbol{b_A}, \boldsymbol{b_B} \in \{0,1\}$ are drawn and returned as $\mathcal{O}_{\boldsymbol{A}}(q_t)$ and $\mathcal{O}_{\boldsymbol{B}}(q_t)$. (It may be helpful to think of this outcome as being "recorded", i.e. when this happens the process "decides" that $\boldsymbol{b_A}, \boldsymbol{b_B}$ are the values of $\boldsymbol{A}$ and $\boldsymbol{B}$ on the point q_t.)
2. If $\boldsymbol{s} = q_t + q_{t'}$ for some $t' \leq t$, then the process halts and outputs "failure."

The key observation is that conditioned on the above process proceeding through t queries without an output of "failure", the length-t transcript is distributed exactly according to the pair of oracles $(\boldsymbol{A}, \boldsymbol{B})$ being $(\boldsymbol{A}_{\text{yes}}, \boldsymbol{B}_{\text{yes}}) \sim \mathcal{D}_{\text{yes}}$, and also exactly according to the pair of oracles being $(\boldsymbol{A}_{\text{no}}, \boldsymbol{B}_{\text{no}}) \sim \mathcal{D}_{\text{no}}$. This is because in either case, as long as no pair of queries $q_t, q_{t'}$ sum to the "hidden" random string $\boldsymbol{s} \in \mathbb{F}_2^n$, every response to every oracle call is distributed as an independent uniform random bit.

We finish the proof by showing that the probability that the process above outputs "failure" is at most 0.02. We emphasize that the probability here is taken over the entire random process which includes the initial uniform random draw of $\boldsymbol{s} \sim \mathbb{F}_2^n$.

To this end, we note that conditioning on no "failure" during the first $t-1$ rounds $q_1, \ldots, q_{t-1}$, $\boldsymbol{s}$ is distributed uniformly among all points in $\mathbb{F}_2^n$ that are not equal to $q_i + q_j$ for some $i, j \in [t-1]$. The number of such points is at least $2^n - N^2/2 > 0.99 \cdot 2^n$. On the other hand, the process outputs "failure" in round t if one of $q_1 + q_t, \ldots, q_{t-1} + q_t$ is $\boldsymbol{s}$, which happens with probability at most $N/(0.99 \cdot 2^n) < 0.2 \cdot 2^{-n/2}$. It follows from a union bound on the N rounds that the process outputs "failure" with probability at most 0.02. This finishes the proof of the claim. ◁

5 Lower Bound for Testing Sumsets

In this section, we show that the lower bound for shift testing established in Section 4.2 implies a lower bound for the problem of testing sumsets. More formally, we prove the following:

▶ **Theorem 11.** *Let $\mathcal{O}_S : \mathbb{F}_2^n \to \{0,1\}$ be a membership oracle for $S \subseteq \mathbb{F}_2^n$. There is an absolute constant $\varepsilon > 0.0125$ such that the following holds: Let $\mathcal{A}$ be any (adaptive, randomized) algorithm with the following performance guarantee:*

1. *If $S = A + A$ for some $A \subseteq \mathbb{F}_2^n$, $\mathcal{A}$ outputs "sumset" with probability $9/10$; and*
2. *If $\operatorname{dist}(S, A+A) \geq \epsilon$ for all $A \subseteq \mathbb{F}_2^n$, $\mathcal{A}$ outputs "ε-far from sumset" with probability $9/10$.*

Then $\mathcal{A}$ must make $\Omega(2^{n/2})$ calls to $\mathcal{O}_S$.

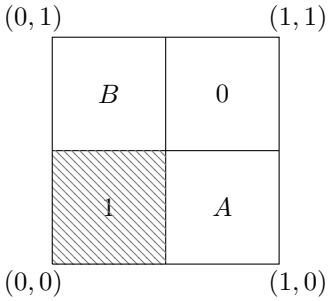


Figure 1 The set $\mathcal{S}(A, B) \subseteq \mathbb{F}_2^{n+2}$. By Proposition 12, for a typical $(\boldsymbol{A}_{\text{yes}}, \boldsymbol{B}_{\text{yes}})$ drawn from $\mathcal{D}_{\text{yes}}$, adding a single point $(1, 1, \boldsymbol{s})$ in the top right cell makes $S(\boldsymbol{A}_{\text{yes}}, \boldsymbol{B}_{\text{yes}})$ into a sumset.

The distributions we use to prove Theorem 11 are based on the distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ defined in Definitions 8 and 9 for shift testing. Given $A, B \subseteq \mathbb{F}_2^n$, we define $\mathcal{S}(A, B) \subseteq \mathbb{F}_2^{n+2}$ as

$$\mathcal{S}(A, B) := \{x : x_1 = x_2 = 0\} \sqcup \{(1, 0, a) : a \in A\} \sqcup \{(0, 1, b) : b \in B\}, \tag{2}$$

where the notation (b_1, b_2, v) indicates that the bits b_1 and b_2 are concatenated with $v \in \mathbb{F}_2^n$ to create an element in $\mathbb{F}_2^{n+2}$. Figure 1 illustrates the set $\mathcal{S}(A, B)$.

We use $\mathcal{D}_{\text{no}}$ to define $\mathcal{S}_{\text{no}}$, a distribution over subsets of $\mathbb{F}_2^{n+2}$ as follows: To draw $\boldsymbol{S}_{\text{no}} \sim \mathcal{S}_{\text{no}}$, we draw $(\boldsymbol{A}_{\text{no}}, \boldsymbol{B}_{\text{no}}) \sim \mathcal{D}_{\text{no}}$ and set $\boldsymbol{S}_{\text{no}} = \mathcal{S}(\boldsymbol{A}_{\text{no}}, \boldsymbol{B}_{\text{no}})$. On the other hand, we use $\mathcal{D}_{\text{yes}}$ to define $\mathcal{S}_{\text{yes}}$ as follows: To draw $\boldsymbol{S}_{\text{yes}} \sim \mathcal{S}_{\text{yes}}$, we draw $(\boldsymbol{A}_{\text{yes}}, \boldsymbol{B}_{\text{yes}} = \boldsymbol{A}_{\text{yes}} + \boldsymbol{s}) \sim \mathcal{D}_{\text{yes}}$ but add one "extra" point to $\boldsymbol{S}_{\text{yes}}$, defining it as: $\boldsymbol{S}_{\text{yes}} = \mathcal{S}(\boldsymbol{A}_{\text{yes}}, \boldsymbol{B}_{\text{yes}}) \sqcup \{(1, 1, \boldsymbol{s})\}$. This will ensure that $\boldsymbol{S}_{\text{yes}} \sim \mathcal{S}_{\text{yes}}$ is likely to be a sumset (see Proposition 12 below).

At a high level, the proof of Theorem 11 contains three steps: we show (1) that $\boldsymbol{S}_{\text{yes}} \sim \mathcal{S}_{\text{yes}}$ is a sumset with high probability (Proposition 12), (2) that $\boldsymbol{S}_{\text{no}} \sim \mathcal{S}_{\text{no}}$ is ϵ-far from being a sumset with high probability (Proposition 13), and (3) that oracles to $\boldsymbol{S}_{\text{yes}} \sim \mathcal{S}_{\text{yes}}$ and $\boldsymbol{S}_{\text{no}} \sim \mathcal{S}_{\text{no}}$ are too similar for an algorithm that makes few queries to tell the difference, where "similarity" is measured in terms of the total variation distance between distributions over transcripts (proof of Theorem 11). The theorem then follows quickly from these three facts.

▶ **Proposition 12.** *With probability at least* $1 - 2^{-\Omega(2^n)}$, $\boldsymbol{S}_{\text{yes}} \sim \mathcal{S}_{\text{yes}}$ *is a sumset over* $\mathbb{F}_2^{n+2}$.

Proof. Let $(A_{\text{yes}}, B_{\text{yes}})$ be a pair of sets in the support of $\mathcal{D}_{\text{yes}}$ with $B_{\text{yes}} = A_{\text{yes}} + s$. It is easy to verify that $\mathcal{S}(A_{\text{yes}}, B_{\text{yes}}) \sqcup \{(1, 1, s)\}$ is equal to $C + C$ with

$$C := \{0^n\} \sqcup \{(1, 0, a) : a \in A_{\text{yes}}\} \sqcup \{(1, 1, s)\}.$$

as long as $A_{\text{yes}} + A_{\text{yes}}$ covers all of $\mathbb{F}_2^n$. So it suffices to show that this holds with extremely high probability with a uniformly random set $\boldsymbol{A}_{\text{yes}}$.

To see this, consider any fixed, nonzero element $z \in \mathbb{F}_2^n$. Without loss of generality, suppose that the first coordinate of z is 1. We have

$$\Pr\left[z \notin \boldsymbol{A}_{\text{yes}} + \boldsymbol{A}_{\text{yes}}\right] = \Pr\left[\text{for all } y \in \mathbb{F}_2^n, \text{ either } y \notin \boldsymbol{A}_{\text{yes}} \text{ or } z + y \notin \boldsymbol{A}_{\text{yes}}\right] = (3/4)^{2^{n-1}},$$

where the second equality holds because $\boldsymbol{A}_{\text{yes}}$ is a uniform random subset of $\mathbb{F}_2^n$ and $y, z + y$ are distinct elements (observe that the first coordinate of y is 0 while the first coordinate of $z + y$ is 1). Since $\Pr[0^n \notin \boldsymbol{A}_{\text{yes}} + \boldsymbol{A}_{\text{yes}}] = \Pr[\boldsymbol{A}_{\text{yes}} \text{ is empty}] = (1/2)^{2^n} < (3/4)^{2^{n-1}}$, we get that each fixed element $z \in \mathbb{F}_2^n$ is missing from $\boldsymbol{A}_{\text{yes}} + \boldsymbol{A}_{\text{yes}}$ with probability at most $(3/4)^{2^{n-1}}$. The claim follows from a union bound over the 2^n elements of $\mathbb{F}_2^n$. ◀

▶ **Proposition 13.** *With probability at least* $1 - o_n(1)$, $\boldsymbol{S}_{\text{no}} \sim \mathcal{S}_{\text{no}}$ *is* 0.0125*-far from every sumset.*

We now complete the proof of Theorem 11 using Propositions 12 and 13. The proof of Proposition 13 is deferred to Section 5.1.

Proof of Theorem 11. Let $\mathcal{A}$ be an algorithm for sumset testing on $\mathbb{F}_2^{n+2}$ that makes at most $N = 0.1 \cdot 2^{n/2}$ queries. As in the proof of Theorem 6, we let $T_{\mathcal{A}}(S)$ denote the N-element transcript of $\mathcal{A}$ given the oracle $\mathcal{O}_S$ and take a "deferred decision" approach to prove that $\mathcal{A}$ cannot distinguish between $\mathcal{S}_{\text{yes}}$ and $\mathcal{S}_{\text{no}}$ with high probability.

By Propositions 12 and 13, the probability that $\boldsymbol{S}_{\text{yes}} \sim \mathcal{S}_{\text{yes}}$ is not a sumset is $o_n(1)$, and the probability that $\boldsymbol{S} \sim \mathcal{S}_{\text{no}}$ is 0.0125-close to any sumset is $o_n(1)$. As a result, to prove Theorem 11 it suffices to show that

$$\text{dist}_{\text{TV}}\left(T_{\mathcal{A}}(\boldsymbol{S}_{\text{yes}}), T_{\mathcal{A}}(\boldsymbol{S}_{\text{no}})\right) < 0.1 - o_n(1),$$

where $\boldsymbol{S}_{\text{yes}} \sim \mathcal{S}_{\text{yes}}$ and $\boldsymbol{S}_{\text{no}} \sim \mathcal{S}_{\text{no}}$.

Consider the sham oracle $\mathcal{O}_{\text{sham}}$ that samples a point $\boldsymbol{s} \in \mathbb{F}_2^n$ uniformly at random, then responds to queries as follows:

1. If the query is a point q_t for which $q_{t,1} = q_{t,2} = 0$, the oracle returns 1.
2. If the query is $(1, 1, \boldsymbol{s})$, the oracle outputs "failure". Otherwise, if $q_{t,1} = q_{t,2} = 1$, it returns 0.
3. If the query is a point q_t such that $q_t + q_{t'} = (1, 1, \boldsymbol{s})$ for some previously queried point $q_{t'}$, the oracle outputs "failure". Otherwise, it returns a random bit.

We proceed to consider the behavior of $\mathcal{A}$ given $\mathcal{O}_{\text{sham}}$, $\mathcal{O}_{\boldsymbol{S}_{\text{yes}}}$, and $\mathcal{O}_{\boldsymbol{S}_{\text{no}}}$. Conditioned on the event that $\mathcal{O}_{sham}$ does not output "failure", $\mathcal{A}$ always receives the answer '1' when querying a point with initial coordinates $(0, 0)$, always receives the answer '0' when querying a point with initial coordinates $(1, 1)$, and receives a random bit when querying a point with the initial coordinates $(0, 1)$ or $(1, 0)$. If, after the point of "failure", our oracle subsequently responds to queries consistently with the distribution $\mathcal{S}(\boldsymbol{A}_{\text{yes}}, \boldsymbol{A}_{\text{yes}} + \boldsymbol{s})$, randomly determining membership in $\boldsymbol{A}_{\text{yes}}$ via deferred decision as necessary, the resulting distribution over transcripts is identical to that given oracle access to $\boldsymbol{S}_{\text{yes}}$. Likewise, if the oracle responds '0' on $(1, 1, s)$ and continues to return random bits on queries whose initial coordinates begin with $(0, 1)$ or $(1, 0)$, the resulting distribution over transcripts is identical to that given oracle access to $\boldsymbol{S}_{\text{no}}$. We conclude that the distribution of $T_{\mathcal{A}}(\text{sham})$, the transcript of $\mathcal{A}$ given $\mathcal{O}_{\text{sham}}$, is identical to the distribution of transcripts given $\mathcal{O}_{\boldsymbol{S}_{\text{yes}}}$ and $\mathcal{O}_{\boldsymbol{S}_{\text{no}}}$ unless failure occurs.

Failure is unlikely for any algorithm $\mathcal{A}$ that makes at most N queries: With N queries, the algorithm can rule out at most $N = O(2^{n/2})$ candidates for $\boldsymbol{s}$ by querying points with the initial coordinates $(1, 1)$, and at most $N^2 = 0.01 \cdot 2^n$ candidates for $\boldsymbol{s}$ by querying points with the initial coordinates $(0, 1)$ and $(1, 0)$. Conditioned on no failure, the posterior distribution of $\boldsymbol{s}$ is thus uniform over at least $(0.99 - o_n(1))2^n$ points. Thus subsequently querying a point discovers s with probability at most

$$\frac{N}{(0.99 - o_n(1))2^n} \leq \frac{0.2}{2^{n/2}}.$$

Union-bounding over all N rounds gives a failure probability of at most 0.02.

We conclude that

$$\text{dist}_{\text{TV}}(T_{\mathcal{A}}(\boldsymbol{S}_{\text{yes}}), T_{\mathcal{A}}(\boldsymbol{S}_{\text{no}})) \leq 0.02 + o_n(1),$$

and thus any algorithm that makes at most $N = 0.1 \cdot 2^{n/2}$ queries cannot answer correctly with probability 9/10. ◀

5.1 Proof of Proposition 13

We prove Proposition 13 via a counting argument. The distribution $\mathcal{S}_{\text{no}}$ produces subsets of $\mathbb{F}_2^{n+2}$ of a specific form: these subsets contain every point in the subspace $\{x : x_1 = x_2 = 0\}$, no points in the coset $\{x : x_1 = x_2 = 1\}$, and have density roughly 0.5 on the cosets $\{x : x_1 = 0, x_2 = 1\}$ and $\{x : x_1 = 1, x_2 = 0\}$. We first bound the number of sumsets that are ϵ-*eligible* (roughly, "close") to any subset of this form (Proposition 16). Since there are relatively few subsets of $\mathbb{F}_2^{n+2}$ near any ϵ-eligible sumset, we conclude that most subsets drawn from $\mathcal{S}_{\text{no}}$ are far from any sumset (Proposition 13).

▶ Remark 14. It can be shown that the number of sumsets in $\mathbb{F}_2^{n+2}$ is at most $2^{2^{n+1}+O(n^2)}$ (this bound is implicit in the work [33], and for completeness we give a proof in Section 3). However, this upper bound is not enough for us per se since the support of $\mathcal{S}_{\text{no}}$ is also of size $2^{2^{n+1}}$; hence we need to use the more refined notion of "ε-eligible" sumsets mentioned above.

In the remainder of this section, we make frequent reference to the volume of sets within the subspace $\{x : x_1 = x_2 = 0\}$ of $\mathbb{F}_2^{n+2}$ and its three cosets. Given a set $S \subseteq \mathbb{F}_2^{n+2}$ and a pair of bits $(b_1, b_2) \in \{0,1\}^2$, we define

$$\text{Vol}_{b_1 b_2}(S) := \frac{|S \cap \{x \in \mathbb{F}_2^{n+2} : x_1 = b_1, x_2 = b_2\}|}{2^n}.$$

in order to simplify notation.

▶ **Definition 15.** *Given $\varepsilon > 0$, we say that a set $S \subseteq \mathbb{F}_2^{n+2}$ is an* ε-eligible sumset *if $S = A + A$ for some $A \subseteq \mathbb{F}_2^{n+2}$ and if the following holds:*

$$\text{Vol}_{00}(S) \geq 1 - \varepsilon \qquad \textit{and} \qquad \text{Vol}_{11}(S) \leq \varepsilon.$$

Roughly, the ϵ-eligible sumsets are all those that might be close to $\boldsymbol{S}_{\text{no}} \sim \mathcal{S}_{\text{no}}$.

▶ **Proposition 16.** *For any ε, the number of ε-eligible sumsets in $\mathbb{F}_2^{n+2}$ is at most*

$$\max\left\{2^{4H(\epsilon)\cdot 2^n}, 2^{(1+2H(\epsilon))2^n}\right\} \cdot 2^{O(n)}.$$

We defer the proof of Proposition 16 to Appendix A. We conclude with the proof of Proposition 13.

Proof of Proposition 13. By Proposition 16, the number of ϵ-eligible sumsets is $2^{(1+2H(\epsilon))2^n} \cdot 2^{O(n)}$ when $\epsilon < 0.1$. By Equation (1), the number of *subsets* of $\mathbb{F}_2^{n+2}$ that are γ-close to a given sumset is

$$\binom{2^{n+2}}{\gamma 2^{n+2}} = 2^{H(\gamma)2^{n+2}} \cdot 2^{O(n)}.$$

Thus, by union-bounding over all ϵ-eligible sumsets, we conclude that the number of subsets of $\mathbb{F}_2^{n+2}$ that are γ-close to any ϵ-eligible sumset is at most

$$2^{(1+2H(\epsilon))2^n + H(\gamma)2^{n+2}} \cdot 2^{O(n)}.$$

Choosing $\epsilon = 0.05$ and $\gamma = \epsilon/4$ gives an upper bound of $2^{1.96\cdot 2^n}$ subsets of $\mathbb{F}_2^{n+2}$ that are $\epsilon/4$-close to any ϵ-eligible sumset. Since $\mathcal{S}_{\text{no}}$ is distributed uniformly over $2^{2^{n+1}}$ subsets, the probability that $\boldsymbol{S}_{\text{no}} \sim \mathcal{S}_{\text{no}}$ is $(\epsilon/4)$-close to any ϵ-eligible sumset is $2^{-\Omega(2^n)}$.

We further claim that $\boldsymbol{S}_{\text{no}} \sim \mathcal{S}_{\text{no}}$ is always $(\varepsilon/4)$-far from any sumset that is not ε-eligible. This is just because that we always have $\text{Vol}_{11}(\boldsymbol{S}_{\text{no}}) = 0$ and $\text{Vol}_{00}(\boldsymbol{S}_{\text{no}}) = 1$. On the other hand, any non-ϵ-eligible sumset S has either $\text{Vol}_{00}(S) < 1 - \epsilon$ or $\text{Vol}_{11}(S) > \epsilon$ by definition and thus, must be at least $(\epsilon/4)$-far from $\boldsymbol{S}_{\text{no}}$.

Thus with probability at least $1 - o_n(1)$, $\boldsymbol{S}_{\text{no}} \sim \mathcal{S}_{\text{no}}$ is $\epsilon/4 = 0.0125$-far from any sumset. ◀

6 Refuting Sumsets in the Smoothed Analysis Setting

In this section we study the smallest size of a *certificate* that a set is *not* a sumset. Informally, for a set $S \subseteq \mathbb{F}_2^n$ a sumset 0-certificate is a set $D \subseteq \mathbb{F}_2^n$ of points such that querying the oracle $\mathcal{O}_S$ on every point in D suffices to prove that S is not a sumset. More formally:

▶ **Definition 17.** *A set $D \subseteq \mathbb{F}_2^n$ is a* sumset 0-certificate for $S \subseteq \mathbb{F}_2^n$ *if there is no sumset $S' = A + A \subseteq \mathbb{F}_2^n$ for which $S \cap D = S' \cap D$.*

Small 0-certificates are important objects of study for many property testing problems; for example, consider the classic problem of linearity testing. Since a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is linear if and only if $f(x+y) = f(x) + f(y)$ for all $x, y \in \mathbb{F}_2^n$, the property of linearity is characterized by the non-existence of a "linearity 0-certificate" of size three. As is well known, in the seminal work [12] Blum et al. showed that this is a *robust* characterization, in the sense that a simple sampling procedure which queries random triples $x, y, x+y$ and checks whether they constitute a linearity 0-certificate suffices to distinguish linear functions from functions which are far from being linear. A similar framework of sampling 0-certificates is at the heart of many other important property testing results such as low degree testing (see e.g. [5, 28] and many other works) and testing triangle-freeness (see e.g. [4, 2] and many other works). Of course, testing results of this sort rely on, and motivate the discovery of, structural results showing that functions which are far from having the property in question must have "many" "small" 0-certificates.

With this motivation, it is natural to study the size of sumset 0-certificates. Our sumset testing lower bound from Section 5 suggests that there are sets which are far from being sumsets but which do not have "many" "small" sumset 0-certificates. In fact, known results imply that for every non-sumset the smallest 0-certificate is of size $\Omega(2^{n/2}/\sqrt{n})$:

▶ **Lemma 18.** *Let $S \subseteq \mathbb{F}_2^n$ be any non-sumset. Then any sumset 0-certificate D for S must have $|D| \geq \Omega(2^{n/2}/\sqrt{n})$.*

Proof. This is an immediate corollary of a result due to Alon (Section 4 of [3]), which shows that any subset $T \subseteq \mathbb{F}_2^n$ of size $|T| \geq 2^n - \frac{1}{4000}\frac{2^{n/2}}{\sqrt{n}}$ is a sumset. It follows that if $|D| < \frac{1}{4000}\frac{2^{n/2}}{\sqrt{n}}$, then any 0/1 labeling of the points in D is consistent with a sumset (by labeling all points in $\mathbb{F}_2^n \setminus D$ as belonging to the set). ◀

The previous lemma, which establishes that any 0-certificate for sumset testing must have size $\Omega(2^{n/2}/\sqrt{n})$, establishes Part (2) of Theorem 3. In the full version of our paper, we prove a matching upper bound (up to a factor of $\mathrm{poly}(n, 1/\epsilon)$) for any set perturbed by a small amount of random noise, thereby establishing Part (1) of Theorem 3:

▶ **Theorem 19.** *For any set $S \subseteq \mathbb{F}_2^n$ and any $\varepsilon \in (0, \frac{1}{2}]$, there exists a sumset 0-certificate for $\boldsymbol{N}_\varepsilon(S) = S \triangle \boldsymbol{R}_\varepsilon$ of size $2^{n/2} \cdot O(n^{1.5}/\epsilon^{1.5})$, with probability $1 - o_n(1)$ over the random draw of $\boldsymbol{R}_\varepsilon$. Moreover, such a 0-certificate can be found efficiently and non-adaptively (by querying $2^{n/2} \cdot O(n^{1.5}/\epsilon^{1.5})$ points) given oracle access to $\boldsymbol{N}_\varepsilon(S)$.*

References

1 Amir Abboud, Nick Fischer, Ron Safier, and Nathan Wallheimer. Recognizing Sumsets is NP-Complete. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4484–4506, 2025. doi:10.1137/1.9781611978322.153.

2 N. Alon. Testing subgraphs in large graphs. *Random Structures Algorithms*, 21:359–370, 2002. doi:10.1002/RSA.10056.

3 N. Alon. Large sets in finite fields are sumsets. *Journal of Number Theory*, 126(1):110–118, 2007.

4 N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20:451–476, 2000. doi:10.1007/S004930070001.

5 N. Alon, T. Kaufman, M. Krivelevich, S. Litsyn, and D. Ron. Testing low-degree polynomials over GF(2). In *Proc. RANDOM*, pages 188–199, 2003.

6 Noga Alon and Or Zamir. Sumsets in the hypercube. *SIAM Journal on Discrete Mathematics*, 39(1):314–326, 2025. doi:10.1137/24M165569X.

7 Boaz Barak, Russell Impagliazzo, and Avi Wigderson. Extracting randomness using few independent sources. *SIAM Journal on Computing*, 36(4):1095–1118, 2006. doi:10.1137/S0097539705447141.

8 Boaz Barak, Luca Trevisan, and Avi Wigderson. A mini-course on additive combinatorics, 2007. Available at https://www.math.cmu.edu/~af1p/Teaching/AdditiveCombinatorics/allnotes.pdf.

9 Eli Ben-Sasson, Shachar Lovett, and Noga Ron-Zewi. An Additive Combinatorics Approach Relating Rank to Communication Complexity. *J. ACM*, 61(4), July 2014. doi:10.1145/2629598.

10 Khodakhast Bibak. Additive combinatorics: With a view towards computer science and cryptography—an exposition. In Jonathan M. Borwein, Igor Shparlinski, and Wadim Zudilin, editors, *Number Theory and Related Fields*, pages 99–128, New York, NY, 2013. Springer New York. doi:10.1007/978-1-4614-6642-0_4.

11 Eric Blais. Testing juntas nearly optimally. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 2009. doi:10.1145/1536414.1536437.

12 M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47:549–595, 1993. Earlier version in STOC'90. doi:10.1016/0022-0000(93)90044-W.

13 Jean Bourgain. More on the sum-product phenomenon in prime fields and its applications. *Internat. J. Number Theory*, 1(1):1–32, 2005. doi:10.1142/S1793042105000108.

14 Mark Braverman, Subhash Khot, and Dor Minzer. Parallel repetition for the GHZ game: Exponential decay. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1337–1341. IEEE, 2023. doi:10.1109/FOCS57990.2023.00080.

15 Laurent Bulteau, Guillaume Fertin, Romeo Rizzi, and Stéphane Vialette. Some algorithmic results for [2]-sumset covers. *Information Processing Letters*, 115(1):1–5, 2015. doi:10.1016/J.IPL.2014.07.008.

16 Ashok K. Chandra, Merrick L. Furst, and Richard J. Lipton. Multi-party protocols. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 94–99, New York, NY, USA, 1983. Association for Computing Machinery. doi:10.1145/800061.808737.

17 Michael J Collins, David Kempe, Jared Saia, and Maxwell Young. Nonnegative integral subset representations of integer sets. *Information Processing Letters*, 101(3):129–133, 2007. doi:10.1016/J.IPL.2006.08.007.

18 Ernie Croot and Seva Lev. Open problems in additive combinatorics. In *Additive Combinatorics*, volume 43 of *CRM Proceedings and Lecture Notes*, page 207. American Mathematical Society, 2007.

19 Anindya De, Shivam Nadimpalli, and Rocco A. Servedio. Approximating sumset size. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2339–2357. SIAM, 2022. doi:10.1137/1.9781611977073.94.

20 I. Diakonikolas, H. Lee, K. Matulef, K. Onak, R. Rubinfeld, R. Servedio, and A. Wan. Testing for concise representations. In *Proc. 48th Ann. Symposium on Computer Science (FOCS)*, pages 549–558, 2007.

21 Zeev Dvir and Amir Shpilka. An improved analysis of linear mergers. *Comput. Complex.*, 16(1):34–59, May 2007. doi:10.1007/s00037-007-0223-z.

22 Isabelle Fagnot, Guillaume Fertin, and Stéphane Vialette. On finding small 2-generating sets. In *Computing and Combinatorics: 15th Annual International Conference, COCOON 2009 Niagara Falls, NY, USA, July 13-15, 2009 Proceedings 15*, pages 378–387. Springer, 2009. doi:10.1007/978-3-642-02882-3_38.

23 G.A. Freiman. *Foundations of a Structural Theory of Set Addition*. Translations of mathematical monographs. American Mathematical Society, 1973. URL: https://books.google.com/books?id=8zc14FDkWlAC.

24 David Galvin. Independent sets in the discrete hypercube. arXiv preprint 1901.01991, 2019.

25 Chris Godsil and Brendan Rooney. Hardness of computing clique number and chromatic number for cayley graphs. *European Journal of Combinatorics*, 62:147–166, 2017. doi:10.1016/J.EJC.2016.12.005.

26 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. doi:10.1007/s00453-001-0078-7.

27 Ben J. Green. Finite field models in additive combinatorics. In Bridget S. Webb, editor, *Surveys in combinatorics*, pages 1–27. Cambridge Univ. Press, 2005.

28 Charanjit S. Jutla, Anindya C. Patthak, Atri Rudra, and David Zuckerman. Testing low-degree polynomials over prime fields. In *Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 423–432. IEEE Computer Society Press, 2004. doi:10.1109/FOCS.2004.64.

29 Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and boolean isoperimetric-type theorems. *SIAM Journal on Computing*, 47(6):2238–2276, 2018. doi:10.1137/16M1065872.

30 Shachar Lovett. *Additive Combinatorics and its Applications in Theoretical Computer Science*. Number 8 in Graduate Surveys. Theory of Computing Library, 2017. doi:10.4086/toc.gs.2017.008.

31 Anup Rao. An exposition of Bourgain's 2-source extractor. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(34), 2007. ECCC.

32 Alex Samorodnitsky. Low-degree tests at large distances. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, STOC '07, pages 506–515, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1250790.1250864.

33 V. G. Sargsyan. Counting Sumsets and Differences in an Abelian Group. *Journal of Applied and Industrial Mathematics*, 9(2):275–282, 2015.

34 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing*, pages 296–305, 2001. doi:10.1145/380752.380813.

35 Luca Trevisan. Additive combinatorics and theoretical computer science. *ACM SIGACT News*, 40(2):50–66, 2009.

36 Emanuele Viola. *Selected Results in Additive Combinatorics: An Exposition*. Number 3 in Graduate Surveys. Theory of Computing Library, 2011. doi:10.4086/toc.gs.2011.003.

A Proof of Proposition 16

Proof. Let S be any ϵ-eligible sumset, and let A satisfying $A + A = S$ be an additive root of S. We bound the number of ϵ-eligible sumsets by considering possibilities for A.

We begin with the observation that if $\mathrm{Vol}_{00}(A), \mathrm{Vol}_{11}(A) > 0$, then it must be true that $\mathrm{Vol}_{00}(A), \mathrm{Vol}_{11}(A) \leq \epsilon$. Otherwise, we would have $\mathrm{Vol}_{11}(A+A) = \mathrm{Vol}_{11}(S) > \epsilon$, contradicting our assumption that S is ϵ-eligible. Likewise, we have that if $\mathrm{Vol}_{01}(A), \mathrm{Vol}_{10}(A) > 0$, then $\mathrm{Vol}_{01}(A), \mathrm{Vol}_{10}(A) \leq \epsilon$. We split into cases accordingly.

1. All four cosets of $\{x : x_1 = x_2 = 0\}$ are nonempty:
   $\mathrm{Vol}_{00}(A), \mathrm{Vol}_{11}(A), \mathrm{Vol}_{01}(A), \mathrm{Vol}_{10}(A) > 0$.
   In this case, we have that $\mathrm{Vol}_{00}(A), \mathrm{Vol}_{11}(A), \mathrm{Vol}_{01}(A), \mathrm{Vol}_{01}(A) \leq \epsilon$. Using Equation (1), we can then bound the number of possibilities for A (and S) by

   $$\binom{2^n}{\varepsilon 2^n}^4 \leq 2^{4H(\varepsilon)\cdot 2^n} \cdot 2^{O(n)}.$$

2. Either $\mathrm{Vol}_{00}(A)$ and $\mathrm{Vol}_{11}(A) > 0$, or $\mathrm{Vol}_{01}(A)$ and $\mathrm{Vol}_{10}(A) > 0$, but not both.
   Here the volume of A on two of the four cosets is at most ϵ, in one other coset it is 0, and in the final coset it may be as large as 1. In this case, again using Equation (1), the number of possibilities for A (and S) is bounded by

   $$2^{2^n} \cdot \binom{2^n}{\varepsilon 2^n}^2 \leq 2^{(1+2H(\varepsilon))\cdot 2^n} \cdot 2^{O(n)}.$$

3. Either $\mathrm{Vol}_{00}(A)$ or $\mathrm{Vol}_{11}(A) = 0$, and either $\mathrm{Vol}_{01}(A)$ or $\mathrm{Vol}_{10}(A) = 0$, hence at least two of the four cosets contain no points in A.
   Assume without loss of generality that $\mathrm{Vol}_{11}(A) = \mathrm{Vol}_{01}(A) = 0$. This immediately implies that $\mathrm{Vol}_{01}(S) = \mathrm{Vol}_{11}(S) = 0$, as $A + A$ cannot contain points in either coset. (Note that, whichever pair of cosets we choose to zero out, this implies that $\mathrm{Vol}_{11}(S) = 0$ and either that $\mathrm{Vol}_{01}(S) = 0$ or $\mathrm{Vol}_{10}(S) = 0$.) Using the fact that $\mathrm{Vol}_{00}(S) \geq 1 - \epsilon$, the number of possibilities for S is bounded by

   $$2^{2^n} \cdot \binom{2^n}{\varepsilon 2^n} \leq 2^{(1+H(\varepsilon))\cdot 2^n} \cdot 2^{O(n)}.$$

Summing the number of ϵ-eligible sumsets covered by each case completes the proof. ◀

Linear Layouts Revisited: Stacks, Queues, and Exact Algorithms

Thomas Depian
Algorithms and Complexity Group, TU Wien, Austria

Simon D. Fink
Algorithms and Complexity Group, TU Wien, Austria

Robert Ganian
Algorithms and Complexity Group, TU Wien, Austria

Vaishali Surianarayanan
University of California at Santa Barbara, CA, USA

Abstract

In spite of the extensive study of stack and queue layouts, many fundamental questions remain open concerning the complexity-theoretic frontiers for computing stack and queue layouts. A stack (resp. queue) layout places vertices along a line and assigns edges to pages so that no two edges on the same page are crossing (resp. nested). We provide three new algorithms which together substantially expand our understanding of these problems:

1. A fixed-parameter algorithm for computing minimum-page stack and queue layouts w.r.t. the vertex integrity of an n-vertex graph G. This result is motivated by an open question in the literature and generalizes the previous algorithms parameterizing by the vertex cover number of G. The proof relies on a newly developed Ramsey pruning technique. Vertex integrity intuitively measures the vertex deletion distance to a subgraph with only small connected components.
2. An $n^{\mathcal{O}(q\ell)}$ algorithm for computing ℓ-page stack and queue layouts of page width at most q. This is the first algorithm avoiding a double-exponential dependency on the parameters. The page width of a layout measures the maximum number of edges one needs to cross on any page to reach the outer face.
3. A $2^{\mathcal{O}(n)}$ algorithm for computing 1-page queue layouts. This improves upon the previously fastest $n^{\mathcal{O}(n)}$ algorithm and can be seen as a counterpart to the recent subexponential algorithm for computing 2-page stack layouts [ICALP'24], but relies on an entirely different technique.

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

Keywords and phrases stack layouts, queue layouts, parameterized algorithms, vertex integrity, Ramsey theory

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.15

Related Version *Full Version*: `https://arxiv.org/abs/2508.16319` [14]

Funding *Thomas Depian*: Project No. 10.47379/ICT22029 of the Vienna Science Foundation (WWTF).
Simon D. Fink: Project No. 10.47379/ICT22029 of the Vienna Science Foundation (WWTF) and Project No. 10.55776/Y1329 of the Austrian Science Fund (FWF).
Robert Ganian: Project No. 10.47379/ICT22029 of the Vienna Science Foundation (WWTF) and Projects No. 10.55776/Y1329, 10.55776/COE12 of the Austrian Science Fund (FWF).

1 Introduction

A linear layout $\langle \prec, \sigma \rangle$ of a graph G is a total ordering $\prec$ of its vertices and a partitioning σ of its edges into k *pages* such that each page satisfies certain conditions. The two by far most commonly studied types of linear layouts are *stack layouts* and *queue layouts*; in the


© Thomas Depian, Simon D. Fink, Robert Ganian, and Vaishali Surianarayanan;
licensed under Creative Commons License CC-BY 4.0



former, we require that no four vertices $a \prec b \prec c \prec d$ have the edges ac and bd placed on the same page, while for the latter we forbid any page from containing the edges ad and bc. Intuitively, this corresponds to forbidding edges crossing and edges nesting (in a rainbow pattern), respectively – see Figure 1.

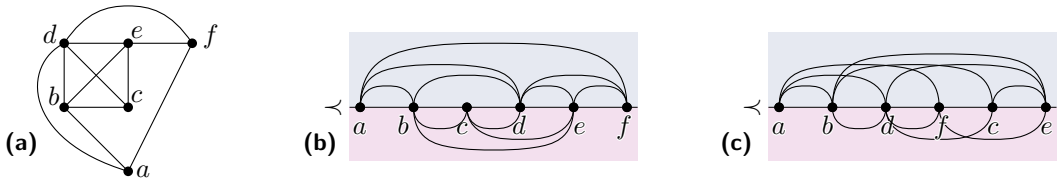


Figure 1 A graph G **(a)** together with a 2-page stack layout **(b)** and 2-page queue layout **(c)**. The two pages are colored blue and lilac, respectively.

Originally motivated from applications in VLSI [10, 41] and bioinformatics [31], stack and queue layouts have become the focus of extensive research. While there are many classical works studying the structural properties of the two notions [17–21, 32, 34, 46], several more recent studies have targeted exact and parameterized algorithms for computing stack and queue layouts that minimize the number of used pages. These results are inherently aimed at circumventing the long-known intractability of these problems: the NP-hardness of determining whether an input graph admits a 2-page stack layout or a 1-page queue layout has been established over thirty years ago [34, 46].

To that end, recent works have pursued the use of structural graph parameters to establish tractability on graphs which are "well-structured". Computing ℓ-page stack layouts is now known to be fixed-parameter tractable w.r.t. the vertex cover number [7] or, alternatively, the feedback edge number of the input graph G [26]. Computing ℓ-page queue layouts is likewise known to admit a fixed-parameter algorithm w.r.t. the vertex cover number of G [8]. One drawback of these results is that they only yield tractability under highly restrictive graph parameters, in the sense of achieving low values only on very "simple" graphs. While computing 1-page queue layouts is also known to be fixed-parameter tractable when parameterized by the treedepth of G [8], for general ℓ it was repeatedly posed as an open question whether the aforementioned vertex-cover based algorithms can be lifted to less restrictive structural graph parameters [7, 8, 21, 25, 26]. Another recent direction is the establishment of tighter algorithmic upper bounds for special cases: while ℓ-page stack and queue layouts can both be computed in time $n^{\mathcal{O}(n)}$ via trivial brute-force algorithms, 2-page stack layouts are now known to admit a subexponential algorithm [26].

Contributions. As our **first contribution**, we lift the aforementioned fixed-parameter tractability of computing linear layouts to a less restrictive structural graph parameter:

▶ **Theorem 1.** *Computing a minimum-page stack layout and minimum-page queue layout is fixed-parameter tractable w.r.t. the* vertex integrity *of the input graph.*

The vertex integrity $\mathsf{vi}(G)$ of a graph G measures, roughly speaking, how many vertex deletions are required to decompose G into small connected components, and has been used in the design of parameterized algorithms for a variety of challenging problems [9, 22–24, 27, 29, 30, 39].[1] More precisely, $\mathsf{vi}(G)$ is the minimum integer k satisfying the following:

Due to space constraints, we defer full proofs of statements marked with ★ to the full version [14].

[1] Some prior works use the *fracture number*, which is parametrically equivalent to vertex integrity.

there exists a vertex set $X \subseteq V(G)$ such that for each connected component C of $G - X$, $|V(C) \cup X| \leq k$. As vertex integrity is upper-bounded by the vertex cover number plus one, Theorem 1 generalizes the fixed-parameter tractable algorithms for computing stack and queue layouts w.r.t. the vertex cover number by Bhore, Ganian, Montecchiani, and Nöllenburg [7, 8]. Moreover, since the vertex integrity is sandwiched between the vertex cover number and treedepth, our result can be seen as a stepping stone towards solving the problem on more general parameterizations.

To establish Theorem 1, we introduce a novel proof technique we call *Ramsey pruning*. On a high level, the algorithm underlying the computation is very simple: assuming $|G|$ is larger than some pre-designated function k, it recursively identifies a connected component C of $G - X$ such that $G - X$ contains sufficiently many copies of C, and deletes C from the instance. The difficulty lies in proving that this operation is safe, i.e., that $G - C$ does not admit a layout with fewer pages than G. The "usual" approach for such a proof would be to show that C can be reinserted into any hypothetical linear layout $\langle \prec, \sigma \rangle$ of $G - C$ without increasing the number of pages; however, this not only seems excruciatingly difficult to prove, but we also believe it to be false for certain choices for $\langle \prec, \sigma \rangle$. Ramsey pruning avoids this issue by using Ramsey-type arguments to argue that $\langle \prec, \sigma \rangle$ must contain a *guiding sub-layout* $\langle \prec', \sigma' \rangle$ – a linear layout of some carefully selected subgraph of $G - C$ – with certain well-defined properties. We then use the well-formedness of $\langle \prec', \sigma' \rangle$ to build a brand new linear layout $\langle \prec^*, \sigma^* \rangle$ of G, thus establishing that G and $G - C$ indeed require the same number of pages. We believe this new technique to be generic and applicable to other problems under the same parameterization, as suggested by the fact that unlike the previous vertex-cover based algorithms our approach works for both stack and queue layouts with almost no problem-specific changes required.

While Theorem 1 pushes the frontiers of tractability for our two problems of interest in a complexity-theoretic sense, our use of Ramsey-type arguments means that the obtained running time bound has an entirely impractical dependency on the parameter (see Lemma 5). In fact, up to now none of the known parameterized algorithms for computing ℓ-page stack or queue layouts have a runtime parameter dependency that would be better than double-exponential – the parameterized algorithms w.r.t. the vertex cover [7, 8] and feedback edge numbers [26] all preprocess the graph to obtain an equivalent instance (a *kernel*) whose size is exponential in the parameter, and then solve that equivalent bounded-size instance via brute force. As our **second contribution**, we provide the first single-exponential algorithms (in the parameters) capable of solving these problems for arbitrary fixed choices of ℓ:

▶ **Theorem 2 (★).** *Given integers ℓ and q along with an n-vertex graph G, we can compute an ℓ-page stack or queue layout with page width at most q of G (if one exists) in time $n^{\mathcal{O}(q \cdot \ell)}$.*

Here, the page width measures the maximum number of edges one needs to cross to reach any vertex from the outer face (see also Section 2). The page width of linear layouts[2] has been studied in several early works [10, 32, 43, 44] and it is generally desirable to obtain layouts not only using few pages, but also with low page width – in fact, the original papers introducing stack layouts explicitly targeted algorithms optimizing both measures. However, the algorithmic applications of page width as a parameter have only been investigated in a recent paper on extending incomplete linear layouts [12]. We remark that while Theorem 2 merely provides a so-called XP *algorithm* in a complexity-theoretic sense, it may still be more efficient in practice than known fixed-parameter algorithms for the problem – especially if the aim is to compute layouts with low pagewidth.

[2] The term *cutwidth* has been used interchangeably with page width in the past [10]; here, we use the latter in order to disambiguate from the related graph parameter cutwidth.

A natural approach towards establishing Theorem 2 would be to use dynamic programming in order to construct the sought-after linear layout in a "left-to-right" fashion; however, the issue is that doing this directly would require us to store roughly 2^n possible subsets of previously processed vertices. To circumvent this, we obtain new insights into the decomposability of layouts with bounded page width, allowing us to construct an $n^{\mathcal{O}(q \cdot \ell)}$-size auxiliary *state graph* H, where we show that the dynamic computation of a sought-after layout can be represented as an easily computable path in H.

As our final **third contribution**, we take a step back from the multivariate analysis of these problems and recall that the trivial $n^{\mathcal{O}(n)}$ barrier was only recently overcome for computing 2-page stack layouts [26], i.e., the lowest-page stack layouts giving rise to an NP-complete problem. Here, we show that the trivial $n^{\mathcal{O}(n)}$ barrier can also be overcome when aiming for the lowest-page queue layouts which still give rise to NP-completeness:

▶ **Theorem 3 (★).** *Given an n-vertex graph G, we can compute a 1-page queue layout of G (if one exists) in time $2^{\mathcal{O}(n)}$.*

We remark that the running time bound provided by Theorem 3 is worse than the $2^{\mathcal{O}(\sqrt{n})}$ bound previously obtained for 2-page stack layouts [26]. The reason for this is that the latter result relied on the equivalence of 2-page stack layouts with the existence of *subhamiltonian paths*; the authors of that previous work then essentially obtained a single-exponential fixed-parameter algorithm for finding such paths w.r.t. the graph parameter treewidth, improving on the previously established fixed-parameter tractability of the problem [4].

However, emulating the same approach seems difficult in the queue layout setting: 1-page queue layouts also have an equivalent formulation in terms of so-called *arched-level planar* drawings [34], but it is not at all clear whether computing such drawings is fixed-parameter tractable w.r.t. treewidth (not to mention the fact that one would need a single-exponential algorithm). Instead, our proof of Theorem 3 relies on a Turing reduction from computing arched-level planar drawings to $2^{\mathcal{O}(n)}$ many instances of a well-studied problem called Level Planarity, which is known to be solvable in polynomial time [33, 37].

Related Work. We refer to the dedicated survey for an overview of many earlier structural results concerning linear layouts [19]. Key structural results in the area include the existence of 4-page stack layouts [46] and 42-page queue layouts [5,17] for all planar graphs. Researchers have also studied the notion of *mixed layouts*, which are linear layouts with some pages behaving like stack and some like queue layouts [3, 38]. We note that due to the techniques used in their proofs, it seems very likely that Theorems 1 and 2 could be adapted to the mixed layout setting with minimum changes required. Finally, we remark that parameterized algorithms for stack and/or queue layouts have additionally been considered in the extension setting (where the task is to complete a provided partial layout) [12, 13], in the upward-planarity setting (where the graph is directed and edges must be oriented in a left-to-right fashion along the layout) [6] and when the vertex ordering in the layout is fixed [1, 40].

2 Preliminaries

For an integer $\ell \geq 1$ we let $[\ell]$ denote the set $\{1, 2, \ldots, \ell\}$. We assume the reader to be familiar with standard graph terminology [15]. Without loss of generality, we assume all input graphs to be connected, have n vertices and m edges. For a set of vertices $V' \in V(G)$, we let $G[V']$ denote the graph induced on V'. Furthermore, for a set of edges $E' \in E(G)$, we let $V(E')$ denote the set of its endpoints and $G[E'] = (V(E'), E')$. A *cut* (A, B) of G is a partition of

the vertices in $A \subseteq V(G)$ and $B = V(G) \setminus A$. We call $F = \{uv \in E(G) \mid u \in A, v \in B\}$ a *cut-set* of *size* $|F|$ that *induces* the cut (A, B). Throughout the paper, we omit an explicit reference to G if it is clear from the context, e.g., we write V and E instead of $V(G)$ and $E(G)$.

Given a linear order $\prec$ of a graph G and two vertices $u, v \in V$, we say that u is *left* of v if $u \prec v$ and *right* of v if $v \prec u$. The vertices u and v are *consecutive on the spine* if they occur consecutively in $\prec$, i.e., there is no vertex $w \in V$ such that $u \prec w \prec v$ or $v \prec w \prec u$ holds. We address with $\text{Left}_v^{\prec} := \{u \in V \mid u \prec v\}$ all vertices $u \in V$ left of v with respect to $\prec$ and define $\text{Right}_v^{\prec} := V \setminus (\text{Left}_v^{\prec} \cup \{v\})$. Finally, we let $X_v^{\prec} := \{uw \in E \mid u \in \text{Left}_v \cup \{v\}, w \in \text{Right}_v\}$ denote the set of edges that *span* the spine between v and its right neighbor, i.e., all edges with one endpoint left of (or at) v and one right of v.

We assume familiarity with the basic foundations of parameterized complexity theory [11]. All algorithms obtained in this work are exact, deterministic, constructive and rely exclusively on computable functions. To express some of our bounds, we will occasionally use the Knuth notation $\uparrow\uparrow$ where for an integer z, $2 \uparrow\uparrow z$ represents an exponential tower of 2's of height z.

Linear Layouts. Let u_1v_1 and u_2v_2 be two edges of G. We say that they *cross* under a linear order $\prec$ if $u_1 \prec u_2 \prec v_1 \prec v_2$ holds, and they *nest* if $u_1 \prec u_2 \prec v_2 \prec v_1$ holds. For an integer $\ell \geq 1$, let $\sigma_G \colon E(G) \to [\ell]$ be a function that assigns each edge to a *page* $p \in [\ell]$. The *linear layout* $\langle \prec_G, \sigma_G \rangle$ is a *stack* (*queue*) *layout* if no two edges $e_1, e_2 \in E$ with $\sigma_G(e_1) = \sigma_G(e_2)$ cross (nest). We call $\prec_G$ the *spine order* and σ_G the *page assignment*. For the remainder of the paper, we write $\prec$ and σ if the graph G is clear from context.

The *page width* $\omega(\langle \prec, \sigma \rangle)$ of an ℓ-page linear layout $\langle \prec, \sigma \rangle$ corresponds to the maximum number of edges on a single page that span the spine between a vertex and its right neighbor, i.e., $\omega(\langle \prec, \sigma \rangle) := \max_{p \in [\ell]} \max_{v \in V(G)} \left| X_v \cap \sigma^{-1}(p) \right|$. We call an ℓ-page linear layout with page width q an *ℓ-page q-width linear layout*, or simply a *solution* when the ℓ and q are clear from context. In line with the literature, we assume no explicit bound on the page width when q is not specified.

Vertex Integrity. A graph G has *vertex integrity* $\mathsf{vi}(G) = p$ if p is the smallest integer with the following property: G contains a vertex set S such that for each connected component H of $G - S$, $|V(H) \cup S| \leq p$. One may observe that the vertex integrity is upper-bounded by the size of a minimum vertex cover in the graph (i.e., the vertex cover number) plus one. The vertex integrity of an n-vertex graph along with a corresponding partition into S and $\mathcal{C} = G - S$ can be computed in time $\mathcal{O}(p^{p+1} \cdot n)$ [16].

3 A Fixed-Parameter Algorithm Parameterized by Vertex Integrity

In this section, we obtain a fixed-parameter algorithm that takes as input a positive integer ℓ and a graph G, is parameterized by $\ell + \mathsf{vi}(G)$ and computes an ℓ-page stack and/or queue layout of G (or both), if such layouts exist. We begin by noting that using well-known relationships between vertex integrity, minimum-page stack and queue layouts and the graph parameter *treewidth* (tw), we can assume ℓ to be upper-bounded by a function of $\mathsf{vi}(G)$ thanks to the well-known relation $\mathsf{tw}(G) \leq \mathsf{vi}(G)$:

▶ **Proposition 4** ([20, 45]). *The number of pages in a minimum stack and queue layout of a graph G is upper-bounded by $\mathsf{tw}(G) + 1 \leq \mathsf{vi}(G) + 1$ and $2^{\mathsf{tw}(G)} + 1 \leq 2^{\mathsf{vi}(G)} + 1$, respectively.*

The algorithm operates by obtaining a problem kernel, as formalized below; recall $p = \mathsf{vi}(G)$.

▶ **Lemma 5.** *There is an $\mathcal{O}(p^{p+1} \cdot n)$ time algorithm that takes a graph G with an integer ℓ and outputs a subgraph G' of G (a* kernel*) of size at most $\left(2 \uparrow\uparrow (p \cdot 2^{2p^2}) \cdot 2 + 6\right)^{\ell \cdot p}$ with the following property: G' admits an ℓ-page stack (or queue) layout $\langle \prec_{G'}, \sigma_{G'} \rangle$ if and only if so does G. Moreover, such a layout for G can be computed from $\langle \prec_{G'}, \sigma_{G'} \rangle$ in polynomial time.*

Lemma 5 directly implies Theorem 1, and the rest of this section is dedicated to proving this lemma. For the following, let us fix a graph G, positive integers ℓ and p, and a choice of $S \subseteq V(G)$ witnessing $\mathsf{vi}(G) = p$ as in Section 2. Further let $\mathcal{C}$ be the set of connected components of $G - S$. First, we define a notion of "component-types" which groups components in $\mathcal{C}$ that exhibit the same outside connections and internal structure.

▶ **Definition 6.** *We say two graphs $H_0, H_1 \in \mathcal{C}$ are* twins*, denoted $H_0 \sim H_1$, if there exists a canonical isomorphism α from H_0 to H_1 such that for each vertex $u \in V(H_0)$ and each $v \in S$, $uv \in E(G)$ if and only if $\alpha(u)v \in E(G)$.*

▶ **Lemma 7 (★).** *Each graph $H \in \mathcal{C}$ has at most p vertices, $\sim$ is an equivalence relation and the number of equivalence classes in $[\sim]$ is upper-bounded by $p \cdot 2^{2p^2}$. Moreover, a partition of connected components into $[\sim]$ can be computed in time at most $\mathcal{O}(p \cdot 2^{2p^2} \cdot n)$.*

We now introduce the notion of large equivalence classes based on their size. We then use this to define what we call a large group of vertices – one that contains exactly one representative from each large equivalence class. Our kernel will keep a bounded number of these large groups.

▶ **Definition 8.** *Let k be a positive integer, an equivalence class $[H]$ of $\sim$ is said to be k-*large *if $|[H]| \geq k$. Further a vertex set $L \subseteq V(G)$ is called a k-*large group *if the induced subgraph $G[L]$ is a disjoint union of exactly one graph from each k-large equivalence class of $\sim$.*

Next we define a special induced subgraph that will serve as our kernel. The definition is based on carefully choosing a k that is bounded by a computable function of p and ℓ so that keeping only k many k-large groups along with the small parts of the graph suffices to capture the necessary structure. Towards this, let us first fix $f(\ell, p, x) := 2^{2^{\ell \cdot x^2 \cdot 2^{12p^2}}}$ to be a computable function that is large enough to apply our Ramsey-type arguments later on; here, x will be an integer that represents the size of a deletion set (initially S, but this will be updated iteratively in the proof of Lemma 11). Moreover, let $g(\ell, p)$ be a computable function of ℓ and p that will upper-bound our nested application of the function f in that same proof; to provide a concrete bound, we set $g(\ell, p) := \left(2 \uparrow\uparrow (p \cdot 2^{2p^2}) \cdot 2 + 4\right)^{\ell \cdot p}$.

▶ **Definition 9.** *An induced subgraph G' of G is said to be a* reduced graph *of G if there exists a positive integer $x \leq g(\ell, p)$ and a partition of $V(G') = S \uplus S' \uplus Y$ satisfying:*

1. *$|S \cup S'| = x$ and S' is the set of all vertices in graphs in $\mathcal{C}$ that do not belong to an $f(\ell, p, x)$-large equivalence class of $\sim$.*
2. *If there are no $f(\ell, p, x)$-large equivalence classes of $\sim$, $Y = \emptyset$ and $V(G') = S \uplus S' = V(G)$. Otherwise, it holds that $Y = L_1 \uplus \cdots \uplus L_{f(\ell,p,x)}$ where L_i is an $f(\ell, p, x)$-large group for each $i \in \{1, \cdots, f(\ell, p, x)\}$, and each pair of L_i and L_j, $i \neq j$, is vertex-disjoint.*

Intuitively, the reduced graphs we will be dealing with will consist of S, a set S' of all equivalence classes of $\sim$ which are too small to fully saturate our large groups (these will later be treated essentially in the same way as S), and a sufficient number of large groups; equivalence classes of size larger than $f(\ell, p, x)$ are "pruned" to have size exactly $f(\ell, p, x)$. A schematic overview of this intuition can be found in Figures 2b and c later on.

The core of our result is the following lemma, which we prove separately in Section 3.1.

▶ **Lemma 10.** *If G' is a reduced graph of G then G has an ℓ-page stack (queue) layout if and only if G' has an ℓ-page stack (queue) layout.*

Before proceeding to that proof, we show how to construct a reduced graph having size bounded by a computable function of p and ℓ in polynomial time.

▶ **Lemma 11 (★).** *There exists a reduced graph G' of G, and given S and $\sim$ such a graph can be computed in polynomial time. Further, the number of vertices in G' is upper-bounded by $\left(2 \uparrow\uparrow (p \cdot 2^{2p^2}) \cdot 2 + 6\right)^{\ell \cdot p}$.*

Lemma 11 combined with Lemma 10 establishes Lemma 5 by constructing a reduced graph which is a kernel of the desired size. Thus, what remains is to establish Lemma 10; this is where the core ideas of Ramsey pruning will come into play.

3.1 Proof of Lemma 10

For this proof, we fix G' to be a reduced graph of G. Since G' is an induced subgraph of G, it is clear that if G has an ℓ-page stack (queue) layout then G' also has an ℓ-page stack (queue) layout. To complete the proof we show that the reverse is true: If G' has an ℓ-page stack (queue) layout, then G also has an ℓ-page stack (queue) layout.

Recall that $f(\ell, p, x) := 2^{2^{\ell \cdot x^2} \cdot 2^{12p^2}}$. Let $\langle \prec, \sigma \rangle$ be a fixed (but hypothetical) ℓ-page stack (queue) layout of G'; throughout this section, for $H \subseteq G'$ we will use $\langle \prec_H, \sigma_H \rangle$ to refer to the sublayout of $\langle \prec, \sigma \rangle$ induced on the subgraph H of G'. If $V(G') = V(G)$ we are done. So let $V(G') = S \uplus S' \uplus L_1 \uplus \cdots \uplus L_{f(\ell,p,|S \cup S'|)}$ be a partition witnessing that G' is a reduced graph. Here each L_i is a $f(\ell, p, |S \cup S'|)$-large group and S' is the set of all vertices in graphs that do not belong to an $f(\ell, p, |S \cup S'|)$-large equivalence class in $\sim$. For brevity, we will hereinafter use *large* as shorthand for $f(\ell, p, |S \cup S'|)$-large.

Let $\mathcal{L} = \{L_1, \cdots, L_{f(\ell,p,|S \cup S'|)}\}$. Our key idea is to identify three large groups $X, Y, Z \in \mathcal{L}$ with a special structure (a "guiding sublayout") in the solution $\langle \prec, \sigma \rangle$ using Ramsey theory. We will then use this pattern to insert the remaining parts of the graph. Before proceeding, we define some notations to be able to identify these groups, starting with a "template" R.

▶ **Definition 12.** *Let k be the number of large equivalence classes in $\sim$ and let $R_i \in \mathcal{C}$ be a canonical representative of the i^{th} large equivalence class $[R_i]$. Let $R := V(R_1) \uplus \cdots \uplus V(R_k)$.*

Recall that by Definition 8, if X is a large group then $G[X]$ is a disjoint union of exactly one graph from each large equivalence class of $\sim$. We now define a natural isomorphism from $G[X]$ to $G[R]$. This will allow us to map consistently between different large groups via R.

▶ **Definition 13.** *For a large group X with $G[X] = H_1 \uplus \cdots \uplus H_k$, $H_i \in [R_i]$ for each $i \in [k]$, let α_X be an isomorphism from $G[X]$ to $G[R] = R_1 \uplus \cdots \uplus R_k$ such that for each $i \in [k]$ and vertex $u \in H_i$, $\alpha_X(u) \in V(R_i)$, and for each $v \in S$, $uv \in E(G)$ if and only if $\alpha_X(u)v \in E(G)$.*

For any large group A and for each $u \in R$ we will sometimes use u_A as shorthand for $\alpha_A^{-1}(u)$. From now we fix two distinct large groups L and L' in $\mathcal{L}$. Further, for distinct large groups $X, Y \in \mathcal{L}$, we say $X \prec Y$ if in $\prec$, the first vertex in $X \cup Y$ is from X.

▶ **Definition 14.** *For $X \prec Y \in \mathcal{L} \setminus \{L, L'\}$, we define $\phi_{X,Y} : S \cup S' \cup X \cup Y \to S \cup S' \cup L \cup L'$*

- *$\phi_{X,Y}(s) = s$ for each $s \in S \cup S'$*
- *$\phi_{X,Y}(x) = \alpha_L^{-1}(\alpha_X(x))$ for each $x \in X$*
- *$\phi_{X,Y}(y) = \alpha_{L'}^{-1}(\alpha_Y(y))$ for each $y \in Y$*

Note that $\phi_{X,Y}$ is an isomorphism from $G[S \cup S' \cup X \cup Y]$ to $G[S \cup S' \cup L \cup L']$.

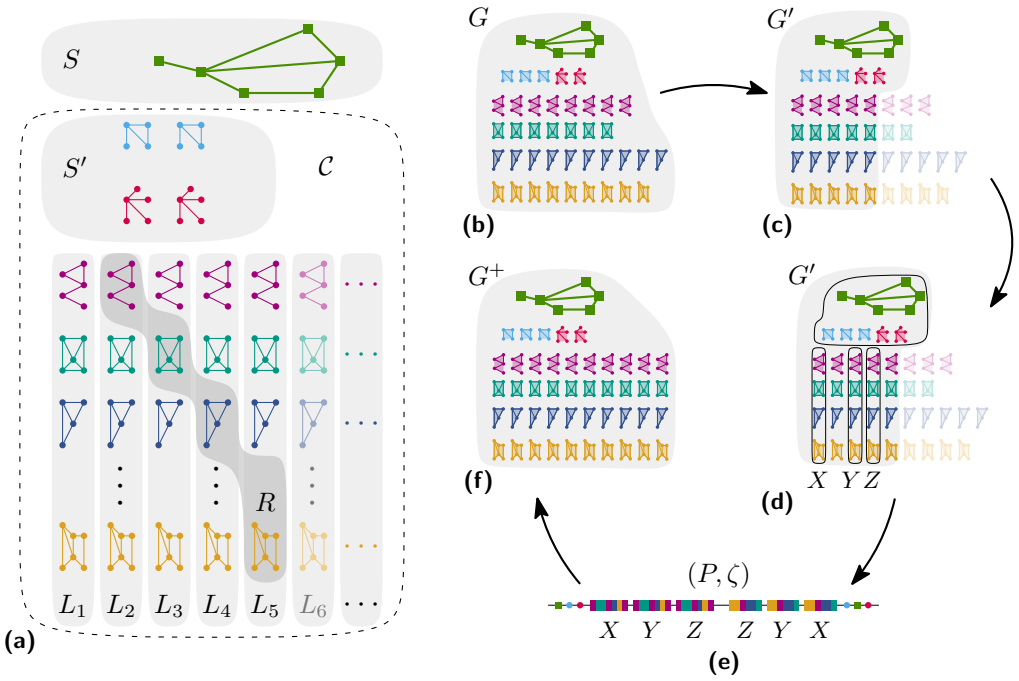


Figure 2 A schematized overview of **(a)** the used notation and **(b)**–**(f)** proof of Theorem 1. In the latter, G is the original input graph, G' the reduced graph obtained after applying Lemma 11, and $S \cup S' \cup X \cup Y \cup Z$ form the guiding sublayout that is used to construct an ℓ-page layout for a graph G^+ which is a supergraph of G – hence establishing that G' admits an ℓ-page stack (or queue) layout if and only if so does G.

▶ **Definition 15.** *For $X \prec Y \in \mathcal{L} \setminus \{L, L'\}$, $\mathit{info}_{\langle \prec, \sigma \rangle}(X, Y)$ is the stack (queue) layout of $G[S \cup S' \cup L \cup L']$ obtained from $\langle \prec_{G[S \cup S' \cup X \cup Y]}, \sigma_{G[S \cup S' \cup X \cup Y]} \rangle$ using the isomorphism $\phi_{X,Y}$.*

We are now ready to show the existence of three distinct large groups X, Y, Z with a useful consistent pattern between them and $S \cup S'$ based on info. Essentially, these three groups will form the aforementioned guiding sublayout used to argue the correctness of the pruning step, i.e., establish Lemma 10 by showing that we can reinsert all the removed vertices by building on $\langle \prec_{G[S \cup S' \cup X \cup Y \cup Z]}, \sigma_{G[S \cup S' \cup X \cup Y \cup Z]} \rangle$. Note that – perhaps counterintuitively – we will do so by discarding all other information about $\langle \prec, \sigma \rangle$. We refer to Figure 2 for an overview of the entire approach and to Figure 4 for a visualization of info.

▶ **Lemma 16.** *There exists distinct large groups $X, Y, Z \in \mathcal{L} \setminus \{L, L'\}$ with $X \prec Y \prec Z$ such that $\mathit{info}_{\langle \prec, \sigma \rangle}(X, Y) = \mathit{info}_{\langle \prec, \sigma \rangle}(Y, Z) = \mathit{info}_{\langle \prec, \sigma \rangle}(X, Z)$.*

Proof. Construct a complete edge colored auxiliary graph $\mathcal{H}$ with $V(\mathcal{H}) = \mathcal{L} \setminus \{L, L'\}$. For $X, Y \in \mathcal{L} \setminus \{L, L'\}$ with $X \prec Y$, we set $\mathit{color}((X, Y)) = \mathsf{info}_{\langle \prec, \sigma \rangle}(X, Y)$. Let $x := |S \cup S'|$.

▷ Claim 17 (★). The number of possible distinct edge colors of $\mathcal{H}$ is at most $2^{\ell \cdot x^2 \cdot 2^{9p^2}}$.

We now use a well-known fact (from Ramsey theory [2]) that any edge-colored clique on n vertices colored with t colors has a monochromatic clique of size at least $\log_t(n)/t$.

We have $|V(\mathcal{H})| = |\mathcal{L}| - 2 = f(\ell, p, x) - 2 \geq 2^{2^{\ell \cdot x^2 \cdot 2^{11p^2}}}$. Thus $n \geq 2^{2^{\ell \cdot x^2 \cdot 2^{11p^2}}}$ and $t \leq 2^{\ell \cdot x^2 \cdot 2^{9p^2}}$ and therefore there is a monochromatic clique of size at least $\log_t(n)/t \geq 2^{2^2} \geq 3$. A monochromatic clique of size 3 in $\mathcal{H}$ corresponds to distinct large groups $X, Y, Z \in \mathcal{L} \setminus \{L, L'\}$ with $X \prec Y \prec Z$ such that $\mathsf{info}_{\langle \prec, \sigma \rangle}(X, Y) = \mathsf{info}_{\langle \prec, \sigma \rangle}(Y, Z) = \mathsf{info}_{\langle \prec, \sigma \rangle}(X, Z)$. ◀

Let $X, Y, Z \in \mathcal{L}$ be three large groups witnessing the previous lemma. We now show that we can partition the vertices of X, Y, and Z in a nice way that will guide us on how to insert the remaining parts of the graph. Towards this, we first note that the $\mathsf{info}_{\langle \prec, \sigma \rangle}$ forces (1) edges incident to X, Y and Z to have the same page assignment, and moreover (2) forces the individual vertex orderings of $X \cup S \cup S'$, $Y \cup S \cup S'$, $Z \cup S \cup S'$ to be exactly the same as some ordering $\prec'$ of $R \cup S \cup S'$; see also the full version [14, Observations 18 and 19]. However, this does not yet fix how the vertices of X, Y and Z are ordered with respect to each other.

For the following, let us set $\Upsilon = S \cup S' \cup X \cup Y \cup Z$. Let a *block* be a consecutive subsequence of vertices from R w.r.t. $\prec'$, and a *solution-block* of $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$ be a consecutive subsequence of vertices from Υ; see also Figure 3.

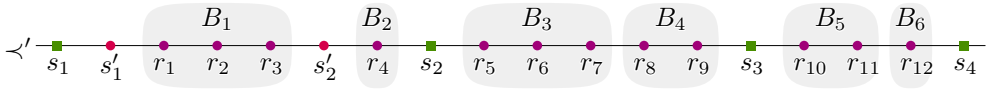


Figure 3 An ordering $\prec'$ of $R \cup S \cup S'$ and a partition of R into the blocks B_1 to B_6, indicated in gray. Note that vertices of $S \cup S'$, colored green and red, respectively, always separate two blocks. However, not every pair of blocks is separated by such a vertex, see, for example, blocks B_3 and B_4.

We are now ready to prove the structural result which provides strong guardrails on $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$; in particular, it guarantees that all vertices from X, Y and Z must occur in consecutive solution-blocks and following a strict "ascending" or "descending" order. We remark that this structural result is only made possible by the extraction of three copies of large groups via Lemma 16; see also Figure 4 for an illustration.

▶ **Lemma 18 (★).** *There exists a pair (P, ζ), where P is a partitioning of R into blocks and $\zeta \colon P \to \{\nearrow, \searrow\}$ satisfying the following. For every block $B \in P$ with $\zeta(B) = \nearrow$, the sequence $\alpha_X^{-1}(B)\alpha_Y^{-1}(B)\alpha_Z^{-1}(B)$ is a solution-block of $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$. Moreover, for every block $B \in P$ with $\zeta(B) = \searrow$, the sequence $\alpha_Z^{-1}(B)\alpha_Y^{-1}(B)\alpha_X^{-1}(B)$ is a solution-block of $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$.*

Crucially, we can now use Lemma 18 to safely insert an arbitrary number of large groups into $G[\Upsilon]$ without increasing the number of required pages.

▶ **Lemma 19 (★).** *Let G^+ be the supergraph of G with each large equivalence class having equal size, then G^+ has an ℓ-page stack (queue) layout.*

Proof sketch. Let $t \geq 3$ be the maximum size of a large equivalence class of $\sim$. Note that G^+ can be expressed as $G^+ = G^+[S \cup S' \cup X_1 \cup \cdots \cup X_t]$ where each $X_i, i \in [t]$ is a large group. We construct a stack (or queue, as desired) layout $\langle \prec^+, \sigma^+ \rangle$ of G^+ as follows.

Construction of σ^+. For each edge $(u, v) \in G^+$,

- If $u, v \in S \cup S'$, we set $\sigma^+(u, v) := \sigma(u, v)$.
- If $u \in X_i$ for some $i \in [t]$ and $v \in S$, we set $\sigma^+(u, v) := \sigma(r_Y, v)$ where $r = \alpha_{X_i}(u)$.
- If $u, v \in X_i$ for some $i \in [t]$, we set $\sigma^+(u, v) := \sigma(r_Y, r'_Y)$ where $r = \alpha_{X_i}(u)$, $r' = \alpha_{X_i}(v)$.

Construction of $\prec^+$. In $\prec'$, replace each block $B \in P$ with $\alpha_{X_1}^{-1}(B)\alpha_{X_2}^{-1}(B) \cdots \alpha_{X_t}^{-1}(B)$ if $\zeta(B) = \nearrow$ and with $\alpha_{X_t}^{-1}(B)\alpha_{X_{t-1}}^{-1}(B) \cdots \alpha_{X_1}^{-1}(B)$ if $\zeta(B) = \searrow$.

To show that $\langle \prec^+, \sigma^+ \rangle$ is indeed an ℓ-page stack (queue) layout of G^+, we show that any conflict in $\langle \prec^+, \sigma^+ \rangle$ would also imply a conflict in $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$ in the full proof [14]. ◀

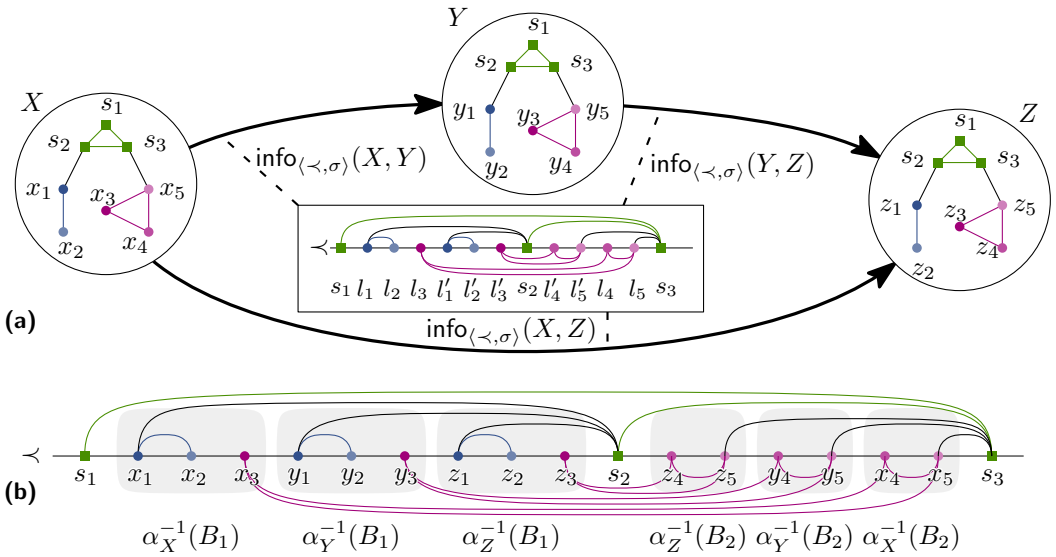


Figure 4 (a) The three large groups $X \prec Y \prec Z \in \mathcal{L} \setminus \{L, L'\}$ form a monochromatic clique in the auxiliary graph $\mathcal{H}$, i.e., we have $\mathsf{info}(X,Y) = \mathsf{info}(Y,Z) = \mathsf{info}(X,Z)$. The tuple (P,ζ), where P partitions R into two blocks $B_1 = \{r_1, r_2, r_3\}$ and $B_2 = \{r_4, r_5\}$ with $\zeta(B_1) = \nearrow$ and $\zeta(B_2) = \searrow$ fulfills the properties from Lemma 18. As X, Y, Z form a monochromatic clique, the solution blocks from **(b)** must occur in $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$.

4 An XP-Algorithm For Bounded-Width Linear Layouts

In this section, we present an algorithm to decide if a given graph G has a stack (or queue) layout on ℓ pages and with page width q, i.e., an ℓ-page q-width linear layout, in $n^{\mathcal{O}(q\cdot\ell)}$ time. In the affirmative case, it can also output such a layout. Since only little adaption is required between stack and queue layouts, we only discuss problem specific changes where necessary. We will first give an intuitive overview over our approach and the core ideas to showing its correctness, details can be found in Sections 4.1 and 4.2 and the full version.

Bounded-size Interfaces. Observe that, for each vertex $v \in V$ in a stack (or queue) layout $\langle \prec, \sigma \rangle$ of G, the edges $X_v^{\prec}$ that span the spine right next to v, induce the cut $(\mathrm{Left}_v^{\prec} \cup \{v\}, \mathrm{Right}_v^{\prec})$ in G; see Figure 5a. The size of the cut-set $F = X_v^{\prec}$ only depends on the number of the pages and their width. Our algorithm is centered around the insight that, through limiting these parameters and thus the size of any such cut, we limit the size of the "interface" (a notion we will define more precisely in a second) between a drawing on its left side and a drawing on its right side (Lemma 22). We note that this insight parallels to the approach used by Saxe [42] to test if a graph has bandwidth at most k. In our setting, replacing the left side of such cut in any solution drawing with one that provides the same interface will yield another solution. This is because having the same interface will imply inducing the same decompositions into left and right vertices (Lemma 24), the statement then follows by simply gluing the two drawings together along the cut.

Our bounded-size interface consists of the set of edges F that intersect the cut right next to v, their vertical order along the cut, and the information which edge endpoint lies on which side of the cut, together with a page assignment for the edges in F. We will encapsulate this information in what we call a (nicely) *oriented cut-set*; see Definition 21.

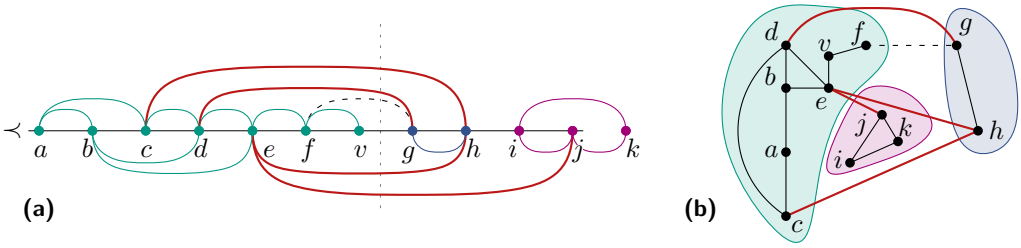


Figure 5 The two-page width-five stack layout in **(a)** induces the cut-set $F = X_v^{\prec}$ indicated with the red edges. **(b)** The three connected components in $G \setminus F$ are indicated with color. Observe that $(F, c \prec d \prec e \prec g \prec h \prec j)$ is nicely oriented. The source e and sink g cannot be in the same connected component, as every path between them contains at least one edge of F; e.g., the edge fg cannot exist.

Our Dynamic Programming Solution. Given only an arbitrary oriented cut-set F for an instance G, we are able to infer which connected components and thus which vertices of the graph lie on the left (or also right) side of the cut (Lemma 23; see also Figure 5b). This now allows us to solve this problem via dynamic programming. We build a directed acyclic state graph where each state corresponds to one configuration of our interface, i.e. one (nicely) oriented cut-set together with a page assignment for its edges (formalized as $\mathcal{N}$C 1 and 2 in the full version [14]). We add an arc between two states if we can move from one oriented cut-set to the other by moving exactly one vertex (which we inferred to be on the right side via Lemma 23) to the (end of a linear order used on the) left without directly violating a necessary stack or queue layout property (i.e., no crossing or nesting of edges) with regard to only the edges in the current cut-set; the exact conditions are again formalized as $\mathcal{A}$C 1–4 in the full version [14]. In Lemma 25, we will show that this means that we can extend any partial solution represented by the current state with the further vertex when moving along such an arc to obtain a solution for the new state. Then, we have a yes-instance if and only if there exists a directed path between the (artificially-distinguished) empty beginning and end states. As finding such path takes linear time, Theorem 2 then follows from the insight that the state graph has a size bounded by $n^{\mathcal{O}(q \cdot \ell)}$ and can be computed in this time.

4.1 Decomposing Linear Layouts Using Oriented Cut-Sets

As already hinted above, our algorithm uses at its core cut-sets of G to compute the desired stack (or queue) layout $\langle \prec_G, \sigma_G \rangle$. Before we discuss their usage in detail, let us first take a closer look at them and define their desired properties.

▶ **Observation 20.** *Given a connected graph G and a cut-set $F \subseteq E$ that separates the graph into $k \geq 2$ connected components $C_1, \ldots, C_k$. For all $i \in [k]$, component C_i contains at least one vertex that is an endpoint of an edge in F.*

We consider in addition to a cut-set F a total order $\prec_F$ on the endpoints of its edges and call this the *oriented cut-set*, denoted as $(F, \prec_F)$. While a single cut-set F has $\mathcal{O}((2\,|F|)!)$ different oriented cut-sets, we are only interested in "nicely oriented" cut-sets. Let F be a cut-set and $\prec_F$ a linear order on the endpoints of edges in F. A vertex $v \in V(F)$ is a *source* (or *sink*) in $(F, \prec_F)$ if for every edge $uv \in F$ we have $v \prec_F u$ ($u \prec_F v$, respectively).

▶ **Definition 21.** *For a cut-set F, $(F, \prec_F)$ is* nicely oriented *if and only if each vertex $v \in V(F)$ is either a source or sink, each connected component of $G - F$ contains either only sources or only sinks, and all sources are left of every sink in $\prec_F$.*

To ease presentation, we use a slight abuse of notation and allow the linear order $\prec_F$ of an oriented cut-set $(F, \prec_F)$ to be a total order of a super-set of $V(F)$. We consider $(F, \prec)$ and $(F, \prec')$ identical if $\prec$ and $\prec'$ agree on $V(F)$. We now establish a crucial connection between linear layouts and nicely oriented cut-sets of G; see also Figure 5.

▶ **Lemma 22 (★).** *Let G be a graph and $\langle \prec, \sigma \rangle$ be an ℓ-page q-width linear layout of G. For every vertex $v \in V$ that is not the rightmost in $\prec$, the edges $X_v^{\prec}$ form a cut-set of size at most $q \cdot \ell$ that induces the cut $(Left_v^{\prec} \cup \{v\}, Right_v^{\prec})$ in G. Furthermore, $(X_v^{\prec}, \prec)$ is nicely oriented.*

A liner order $\prec$ *induces* an oriented cut-set $(F, \prec')$ if there exists a vertex v that witnesses $(X_v^{\prec}, \prec) = (F, \prec')$. Note that the same oriented cut-set can be induced by different linear orders. From Lemma 22, we can deduce that all cuts (A, B) of an ℓ-page q-width stack (and queue) layout of G can be constructed using at most $\binom{m}{\ell \cdot \omega} = \mathcal{O}(m^{\ell \cdot \omega})$ different cut-sets. We aim to use their oriented counterparts to obtain the spine order $\prec$ of the desired stack (or queue) layout. We next show that we can efficiently compute for a given oriented cut-set $(F, \prec_F)$ a witnessing linear order that induces $(F, \prec_F)$.

▶ **Lemma 23 (★).** *Given a graph G and a nicely oriented cut-set $(F, \prec_F)$ of G, we can compute a linear order $\prec$ of V that induces $(F, \prec_F)$ in $\mathcal{O}(n + m)$ time.*

Lemma 23 only shows that we can compute *some* linear order $\prec$ of V that induces $(F, \prec_F)$. However, to efficiently use oriented cut-sets in our state graph, we must be able to precisely determine which vertices have already been placed on the spine without storing them explicitly. The following lemma lays the foundation for this, as it shows that no matter which linear order $\prec$ we choose to induce an oriented cut-set $(F, \prec_F)$, we obtain the same partition into left and right vertices.

▶ **Lemma 24 (★).** *Let $(F, \prec_F)$ be a nicely oriented cut-set. Furthermore, let $\prec$ and $\prec'$ be two linear orders such that $(X_v^{\prec}, \prec) = (X_u^{\prec'}, \prec') = (F, \prec_F)$ for some $u, v \in V$, i.e., they induce $(F, \prec_F)$. Then we have $Left_v^{\prec} \cup \{v\} = Left_u^{\prec'} \cup \{u\}$.*

4.2 Using Nicely Oriented Cut-Sets to Show Theorem 2

The state graph H contains a node for each possible nicely-oriented cut-set of G, together with the corresponding page assignment. The arcs in H represent possible transitions from one nicely oriented cut-set to another that can occur when traversing a hypothetical solution from left to right (see Figure 6), i.e., that are obtained by moving exactly one vertex from the right to the left. A state $S = (F, \prec_{F_X}, \sigma_{F_X})$ is *feasible* if and only if there exists an ℓ-page q-width stack layout $\langle \prec_S, \sigma_S \rangle$ of $G[L(S) \cup V(F)]$ such that for all $u, v \in V(F)$ we have $u \prec_F v \Leftrightarrow u \prec_S v$ and $\sigma_F = \sigma_S|_F$. For queue layouts, the definition is analogous. In the full version, we show that, essentially, having arcs only between state pairs where both induce valid stack (or queue) layouts on their respective cut-sets is enough to preserve feasibilty along arcs.

▶ **Lemma 25 (★).** *Let $H = (N, A)$ be a state graph and $S_X S_Y \in A$ one of its arcs. If S_X is feasible for stack (queue) layouts then so is S_Y.*

We now have all tools at hand to present our algorithm. Since the arcs of H preserve existence of a (partial) ℓ-page q-width stack (or queue) layout of G, we have reduced the problem of finding such a layout to the problem of finding a path in H between the two special nodes $S_\emptyset$ and S_V. In the full version, we show that H can be computed in polynomial time and use this to summarize the main result of this section in the following theorem.

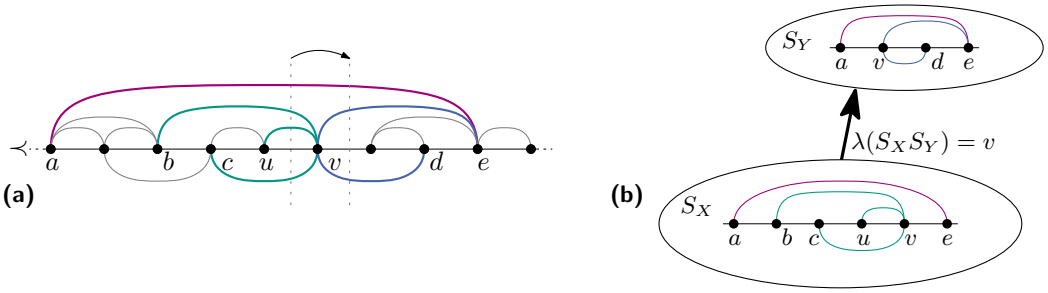


Figure 6 (a) Parts of a stack layout of G with the two cut-sets $X_u^{\prec}$ and $X_v^{\prec}$. **(b)** The corresponding nodes in the state graph. The arc in (b) corresponds to the transition from $X_u^{\prec}$ to $X_v^{\prec}$ in (a). Edges appearing in at least one cut-set are colored.

▶ **Theorem 2 (★).** *Given integers ℓ and q along with an n-vertex graph G, we can compute an ℓ-page stack or queue layout with page width at most q of G (if one exists) in time $n^{\mathcal{O}(q \cdot \ell)}$.*

The algorithm from Theorem 2 runs in polynomial time for constant ℓ and q. Furthermore, recall that deciding if a graph has a 2-page stack layout or 1-page queue layout is NP-complete [34, 46]. An ℓ-page 1-width stack or queue layout witnesses that G has cutwidth at most ℓ, which is in general NP-complete to decide [28]. This means that under well-established complexity assumptions, it is not possible to preserve XP-tractability if either of the two parameters is dropped.

5 Single-Exponential Algorithm for 1-Page Queue Layouts

In this section, we show that we can find a one-page queue layout, if it exists, in $2^{\mathcal{O}(n)}$ time, which improves on the trivial algorithm that exhaustively considers all $n!$ possible spine orders. Our algorithm exploits the equivalence between one-page queue layouts and arched leveled planar embeddings established by Heath and Rosenberg [34]. We first show that we can reduce the problem of deciding whether a graph admits a one-page queue layout to deciding whether at least one of $2^{\mathcal{O}(n)}$ *labeled* instances of Arched Leveled Planarity admits a solution. While Arched Leveled Planarity is NP-complete [34], our labeled instances can be solved in linear time thanks to a reduction to the well-studied problem Level Planarity [35, 36]. We will assume that G is connected (as each component can be treated separated) and, as G must be planar [34], we assume $|E| = \mathcal{O}(n)$.

We translate the definition of Heath and Rosenberg [34] into moderns terms as follows. Graph G has a *leveled planar embedding* if there exists a *level assignment* function $\gamma \colon V \to [h]$ that maps each $v \in V$ to one of $h \geq 1$ *levels*, and a straight-line planar drawing Γ of G such that each vertex v has y-coordinate $\gamma(v)$ and all edges are *proper*, i.e., for all $uv \in E$ we have $|\gamma(u) - \gamma(v)| = 1$. The latter property allows us to express the drawing as a collection of total orders $\prec_i$, one for each level $i \in [h]$, that specifies the left-to-right order of the vertices on level i [36]. For each level i, let s_i be the rightmost vertex on level i in Γ such that we have $s_i w \in E$ for some $w \in V$ with $\gamma(w) = i+1$, or if there is no such vertex, let s_i be the leftmost vertex on level i. *Arched leveled planar embeddings* extend leveled planar embeddings by also allowing non-monotone *arching* edges $t_i w$ between the leftmost vertex t_i on level i and vertices $w \in V$ with $\gamma(w) = i$ and $s_i \preceq_i w$ [34]. These edges can be drawn crossing-free below the lowest level; see Figure 7b. The problem Arched Leveled Planarity asks whether G has a level assignment that admits an arched leveled planar embedding. Heath

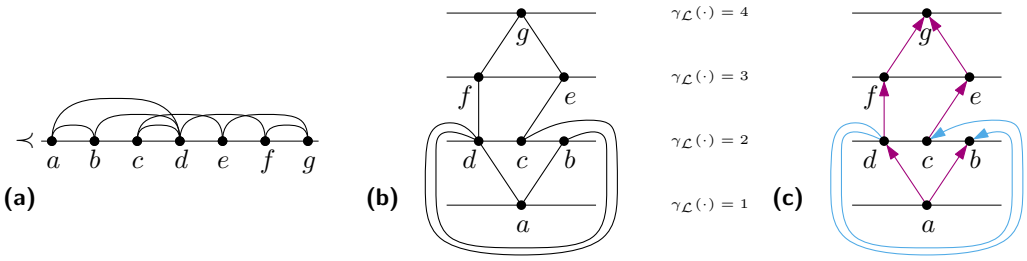


Figure 7 (a) A one-page queue layout of G and **(b)** the corresponding arched leveled planar embedding obtained by the equivalence established by Heath and Rosenberg [34]. **(c)** The labeling induced by the embedding from **(b)**: Arcs labeled $\uparrow$ and $\smile$ are colored lilac and blue, respectively.

and Rosenberg [34] showed that the spine order of a one-page queue layout of a graph G induces a level assignment together with a vertex order on each level that together yield an arched leveled planar embedding of G and vice versa; see Figure 7.

We show that, when seeking a one-page queue layout of G, one can branch over the information needed to infer the level assignment for a corresponding arched leveled planar embedding. Consider an arbitrary orientation of the edge set E as an oriented arc set A and let $\delta\colon A \to \{\uparrow, \smile\}$ be a function that determines if an arc $uv \in A$ should be *ordinary* ($\uparrow$) or *arching* ($\smile$). The tuple $\mathcal{L} = (A, \delta)$ is called a *labeling* of G; see also Figure 7c. A level assignment γ is said to be *consistent* with $\mathcal{L}$ if and only if, for every $uv \in A$, it holds $\gamma(u) = \gamma(v) - 1$ if $\delta(uv) = \uparrow$ and $\gamma(u) = \gamma(v)$ if $\delta(uv) = \smile$. Similarly, an embedding Γ is consistent with $\mathcal{L}$ if its level assignment is consistent and we have $u \prec_{\gamma(u)} v$ for every $uv \in \delta^{-1}(\smile)$. Clearly, every arched leveled planar embedding Γ of G induces a labeling $(A_\Gamma, \delta_\Gamma)$ that is consistent with Γ. Conversely, we can reobtain Γ from this $(A_\Gamma, \delta_\Gamma)$ by fixing an arbitrary vertex to level 0, performing a BFS from v using the direction information in A_Γ to determine levels for all other vertices above and below, and potentially shifting all assigned levels such that their numbers span the same range as those in Γ. This is formalized by the following lemma.

▶ **Lemma 26 (★).** *Let $\mathcal{L} = (A, \delta)$ be a labeling of G. In linear time we can compute a level assignment $\gamma_{\mathcal{L}}\colon V \to [h]$ that is consistent with the labeling or report that no such assignment exists. If there exists an arched leveled planar embedding Γ that induces $\mathcal{L}$, then there also exists one that induces $\mathcal{L}$ and uses the level assignment $\gamma_{\mathcal{L}}$.*

To now decide if a given graph G has an arched leveled planar embedding Γ, we can branch over all $\mathcal{O}(4^n) = 2^{\mathcal{O}(n)}$ different labelings $\mathcal{L} = (A, \delta)$ of G. In each branch, we use Lemma 26 to compute a level assignment $\gamma_{\mathcal{L}}$ or immediately reject the branch. Thus, it remains to check in each branch, i.e., for each level assignment, if G admits a corresponding arched leveled planar embedding. We do so using a linear-time reduction to the problem Level Planarity, which requires a level assignment as part of its input but does not allow arching edges. We emulate arching edges in Level Planarity as follows (see also Figure 8). First, we enclose the whole graph in a *frame*, which is a cycle spanning from below its lowest level to above its highest that will have a unique embedding (up to reflection) and for which we ensure that the remaining graph is drawn inside of it. Between each pair of levels $i, i+1$, we add a level $i + 0.5$ and subdivide all edges to ensure properness. We use l_i and r_i to refer to the two vertices of the frame cycle that are on level $i + 0.5$. Replace each arching arc $uv \in A$ with $\delta(uv) = \smile$ on level i with the edges ul_{i-1}, ul_i, and vr_i to G'; see Figure 8. Observe that this forces u to become the leftmost vertex on level i and v to be right of any

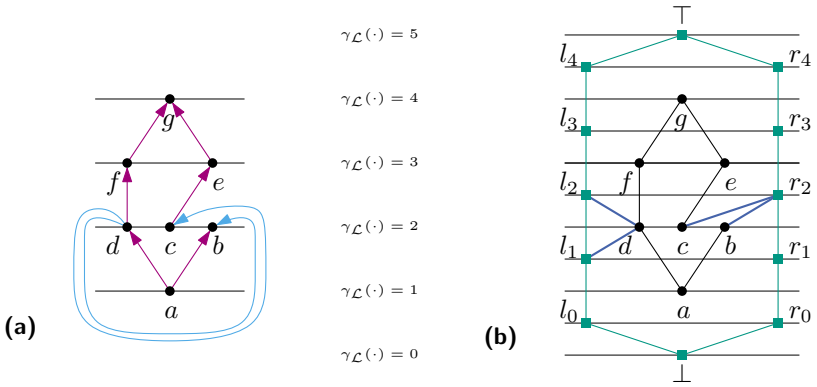


Figure 8 (a) An instance of Labeled Arched Level Planarity and **(b)** the obtained instance of Level Planarity. We also visualize a drawing and color the frame and edges that simulate the arching edges from **(a)** in green and blue, respectively. Only relevant subdivision vertices are shown.

vertex adjacent to one on level $i+1$. We provide a formal construction and show equivalence in the full version [14]. Using the known algorithm for testing Level Planarity in linear time [35, 36], we finally obtain our third contribution.

▶ **Theorem 3 (★).** *Given an n-vertex graph G, we can compute a 1-page queue layout of G (if one exists) in time $2^{\mathcal{O}(n)}$.*

6 Concluding Remarks

While our results improve the state of the art on computing linear layouts in several directions, they also highlight the prominent open questions in this area. In particular, Theorem 1 moves us closer to settling the long-standing open questions of whether treewidth or treedepth can be used to facilitate the computation of linear layouts [7, 8, 21, 25, 26]. At the same time, Theorems 2 and 3 yield the question of whether these classical problems can be solved in single-exponential time.

References

1 Akanksha Agrawal, Sergio Cabello, Michael Kaufmann, Saket Saurabh, Roohani Sharma, Yushi Uno, and Alexander Wolff. Eliminating Crossings in Ordered Graphs. In Hans L. Bodlaender, editor, *Proc. 19th Scandinavian Workshop on Algorithm Theory (SWAT'24)*, volume 294 of *LIPIcs*, pages 1:1–1:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, 2024. doi:10.4230/LIPICS.SWAT.2024.1.

2 Noga Alon and Vojtěch Rödl. Asymptotically tight bounds for some multicolored ramsey numbers.

3 Patrizio Angelini, Michael A. Bekos, Philipp Kindermann, and Tamara Mchedlidze. On mixed linear layouts of series-parallel graphs. *Theoretical Computer Science*, 936:129–138, 2022. doi:10.1016/J.TCS.2022.09.019.

4 Michael Bannister and David Eppstein. Crossing Minimization for 1-page and 2-page Drawings of Graphs with Bounded Treewidth. *Journal of Graph Algorithms and Applications*, 22(4):577–606, 2018. doi:10.7155/JGAA.00479.

5 Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou. An Improved Upper Bound on the Queue Number of Planar Graphs. *Algorithmica*, 85(2):544–562, 2023. doi:10.1007/S00453-022-01037-4.

6 Sujoy Bhore, Giordano Da Lozzo, Fabrizio Montecchiani, and Martin Nöllenburg. On the Upward Book Thickness Problem: Combinatorial and Complexity Results. In Helen C. Purchase and Ignaz Rutter, editors, *Proc. 29th International Symposium on Graph Drawing and Network Visualization (GD'21)*, volume 12868 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2021. doi:10.1007/978-3-030-92931-2_18.

7 Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized Algorithms for Book Embedding Problems. *Journal of Graph Algorithms and Applications*, 24(4):603–620, 2020. doi:10.7155/JGAA.00526.

8 Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized Algorithms for Queue Layouts. *Journal of Graph Algorithms and Applications*, 26(3):335–352, 2022. doi:10.7155/JGAA.00597.

9 Hans L. Bodlaender, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Yoshio Okamoto, Yota Otachi, and Tom C. van der Zanden. Subgraph Isomorphism on Graph Classes that Exclude a Substructure. *Algorithmica*, 82(12):3566–3587, 2020. doi:10.1007/S00453-020-00737-Z.

10 Fan R. K. Chung, Frank Thomson Leighton, and Arnold L. Rosenberg. Embedding graphs in books: a layout problem with applications to VLSI design. *SIAM Journal on Algebraic Discrete Methods*, 8(1):33–58, 1987. doi:10.1137/0608002.

11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

12 Thomas Depian, Simon D. Fink, Robert Ganian, and Martin Nöllenburg. The Parameterized Complexity Of Extending Stack Layouts. In Stefan Felsner and Karsten Klein, editors, *Proc. 32nd International Symposium on Graph Drawing and Network Visualization (GD'24)*, volume 320 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.GD.2024.12.

13 Thomas Depian, Simon D. Fink, Robert Ganian, and Martin Nöllenburg. The Peculiarities of Extending Queue Layouts. In *Proc. 51st International Workshop on Graph-Theoretic Concepts in Computer Science (WG'25)*, 2025. To appear.

14 Thomas Depian, Simon D. Fink, Robert Ganian, and Vaishali Surianarayanan. Linear Layouts Revisited: Stacks, Queues, and Exact Algorithms, 2025. Full Version. doi:10.48550/arXiv.2508.16319.

15 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2012.

16 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/S00453-016-0127-X.

17 Vida Dujmović, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar Graphs Have Bounded Queue-Number. *Journal of the ACM*, 67(4):22:1–22:38, 2020. doi:10.1145/3385731.

18 Vida Dujmovic, Pat Morin, and David R. Wood. Layout of Graphs with Bounded Tree-Width. *SIAM Journal on Computing*, 34(3):553–579, 2005. doi:10.1137/S0097539702416141.

19 Vida Dujmović and David R. Wood. On Linear Layouts of Graphs. *Discrete Mathematics & Theoretical Computer Science*, 6(2):339–358, 2004. doi:10.46298/DMTCS.317.

20 Vida Dujmovic and David R. Wood. Graph Treewidth and Geometric Thickness Parameters. *Discrete & Computational Geometry*, 37(4):641–670, 2007. doi:10.1007/S00454-007-1318-7.

21 Vida Dujmović and David R. Wood. On the Book Thickness of k-Trees. *Discrete Mathematics & Theoretical Computer Science*, 13(3):39–44, 2011. doi:10.46298/DMTCS.550.

22 Pavel Dvorák, Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. Solving Integer Linear Programs with a Small Number of Global Variables and Constraints. In Carles Sierra, editor, *Proc 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 607–613, 2017. doi:10.24963/IJCAI.2017/85.

23 Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. he complexity landscape of decompositional parameters for ILP: Programs with few global variables and constraints. *Artificial Intelligence*, 300:103561, 2021. doi:10.1016/J.ARTINT.2021.103561.

24 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On Structural Parameterizations of the Bounded-Degree Vertex Deletion Problem. *Algorithmica*, 83(1):297–336, 2021. doi:10.1007/S00453-020-00758-8.

25 Robert Ganian, Fabrizio Montecchiani, Martin Nöllenburg, and Meirav Zehavi. Parameterized Complexity in Graph Drawing (Dagstuhl Seminar 21293). *Dagstuhl Reports*, 11(6):82–123, 2021. doi:10.4230/DAGREP.11.6.82.

26 Robert Ganian, Haiko Müller, Sebastian Ordyniak, Giacomo Paesani, and Mateusz Rychlicki. A Tight Subexponential-Time Algorithm for Two-Page Book Embedding. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *Proc. 51st International Colloquium on Automata, Languages and Programming (ICALP'24)*, volume 297 of *LIPIcs*, pages 68:1–68:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.68.

27 Robert Ganian, Sebastian Ordyniak, and Maadapuzhi Sridharan Ramanujan. On Structural Parameterizations of the Edge Disjoint Paths Problem. *Algorithmica*, 83(6):1605–1637, 2021. doi:10.1007/S00453-020-00795-3.

28 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

29 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theoretical Computer Science*, 918:60–76, 2022. doi:10.1016/J.TCS.2022.03.021.

30 Tesshu Hanaka, Michael Lampis, Manolis Vasilakis, and Kanae Yoshiwatari. Parameterized Vertex Integrity Revisited. In Rastislav Královic and Antonín Kucera, editors, *Proc. 49th International Symposium on Mathematical Foundations of Computer Science (MFCS'24)*, volume 306 of *LIPIcs*, pages 58:1–58:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.MFCS.2024.58.

31 Christian Haslinger and Peter F. Stadler. RNA Structures with Pseudo-knots: Graph-theoretical, Combinatorial, and Statistical Properties. *Bulletin of Mathematical Biology*, 61(3):437–467, 1999. doi:10.1006/bulm.1998.0085.

32 Lenwood S. Heath. Embedding outerplanar graphs in small books. *SIAM Journal on Algebraic Discrete Methods*, 8(2):198–218, 1987. doi:10.1137/0608018.

33 Lenwood S. Heath and Sriram V. Pemmaraju. Recognizing Leveled-Planar Dags in Linear Time. In Franz-Josef Brandenburg, editor, *Proc. 3rd International Symposium on Graph Drawing and Network Visualization (GD'95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 300–311. Springer, 1996. doi:10.1007/BFB0021813.

34 Lenwood S. Heath and Arnold L. Rosenberg. Laying out Graphs Using Queues. *SIAM Journal on Computing*, 21(5):927–958, 1992. doi:10.1137/0221055.

35 Michael Jünger and Sebastian Leipert. Level Planar Embedding in Linear Time. In Jan Kratochvíl, editor, *Proc. 7th International Symposium on Graph Drawing and Network Visualization (GD'99)*, volume 1731 of *Lecture Notes in Computer Science*, pages 72–81. Springer, 1999. doi:10.1007/3-540-46648-7_7.

36 Michael Jünger and Sebastian Leipert. Level Planar Embedding in Linear Time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002. doi:10.7155/JGAA.00045.

37 Michael Jünger, Sebastian Leipert, and Petra Mutzel. Level Planarity Testing in Linear Time. In Sue Whitesides, editor, *Proc. 6th International Symposium on Graph Drawing and Network Visualization (GD'98)*, volume 1547 of *Lecture Notes in Computer Science*, pages 224–237. Springer, 1998. doi:10.1007/3-540-37623-2_17.

38 Julia Katheder, Michael Kaufmann, Sergey Pupyrev, and Torsten Ueckerdt. Transforming Stacks into Queues: Mixed and Separated Layouts of Graphs. In Olaf Beyersdorff, Michal Pilipczuk, Elaine Pimentel, and Kim Thang Nguyen, editors, *Proc. 42nd Symposium on Theoretical Aspects of Computer Science (STACS'25)*, volume 327 of *LIPIcs*, pages 56:1–56:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.STACS.2025.56.

39 Michael Lampis and Valia Mitsou. Fine-grained Meta-Theorems for Vertex Integrity. *Logical Methods in Computer Science*, 20, 2024. doi:10.46298/LMCS-20(4:18)2024.

40 Yunlong Liu, Jie Chen, Jingui Huang, and Jianxin Wang. On parameterized algorithms for fixed-order book thickness with respect to the pathwidth of the vertex ordering. *Theoretical Computer Science*, 873:16–24, 2021. doi:10.1016/J.TCS.2021.04.021.

41 Arnold L. Rosenberg. Book embeddings and wafer-scale integration. In *Proc. 17th Southeastern International Conference on Combinatorics, Graph Theory, and Computing*, volume 54, pages 217–224, 1986.

42 James B. Saxe. Dynamic-Programming Algorithms for Recognizing Small-Bandwidth Graphs in Polynomial Time. *SIAM Journal on Algebraic Discrete Methods*, 1(4):363–369, 1980. doi:10.1137/0601042.

43 Elena Stöhr. A Trade-off between Page Number and Page Width of Book Embeddings of Graphs. *Information and Computation*, 79(2):155–162, 1988. doi:10.1016/0890-5401(88)90036-3.

44 Elena Stöhr. The pagewidth of trivalent planar graphs. *Discrete Mathematics*, 89(1):43–49, 1991. doi:10.1016/0012-365X(91)90398-L.

45 Veit Wiechert. On the queue-number of graphs with bounded tree-width. *Electron. J. Comb.*, 24(1):1, 2017. doi:10.37236/6429.

46 Mihalis Yannakakis. Embedding Planar Graphs in Four Pages. *Journal of Computer and System Sciences*, 38(1):36–67, 1989. doi:10.1016/0022-0000(89)90032-9.

On Algorithmic Applications of $\mathcal{F}$-Branchwidth

Benjamin Bergougnoux
Aix-Marseille Université, LIS, CNRS, France

Thekla Hamm
TU Eindhoven, The Netherlands

Lars Jaffke
NHH Norwegian School of Economics, Bergen, Norway

Paloma T. Lima
IT University of Copenhagen, Denmark

Abstract

$\mathcal{F}$-branchwidth is a framework for width measures of graphs, recently introduced by Eiben et al. [ITCS 2022], that captures tree-width, co-tree-width, clique-width, and mim-width, and several of their generalizations and interpolations. In this work, we search for algorithmic applications of $\mathcal{F}$-branchwidth measures that do not have an equivalent counterpart in the literature so far. Our first contribution is a minimal set of eleven $\mathcal{F}$-branchwidth measures such that each of the infinitely many $\mathcal{F}$-branchwidth measures is equivalent to one of the eleven. We observe that for the FO Model Checking problem, each $\mathcal{F}$-branchwidth is either equivalent to clique-width (and therefore has an FPT-algorithm by formula length plus the width) or the problem remains as hard as on general graphs even on graphs of constant width. Next, we study the number of equivalence classes of the neighborhood equivalence in a decomposition, which upper bounds the run time of the model checking algorithm for A&C DN logic recently introduced by Bergougnoux et al. [SODA 2023]. We give structural lower bounds that show that for each $\mathcal{F}$-branchwidth, an efficient model checking algorithm was already known or cannot be obtained via this method. Lastly, we classify the complexity of Independent Set parameterized by any $\mathcal{F}$-branchwidth except for one open case. Also here, our contributions are lower bounds. In this context, we also prove that Independent Set on graphs of mim-width w cannot be solved in time $n^{o(w)}$ unless the Exponential Time Hypothesis fails, answering an open question in the literature.

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

Keywords and phrases Graph width parameters, parameterized complexity, F-branchwidth, tree-width, clique-width, rank-width, mim-width, FO model checking, DN logic, Independent Set, ETH

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.16

Related Version A full version of the paper is currently in preparation.

Funding *Thekla Hamm*: Supported during part of this research by the Austrian Science Fund FWF project number J4651.
Paloma T. Lima: Supported by the Independent Research Fund Denmark grant agreement number 2098-00012B.

Acknowledgements We thank Édouard Bonnet for the proof strategy behind Lemma 41.

1 Introduction

Eiben et al. [12] recently introduced *$\mathcal{F}$-branchwidth*, a framework of graph width measures that allows for an infinite number of instantiations. Among them are width measures that are asymptotically equivalent to treewidth, clique-width, and mim-width. These three have wide algorithmic applications which immediately begs the following question:

Are there any *unknown* $\mathcal{F}$-branchwidth measures that have interesting algorithmic applications?


© Benjamin Bergougnoux, Thekla Hamm, Lars Jaffke, and Paloma T. Lima;
licensed under Creative Commons License CC-BY 4.0



The goal of this paper is to address this question – to which we provide negative answers in several contexts.

For a family of bipartite graphs $\mathcal{F}$, the $\mathcal{F}$-branchwidth of a graph G bounds the size of any member of $\mathcal{F}$ that appears as a semi-induced subgraph in a recursive partitioning of the vertex set of G. Therefore, any family of bipartite graphs $\mathcal{F}$ gives rise to a new $\mathcal{F}$-branchwidth measure. To avoid chaotic definitions, some technical restrictions are imposed on $\mathcal{F}$, but there is still an infinite number of families $\mathcal{F}$ satisfying them. However, as already observed in [12], there is a finite number of classes of $\mathcal{F}$-branchwidth measures that are bounded on different families of graphs. This means that we can restrict the search for new algorithmically useful width measures to the representatives of these classes. These are obtained as any combination of the six families shown in Figure 1.

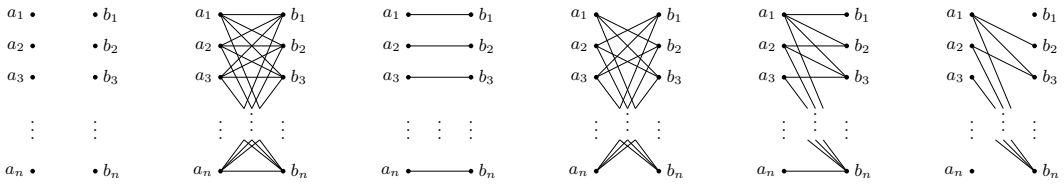


Figure 1 The six canonical families usable as building blocks for $\mathcal{F}$-branchwidth. From left to right: edgeless (⁝ ⁝), complete (⨉), matchings (=), antimatchings (⋈), chains (⧅), and skewed chains (⧅).

This brings down the number of cases to consider from infinite to constant, if one wants to determine the complexity of a problem parameterized by *any* of the possible $\mathcal{F}$-branchwidth measures. However, a priori there are still 2^6 cases to consider. Our first contribution is to provide a family of eleven $\mathcal{F}$-branchwidth, called *The Eleven*, that captures the modeling power of every possible $\mathcal{F}$-branchwidth in the following sense. For an illustration see Figure 2.

▶ **Theorem 1.** *For each $\mathcal{F}$-branchwidth, one of the following holds.*

(i) *There is a computable function f such that for each graph G with $\mathcal{F}$-$\mathsf{bw}(G) \leq k$, $|V(G)| \leq f(k)$.*

(ii) *$\mathcal{F}$-branchwidth is asymptotically equivalent to $\mathcal{F}^*$-branchwidth, where $\mathcal{F}^*$ is one of the Eleven branchwidths (presented in Figure 2 and formally defined in Table 1).*

We provide families of obstructions to bounded width that show that the classification in Theorem 1 is in fact minimal, in the sense that no pair of width measures can be asymptotically equivalent to each other.

▶ **Theorem 2.** *For each pair of distinct $\mathcal{F}_1, \mathcal{F}_2 \in \mathfrak{F}$ there is some $i \in [2]$ such that there is a graph class of bounded $\mathcal{F}_i$-width but unbounded $\mathcal{F}_{3-i}$-width.*

Throughout the following, we refer to the $\mathcal{F}$-branchwidth measures based on the families in $\mathfrak{F}$ as the *canonical* ones. Notice that the second item of Theorem 1 says that $\mathcal{F}$-branchwidth and $\mathcal{F}^*$-branchwidth are bounded on the same graph classes, but the exact bounds may differ. Theorem 1, along with the hierarchy among the eleven relevant width measures, allows us to determine the complexity of a problem parameterized by any $\mathcal{F}$-branchwidth by considering these cases alone.

A common way to classify the algorithmic power of a width measure is to determine for which logic L, the L-Model Checking problem can be solved efficiently whenever the input graph is given together with a decomposition of constant width. For instance, an

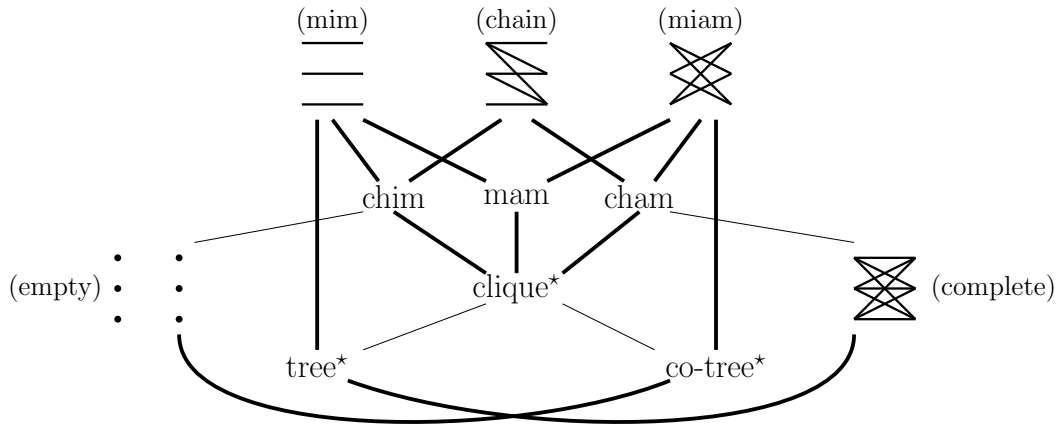


Figure 2 The Eleven canonical $\mathcal{F}$-branchwidth measures, ordered according to expressive power from top (high) to bottom (low). The five base cases show the singleton classes, and bold edges show combinations. For instance, chim-width is ($\equiv \cup \lessapprox$)-branchwidth, clique*-width is ($\equiv \cup \lessapprox \cup \approx$)-branchwidth, and tree*-width is ($\equiv \cup \approx$)-branchwidth. The remaining edges show some relationships given by the structures of the families. As one may expect, tree*-width is asymptotically equivalent to tree-width, clique*-width is asymptotically equivalent to clique-width, and co-tree*-width is asymptotically equivalent to co-tree-width.

L-formula ϕ may express that a graph has an independent set of size k. Then, the evaluation of ϕ at a graph G is true, in symbols $G \models \phi$ (read: "G models ϕ"), if and only if G has an independent set of size k. Typically, one aims for efficient parameterized algorithms in the parameterization formula length plus width.

L-Model Checking parameterized by $|\phi| + w$

Input:	Graph G given together with a decomposition of width w, L-formula ϕ.
Question:	Does $G \models \phi$?

Classical examples of such meta-theorems are Courcelle's Theorem [8] which gives an FPT-algorithm when L is MSO_2 logic and the width is tree-width and its extension to the more general clique-width while restricting the logic to MSO_1 [9]. More recent examples include an FPT-algorithm for FO logic and twin-width [6], and an XP-algorithm for A&C DN logic and mim-width [3]. The following hierarchy holds for the first three logics: $\mathsf{FO} \subsetneq \mathsf{MSO}_1 \subsetneq \mathsf{MSO}_2$.

We consider the Model Checking problem for any of these logics parameterized by any of the $\mathcal{F}$-branchwidth measures. Our first result is that for MSO/FO, we cannot expect any new tractability islands via $\mathcal{F}$-branchwidth, so in fact, our current understanding of the complexity of MSO/FO Model Checking is tight for any $\mathcal{F}$-branchwidth. We illustrate the following theorem for the canonical $\mathcal{F}$-branchwidth measures in Figure 3.

▶ **Theorem 3.** *For each $\mathcal{F}$-branchwidth, one of the following holds.*

1. *There is a computable function f such that each graph of $\mathcal{F}$-branchwidth k has clique-width at most $f(k)$, implying that* FO *Model Checking is* FPT *parameterized by formula length plus k.*
2. *There is a constant c such that* FO *Model Checking parameterized by formula length is* AW[$*$]*-hard even when the input graph is given with one of its decompositions of $\mathcal{F}$-branchwidth c.*

Next, we move on to A&C DN logic which was recently introduced by Bergougnoux et al. [3] as a logic to capture the problems that are solvable in XP time parameterized by the mim-width of a given decomposition of the input graph. This logic is a good target to study under the lens of $\mathcal{F}$-branchwidth. It is the right fit for mim-width which is Ξ-branchwidth where Ξ is the family of matchings – one of the base families of $\mathcal{F}$-branchwidth. A&C DN logic is essentially[1] based on existential MSO_1 and a *neighborhood operator* which, for a given set variable X, returns the set $\bigcup_{v \in X} N(v)$, and predicates checking acyclicity and connectivity. The runtime of the algorithm for A&C DN Model Checking is bounded by the index of the *neighborhood equivalence relation* (nec), first studied by Bui-Xuan et al. [7].To get efficient algorithms, it therefore suffices to show, for graphs of width k, that the index of the neighborhood equivalence grows relatively slowly in terms of n and k. For instance, a bound of $f(k) \cdot n^{\mathcal{O}(1)}$ implies an $f(k) \cdot n^{\mathcal{O}(1)}$ time algorithm for A&C DN Model Checking, while a bound of $n^{g(k)}$ implies an $n^{\mathcal{O}(g(k))}$ time algorithm, using the meta-theorem of [3].

Several such bounds on the index of this equivalence relation are already known in the literature. First, mim-width k implies that there are at most n^k equivalence classes [2]. Furthermore, there are bounds of the form $2^{\mathcal{O}(k)}$ when k is either the tree-width, clique-width, or co-tree-width (see, for instance, [3]). Next, we cannot hope to obtain any meaningful bound for any $\mathcal{F}$-branchwidth that generalizes complete-width. When a cut consists of matching with n vertices on each side, the complete-width is just 1, while the number of neighborhoods across the cut is 2^n. Indeed, for each subset of vertices contained in one side of the cut, there is a different subset across the cut whose neighborhood is exactly that subset.

The cases that remain are the ones asymptotically equivalent to empty-width, chim-width, and mam-width. As mim-width generalizes all of them, we immediately have XP-algorithms parameterized by each of them. Also here, we provide a number of negative results. For chim-width and mam-width, not much improvement can be made over the bound given by mim-width. We show that there are graphs of chim-width (or mam-width) k such that each of their decompositions induces a cut with at least $(n/(k^2 \log n))^{\Omega(k)}$ neighborhoods. By a probabilistic argument, we show that there are graphs whose decompositions have cuts of empty-width k that have $n^{\Omega(k)}$ neighborhoods. We conclude that the technology from [3] cannot be applied directly to any $\mathcal{F}$-branchwidth measure to extend the range of tractability of A&C DN Model Checking, see Figure 3 for an illustration.

▶ **Theorem 4.** *For each $\mathcal{F}$-branchwidth, one of the following holds.*

1. *There is a computable function f such that each graph of $\mathcal{F}$-branchwidth k has clique-width at most $f(k)$, implying that* A&C DN *Model Checking is* FPT *parameterized by formula length plus the width of a given decomposition of the input graph.*
2. *Each graph of $\mathcal{F}$-branchwidth k has mim-width at most k, implying that* A&C DN *Model Checking is* XP *parameterized by formula length plus the width of a given decomposition of the input graph. On the other hand, there are cuts of $\mathcal{F}$-branchwidth k with $n^{\Omega(k)}$ neighborhood equivalence classes.*
3. *There is a computable function f such that each graph of complete-width at most k has $\mathcal{F}$-branchwidth at most $f(k)$, implying that there are cuts on n vertices of constant $\mathcal{F}$-branchwidth with $2^{\Omega(n)}$ neighborhood equivalence classes.*

[1] The neighborhood operator in [3] is somewhat more general, but for the sake of this introduction it suffices to think of simple neighborhoods.

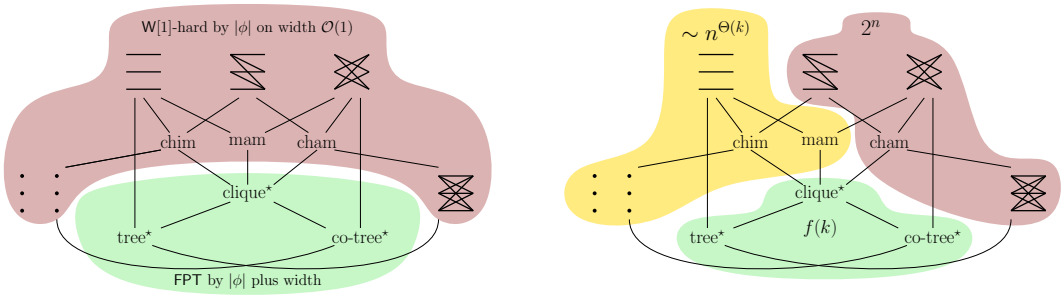


Figure 3 Results about the complexity of model checking problems on graphs of bounded $\mathcal{F}$-branchwidth. Left: Complexity of FO Model Checking. Right: Number of neighborhoods in cuts of bounded width.

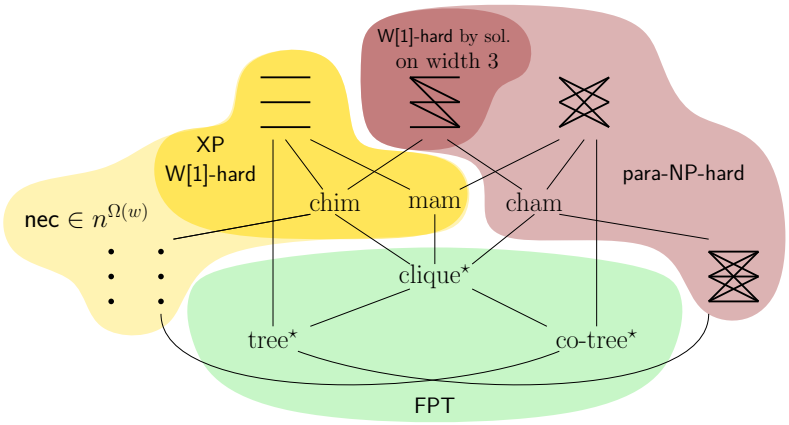


Figure 4 The complexity of Independent Set parameterized by any $\mathcal{F}$-branchwidth measure. Note that the complexity of Independent Set parameterized by empty-width is still open, but the XP lower bound on the number of equivalence classes in the neighborhood equivalence implies that known techniques are not applicable to show fixed-parameter tractability in this case. Moreover, it is worth looking into the complexity of IS parameterized by solution size on graphs of constant miam-width, cham-width, or complete-width.

For the canonical $\mathcal{F}$-branchwidth measures, we also take a more fine-grained look at the index of the neighborhood equivalence. The upper bounds on the number of equivalence classes in terms of clique*-width and tree*-width follow from known bounds [3] together with Ramsey-theory based arguments from [12]. This results in upper bounds that are double-exponential in the width k, i.e., bounds of the form $2^{2^{\mathsf{poly}(k)}}$. For tree*-width and co-tree*-width k, we give improved, single-exponential, bounds of $2^{\mathcal{O}(k^3 \log k)}$ each, and a lower bound of $2^{\Omega(k \log k)}$ for tree*-width k.

For the lower bound statements of Theorem 4, observe two things. First, for chim-width and mam-width, the bounds in fact hold for *any* decomposition of graphs within a family we construct. Second, lower bounds on the number of equivalence classes of the neighborhood equivalence only rule out direct applications of the meta-theorem from [3], and are not hardness results per se.

Due to that, we explicitly consider the Independent Set and Dominating Set problems as two of the most fundamental problems expressible in A&C DN logic. Here, we provide several new hardness results. First, we show W[1]-hardness parameterized by chim-width and mam-width, and we show that under the Exponential Time Hypothesis, no $n^{o(k)}$ time

algorithm can exist. The latter gives a negative answer to the question explicitly asked for instance in [1, 4] whether the linear dependence on the mim-width in the exponent of the algorithms from [7] can be lowered significantly.

▶ **Theorem 5.** *Unless the Exponential Time Hypothesis fails, neither* Independent Set *nor* Dominating Set *can be solved in $f(k)n^{o(k)}$ time for any computable function f, where k is either the chim-width, the mam-width, or the mim-width of a given decomposition of the n-vertex input graph. Moreover, both problems parameterized by either the chim-width or the mam-width are* W[1]*-hard.*

Since Independent Set is NP-complete on cubic graphs, it is para-NP-hard when parameterized by any width measure in whose obstructions the degree increases proportionally to the size. This is the case for complete bipartite graphs, anti-matchings, and chains. However, one might hope for efficient algorithms for Independent Set parameterized by solution size on graphs of constant width. For chain-width, we rule out this possibility by proving that the graphs from a reduction due to Fomin, Golovach, and Raymond [13] have bounded chain-width.

▶ **Theorem 6.** *Independent Set parameterized by solution size is* W[1]*-hard even when the input graph is provided with a linear decomposition of chain-width* 3*.*

We summarize the results for the Independent Set problem in Figure 4.

This paper is organized as follows. In the next section, we introduce our notations and the $\mathcal{F}$-branchwidths framework, as wells as some basics facts on these parameters. Theorem 2 is proved in Section 3. Theorem 3 is proved in Section 4. Theorem 4 is proved in Section 5. Finally, Theorem 5 is proved in Section 5. Missing proofs are in the full version.

2 Preliminaries

For an integer i, we let $[i] = \{1, 2, \ldots, i\}$.

We use standard graph terminology and notation [11], some of which we briefly recall in the following. In every notation with a graph subscript, we may omit it if the graph is clear from the context.

A *cut* of a graph G is a partition of $V(G)$ into two sets A, B. We denote by $G[A, B]$ the subgraph of G with edge set $\{uv \in E(G) \mid u \in A, v \in B\}$. A graph G is *bipartite* if it can be written as $G[A, B]$ for some cut A, B. In this case we also use (A, B, E) to denote G. If $|A| = |B|$, then we call $|V(G)/2|(= |A| = |B|)$ the *half-order* of G.

For a graph G, its *(edge) complement* $\overline{G}$ is given by $(V(G), \{uv \mid u, v \in V(G) \wedge vw \notin E(G)\})$. Given a set $A \subseteq V(G)$, we use $\overline{A}$ as shorthand for $V(G) \setminus A$. For a bipartite graph $G = (A, B, E)$, its *bipartite (edge) complement* is given by $(A, B, \{uv \mid u \in A \wedge v \in B \wedge uv \notin E\})$.

Branch decomposition. A *branch decomposition* or *tree layout* (or simply *layout*) of a graph G is a pair (T, λ) where T is a ternary tree (i.e., every internal node of T has degree 3) and λ is a bijection from $V(G)$ to the leaves of T. An edge e of T *induces* or *defines* a cut (X_e, Y_e) of G, where X_e and Y_e are the preimages by λ of the leaves in the two components of $T - e$. For a symmetric function $\mathsf{f}_G \colon 2^{V(G)} \to \mathbb{Z}$ which we refer to as a *cut function* of G, the f*-width* of e is $\mathsf{f}_G(X_e)$, the f-width of the branch decomposition T is the maximum width over all edges of T, and the f-branchwidth of G, denoted by $\mathsf{f}\text{-}\mathsf{width}(G)$, is the minimum width of a branch decomposition of G.

The following lemma encapsulates a standard argument on branch decompositions that we need to obtain some of our lower bound results.

▶ **Lemma 7.** *Let G be a graph and $X \subseteq V(G)$ such that $|X| \geq 2$. Any branch decomposition of G has an edge that displays a bipartition (L, R) of $V(G)$ so that $|L \cap X| \geq |X|/3$ and $|R \cap X| \geq |X|/3$.*

Corresponding *linear branchwidth* constrains T to be a path with a pendant for each internal node. The linear f-branchwidth of G is denoted by $\mathsf{lin\text{-}f\text{-}width}(G)$. It is equivalent to view the layout given by such a tree as a permutation of $V(G)$ where the considered cuts for linear branchwidth are between vertices before and vertices after each point in the permutation.

Besides $\mathcal{F}$-branchwidth, we will also use the cut function nec_G (and its associated branchwidth) defined as follows. Given $A \subseteq V(G)$, two subsets X and Y of A are neighborhood-equivalent over A if $N(X) \setminus A = N(Y) \setminus A$, we define $\mathsf{nec}_G(A)$ as the number of equivalence relation over A.

We focus on the branchwidths proposed in [12] that arise as follows. Let $\mathcal{F}$ be an infinite family of bipartite graphs where each graph in $\mathcal{F}$ has bipartition $(A = (a_1, \ldots, a_n), B = (b_1, \ldots, b_n))$ such that for each $2n$-vertex graph in $\mathcal{F}$ and each subset L of $[n]$, the graph induced on $\{a_i, b_i | i \in L\}$ is isomorphic to a graph in $\mathcal{F}$. Given a graph G, we define the *$\mathcal{F}$-cut function* f_G of G such that $\mathsf{f}_G(X)$ is the the largest k such that a $2k$-vertex graph in $\mathcal{F}$ is isomorphic to an induced subgraph of $G[X, V(G) \setminus X]$. It is known that for any such choice of $\mathcal{F}$, $\mathcal{F}$-branchwidth is asymptotically equivalent to $\mathcal{F}'$-branchwidth where $\mathcal{F}'$ is a union of any of the following six graph classes which are called *si ph* (short for *size identifiable partner hereditary*): $\equiv$ the class of all bipartite induced matchings, $\bigtimes$ the class of all bipartite complements of induced matchings, also called *antimatchings* (or co-matchings), $\boxtimes$ the class of all complete bipartite graphs with an equal number of vertices on each side of the bipartition, $\vdots\ \vdots$ the class of all bipartite edgeless graphs with an equal number of vertices on each side of the bipartition, which we simply refer to as *bipartite edgeless graphs*, $\leqslant$ the class of all bipartite graphs with an equal number of vertices on each side of the bipartition such that the vertices on each side permit an indexing under which each vertex u on one side has an edge to a vertex v other the index of u is at most the index of v, which we refer to as *chains*, and $\diagdown\!\!\!\leqslant$ the class of all bipartite graphs with an equal number of vertices on each side of the bipartition such that the vertices on each side permit an indexing under which each vertex u on one side has an edge to a vertex v other the index of u is less than the index of v, which we refer to as *skewed chains*.

It will be useful to introduce the following notation for some bipartite graphs:

- $\vdots\ \vdots_k = ((a_1, \ldots, a_k), (b_1, \ldots, b_k), E)$, where $E = \emptyset$.
- $\boxtimes_k = ((a_1, \ldots, a_k), (b_1, \ldots, b_k), E)$, where $E = \{a_i b_j \mid i, j \in [k]\}$.
- $\equiv_k = ((a_1, \ldots, a_k), (b_1, \ldots, b_k), E)$, where $E = \{a_i b_i \mid i \in [k]\}$.
- $\bigtimes_k = ((a_1, \ldots, a_k), (b_1, \ldots, b_k), E)$, where $E = \{a_i b_j \mid i, j \in [k], i \neq j\}$.
- $\leqslant_k = ((a_1, \ldots, a_k), (b_1, \ldots, b_k), E)$, where $E = \{a_i b_j \mid 1 \leq i \leq j \leq k\}$.
- $\diagdown\!\!\!\leqslant_k = ((a_1, \ldots, a_k), (b_1, \ldots, b_k), E)$, where $E = \{a_i b_j \mid 1 \leq i < j \leq k\}$.

2.1 Co-/degeneracy, empty-width/complete-width, and subdivisions

In this subsection, we explore the relations between degeneracy [16], subdivisions, complete-width and empty-width. All the corollary of this subsection follows from the fact that the (linear) complete-width of a graph is exactly the (linear) empty-width of its complement. This will be useful to establish relationships between $\mathcal{F}$-branchwidths in the next section.

▶ **Proposition 8.** *Let G be a graph with degeneracy at most k. Then, G has linear complete-width at most k.*

▶ **Corollary 9.** *Let G be a graph with co-degeneracy at most k. Then, G has linear empty-width at most k.*

On the other hand, hypercubes have unbounded degeneracy but complete-width at most 2.

▶ **Proposition 10.** *For each $d \in \mathbb{N}$, the hypercube of dimension d has minimum degree d and linear complete-width at most two.*

▶ **Corollary 11.** *For each $d \in \mathbb{N}$, the hypercube of dimension d has minimum co-degree d and linear empty-width at most two.*

For a graph G and a positive integer q, the *q-subdivision of G* is the graph obtained from G by subdividing each edge q times.

▶ **Observation 12.** *Let G be a simple graph, and let G' be obtained from G by subdividing each edge at least once. Then, any linear ordering of G' has complete-width at most one.*

▶ **Corollary 13.** *Let G be a simple graph and let $\overline{G'}$ be the complement of a graph obtained from G by subdividing each edge at least once. Then, any linear ordering of G' has empty-width at most one.*

3 The Eleven

In this section we show that we can restrict ourselves further to eleven different $\mathcal{F}$-branchwidth measures, called *the Eleven* (shown in Figure 2) and still obtain complete classifications. We begin by restating a lemma from [12] to justify that we do not need to consider the family of skewed chains.

▶ **Lemma 14** (Lemma 4.9 in [12])**.** *Let $\mathcal{F}$ be a union of si ph classes. If $\diagup\!\!\!\!\diagup \subseteq \mathcal{F}$, then for every graph G it holds that $|\mathcal{F}\text{-bw}(G) - \mathcal{F}'\text{-bw}(G)| \leq 1$, where $\mathcal{F}' = (\mathcal{F} \setminus \diagup\!\!\!\!\diagup) \cup \boxplus$.*

The remaining five singleton-classes that we restrict our attention to from now on are each interesting in their own right. The following observation will come in useful.

▶ **Observation 15.** *Let $H = ((a_1, \ldots, a_k), (b_1, \ldots, b_k), E)$ be an ordered bipartite graph.*
1. *If $H = \equiv_k$, then H contains an edgeless bipartite graph of half-order $\lfloor k/2 \rfloor$.*
2. *If $H = \boxplus_k$ or $H = \diagup\!\!\!\!\diagup_k$, then H contains an edgeless bipartite graph of half-order $\lfloor k/2 \rfloor$.*
3. *If $H = \boxtimes_k$, or $H = \boxplus_k$, or $H = \diagup\!\!\!\!\diagup_k$, then H contains a complete bipartite graph of half-order $\lfloor k/2 \rfloor$.*

We move on to pairs, first observing that we can restrict ourselves to the case when at most one of $\vdots\ \vdots$ or $\boxtimes$ is contained in $\mathcal{F}$.

▶ **Lemma 16.** *Let $\mathcal{F}$ be a union of si ph classes such that $\vdots\ \vdots \cup \boxtimes \subseteq \mathcal{F}$. There is a computable function f such that for each graph G with $\mathcal{F}\text{-bw}(G) \leq k$, $|V(G)| < f(k)$.*

When $\vdots\ \vdots \subseteq \mathcal{F}$, the presence or absence of $\equiv$ or $\boxplus$ in $\mathcal{F}$ does not matter asymptotically either.

▶ **Lemma 17.** *Let $\mathcal{F}$ be a union of si ph classes such that $\vdots\ \vdots \subseteq \mathcal{F}$. Let $\mathcal{F}^*$ be any (possibly empty) combination of $\{\equiv, \boxplus\}$, and let $\mathcal{F}' = \mathcal{F} \cup \mathcal{F}^*$. Then, for any graph G, $\mathcal{F}\text{-bw}(G) \leq \mathcal{F}'\text{-bw}(G) \leq 2 \cdot \mathcal{F}\text{-bw}(G) + 1$.*

Similarly, we have the following.

▶ **Lemma 18.** *Let $\mathcal{F}$ be a union of si ph classes such that $\boxtimes \subseteq \mathcal{F}$. Let $\mathcal{F}^*$ be any (possibly empty) combination of $\{\boxtimes, \boxplus\}$, and let $\mathcal{F}' = \mathcal{F} \cup \mathcal{F}^*$. Then, for any graph G, $\mathcal{F}$-bw$(G) \leq \mathcal{F}'$-bw$(G) \leq 2 \cdot \mathcal{F}$-bw$(G) + 1$.*

We are now ready to define the Eleven– see Table 1 – and prove Theorem 1.

Table 1 Definitions of *the Eleven* $\mathcal{F}$-branchwidth and introduction of the notations for their associated $\mathcal{F}$-cut functions.

$\mathcal{F}$	Name	$\mathcal{F}$-cut function	$\mathcal{F}$	Name	$\mathcal{F}$-cut function
$\boxtimes$	complete-width	complete$_G$	$\boxtimes \cup \boxplus$	tree*-width	t$^\star{}_G$
$\vdots\ \vdots$	empty-width	empty$_G$	$\vdots\ \vdots \cup \boxtimes$	co-tree*-width	co-t$^\star{}_G$
$\boxplus$	mim-width	mim$_G$	$\boxplus \cup \boxtimes$	mam-width	mam$_G$
$\boxtimes$	miam-width	miam$_G$	$\boxtimes \cup \boxplus$	chim-width	chim$_G$
$\boxplus$	chain-width	chain$_G$	$\boxplus \cup \boxplus$	cham-width	cham$_G$
			$\boxplus \cup \boxtimes \cup \boxplus$	clique*-width	c$^\star{}_G$

Observe that tree*-width is asymptotically equivalent to tree-width and clique*-width is asymptotically equivalent [12, Section 4].

▶ **Theorem 1.** *For each $\mathcal{F}$-branchwidth, one of the following holds.*

- **(i)** *There is a computable function f such that for each graph G with $\mathcal{F}$-bw$(G) \leq k$, $|V(G)| \leq f(k)$.*
- **(ii)** *$\mathcal{F}$-branchwidth is asymptotically equivalent to $\mathcal{F}^*$-branchwidth, where $\mathcal{F}^*$ is one of the Eleven branchwidths (presented in Figure 2 and formally defined in Table 1).*

Next, we argue that the relationships between the Eleven as shown in Figure 2 indeed faithfully represent their hierarchy. That is, whenever two width measures are connected, then the one on top *strictly* generalizes the one the bottom, and whenever there is no connection (in terms of paths) between two, then they are incomparable. The fact that the depicted generalizations hold up follows directly from the definition and in some cases Observation 15.

For the lower bounds on width measures, we need the following canonical classes of graphs of unbounded width. For each $n \in \mathbb{N}$, we consider the following graphs on vertex set $[n] \times [n]$.

- The $n \times n$ *grid*, denoted by $\boxplus_n$, has edges $\{(i,j),(i',j')\}$ where $|i-i'| + |j-j'| = 1$.
- The *complement of the $n \times n$ grid* is denoted by $\overline{\boxplus}_n$.
- The $n \times n$ *comparability grid*, denoted by $\boxtimes_n$, has edges $\{(i,j),(i',j')\}$ where $i \leq i'$ and $j \leq j'$. For an illustration see Figure 5. Note that comparability grids appear in the study of rank-width naturally [15].

Grids have unbounded mim-width [17], which in the language of $\mathcal{F}$-branchwidth can be expressed in a more general form as follows.

▶ **Proposition 19.** *Let $\mathcal{F}$ be a union of si ph families. If $(\boxplus \cup \vdots\ \vdots) \cap \mathcal{F} \neq \emptyset$, then for each $n \in \mathbb{N}$, $\mathcal{F}$-bw$(\boxplus_n) \in \Omega(n)$.*

By complementation, we have the following consequence.

▶ **Corollary 20.** *Let $\mathcal{F}$ be a union of si ph families. If $(\boxtimes \cup \boxtimes) \cap \mathcal{F} \neq \emptyset$, then for each $n \in \mathbb{N}$, $\mathcal{F}$-bw$(\overline{\boxplus}_n) \in \Omega(n)$.*

Lastly, we show that comparability grids are a family of unbounded chain-width, while their mam-width is bounded by a constant.

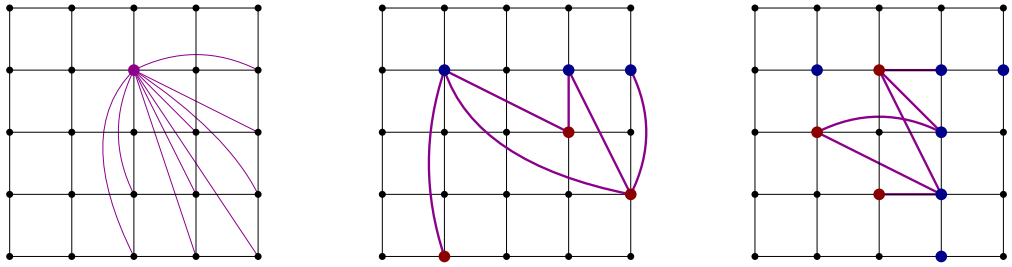


Figure 5 Illustration of comparability grids by the example of $\boxtimes_5$. On the left, the grid structure plus the additional edges incident with one highlighted vertex. In the middle and on the right, some chains that appear in cuts of a branch decomposition.

Table 2 Each cell shows families of graphs whose width is bounded by a constant by the width measure whose cut-function indexes the row and unbounded by the width measure whose cut-function indexes the column. Here, $\boxplus$ stands for the family of grids, $\overline{\boxplus}$ for the complements of grids, and $\boxplus$ for the comparability grids. For instance, the entry $\boxplus$ in row "chain" and column "mim" indicates that the family of grids has constant chain-width but unbounded mim-width. Furthermore, K_n denotes the family of complete graphs, and $\emptyset_n$ the family of edgeless graphs.

	chain	mim	miam	chim	mam	cham	empty	complete	c*/t*/co-t*
chain		$\boxplus$	$\overline{\boxplus}$	$\boxplus$	$\boxplus$ $\overline{\boxplus}$	$\overline{\boxplus}$	$\boxtimes$	$\overline{\boxplus}$	$\boxtimes$ $\overline{\boxplus}$
mim	$\boxtimes$		$\overline{\boxplus}$	$\boxtimes$	$\overline{\boxplus}$	$\boxtimes$ $\overline{\boxplus}$	$\boxtimes$	$\overline{\boxplus}$	$\boxplus$ $\overline{\boxtimes}$
miam	$\boxplus$	$\boxtimes$		$\boxplus$ $\boxtimes$	$\boxtimes$	$\boxplus$	$\boxtimes$ $\boxplus$	$\boxplus$	$\boxplus$ $\boxplus$
chim			$\overline{\boxplus}$		$\overline{\boxtimes}$	$\overline{\boxtimes}$	$\emptyset_n$	$\overline{\boxtimes}$	$\overline{\boxtimes}$
mam	$\boxtimes$			$\boxtimes$		$\boxplus$	$\boxplus$	$\boxplus$	$\boxplus$
cham		$\boxplus$		$\boxplus$	$\boxplus$		$\boxplus$	K_n	$\boxplus$
empty			$\overline{\boxplus}$		$\overline{\boxplus}$	$\overline{\boxplus}$		$\overline{\boxplus}$	$\overline{\boxplus}$
complete		$\boxplus$		$\boxplus$	$\boxplus$		$\boxtimes$		$\boxplus$
c*							$\emptyset_n$	K_n	/ K_n / $\emptyset_n$
t*							$\emptyset_n$		/ / $\emptyset_n$
co-t*								K_n	/ K_n /

▶ **Proposition 21.** *For each $n \in \mathbb{N}$,* lin-mam-width($\boxtimes_n$) $= 1$ *and* chain-width($\boxtimes_n$) $\geq n/4$.

Next, we observe upper bounds on $\mathcal{F}$-branchwidth of grids, their complements, and comparability grids.

▶ **Proposition 22.** *Let $\mathcal{F}$ be a union of si ph families. If $\mathcal{F} \subseteq \boxplus \cup \boxtimes \cup \overline{\boxplus}$, then, for each $n \in \mathbb{N}$, $\mathcal{F}$-bw($\boxplus_n$) $\in \mathcal{O}(1)$.*

By complementation and Corollary 9, we get the following.

▶ **Corollary 23.** *Let $\mathcal{F}$ be a union of si ph families. If $\mathcal{F} \subseteq \vdots \cup \equiv \cup \overline{\boxtimes}$, then, for each $n \in \mathbb{N}$, $\mathcal{F}$-bw($\overline{\boxplus}_n$) $\in \mathcal{O}(1)$.*

Using Propositions 19, 21, and 22 and Corollaries 20 and 23, we can argue almost all of the desired relationships, see Table 2. The remainder is discussed in the full version which concludes the proof of Theorem 2.

4 FO logic

It is known that FO Model Checking is AW[∗]-hard on interval graphs [14]. Furthermore, interval graphs are known to have linear mim-width 1, witnessed by a linear order that can be computed in polynomial time [2]. Next, we can observe that for any graph G and $A \subseteq V(G)$,

$$\mathsf{mim}_G(A) \leq 1 \Leftrightarrow \mathsf{miam}_G(A) \leq 1.$$

This is because a matching of half-order 2 and an anti-matching of half-order 2 are isomorphic. Therefore, we have the following consequence.

▶ **Corollary 24** (of [2, 14]). FO-*Model Checking parameterized by the length of the formula is* AW[∗]*-hard on graphs of linear mam-width one, even if a width-*1 *linear order of the input graph is given.*

Due to Corollary 24, the only remaining cases where we could hope for FPT-algorithms for FO-Model Checking are complete-width and empty-width. However, we show by two trivial reductions that the problem remains W[1]-hard parameterized by formula length on graphs of width 1 as well. Note that the following reduction is folklore, but we state it here for completeness.

▶ **Proposition 25.** FO-*Model Checking parameterized by the length of the formula is* AW[∗]*-hard on graphs of linear complete-width one, even if a width-*1 *linear order of the input graph is given.*

Let (G, ϕ) be an instance of FO-Model Checking such that G has linear complete-width one. Observe that $\overline{G}$ has linear empty-width one. We replace each occurrence of $x_i \sim x_j$ in ϕ with $\neg x_i \sim x_j$ to obtain a formula ϕ'. Then, $\overline{G} \models \phi'$ if and only if $G \models \phi$ and we obtain the following.

▶ **Corollary 26.** FO-*Model Checking parameterized by the length of the formula is* AW[∗]*-hard on graphs of linear empty-width one, even if a width-*1 *linear order of the input graph is given.*

5 Bounds on nec in terms of $\mathcal{F}$-branchwidth

The run time of the algorithm for A&C DN Model Checking from [3] is polynomial in the number of equivalence classes of the neighborhood equivalence relation. Therefore, any bound on this quantity immediately translates to an efficient algorithm for A&C DN Model Checking.

5.1 Upper Bounds

We first give some upper bounds. For tree-width, co-tree-width, and clique-width, single-exponential upper bounds on nec are known (e.g., [3]). Pipelined through the arguments from [12], this gives bounds for the equivalent $\mathcal{F}$-branchwidth parameters, but they might end up double-exponential due to the Ramsey arguments in [12]. Here, we give single-exponential upper bounds in terms of these width parameters by giving more direct arguments.

▶ **Lemma 27.** *Let G be a graph and $A \subseteq V(G)$ such that $\mathsf{t}^\star_G(A) \leq k$. Let M be a matching in $G[A, \overline{A}]$. Then, $|M| < (k+1)^{k+2}$.*

▶ **Lemma 28.** *Let G be a graph, $A \subseteq V(G)$ with $\mathsf{t}^\star{}_G(A) \leq k$. There is a set $\Omega \subseteq A$ of size at most $2^{\mathcal{O}(k^2 \log k)}$ such that $\{N(v) \cap \overline{A} \mid v \in A\} = \{N(v) \cap \overline{A} \mid v \in \Omega\}$.*

▶ **Proposition 29.** *Let G be a graph and $A \subseteq V(G)$ with $\mathsf{t}^\star{}_G(A) \leq k$. Then, $\mathsf{nec}_G(A) \in 2^{\mathcal{O}(k^3 \log k)}$.*

▶ **Proposition 30.** *Let G be a graph and $A \subseteq V(G)$ with $\mathsf{co\text{-}t}^\star{}_G(A) \leq k$. Then, $\mathsf{nec}_G(A) \leq 2^{\mathcal{O}(k^3 \log k)}$.*

5.2 Lower bounds

We first exhibit some cuts with high nec in terms of the respective width parameters. Later, we then prove that there are families of graphs where each of their branch decompositions has to have a cut with large nec.

▶ **Observation 31.** *Let G be obtained by taking the disjoint union of an antimatching with $2n$ vertices and a matching with $2k$ vertices and completely connecting the sides A and $\overline{A}$ of the resulting bipartite graph. It holds that $\mathsf{empty\text{-}width}(G) \leq k$, and $\mathsf{nec}_G(A) \geq 2^k \cdot n$.*

For the following observation, take the disjoint union of k chains of half-order k.

▶ **Observation 32.** *There is a bipartite graph $G = (A, B, E)$ with $\mathsf{t}^\star{}_G(A) = k$ and $\mathsf{nec}_G(A) \in 2^{\Omega(k \log k)}$.*

▶ **Observation 33.** *Let $H = H_{=\!=}^n$ with bipartition (A, B). Then, $\mathsf{nec}_H(A) = 2^n$.*

▶ **Observation 34.** *For each $n \in \mathbb{N}$ there is a graph G on $2n$ vertices and $A \subseteq V(G)$ such that $\mathsf{complete}_G(A) = 1$ and $\mathsf{nec}_G(A) = 2^n$.*

▶ **Proposition 35.** *There is an infinite family of graphs G with $\mathsf{complete\text{-}width}(G) = 1$ and such that each branch decomposition of G induces a cut $(A, \overline{A})$ with $\mathsf{nec}_G(A) \in 2^{\Omega(\sqrt{|V(G)|})}$.*

▶ **Proposition 36.** *For each $n, k \in \mathbb{N}$ with $n \geq k$ there is a bipartite graph $G = (A, B, E)$ on $2n$ vertices such that $\mathsf{mam}_G(A) = k$ and $\mathsf{nec}_G(A) = \lfloor n/k \rfloor^k$.*

By taking a disjoint union of k antimatchings of half-order n/k in the previous proof, we get the following corollary.

▶ **Corollary 37.** *For each $n, k \in \mathbb{N}$ with $n \geq k$ there is a bipartite graph $G = (A, B, E)$ on $2n$ vertices such that $\mathsf{chim}_G(A) = k$ and $\mathsf{nec}_G(A) = \lfloor n/k \rfloor^k$.*

In the following, we provide several lower bounds on the nec-width of graphs of low chim-width, mam-width, tree*-width, or co-tree*-width. The proofs strategy are based on the proof of [5, Theorem 1.3] stating that there exists arbitrary large graphs of rank-width k with Boolean-width $\Omega(k^2)$.

We start by proving a lower bound on nec-width in terms of chim-width. The following construction is based on the reduction used to prove Theorem 5 for chim-width.

▶ **Lemma 38.** *For each $k \in \mathbb{N}$ with $k > 0$, there exist arbitrary large graphs G of linear chim-width at most $2k$ and with $\mathsf{nec\text{-}width}(G) \in (n/(k^2 \log n))^{\Omega(k)}$.*

It is easy to obtain the same result as Lemma 38 for mam-width by replacing anti-matchings by chains in the construction of the graph G.

▶ **Lemma 39.** *For each $k \in \mathbb{N}$ with $k > 0$, there exist arbitrary large graphs G of linear mam-width at most $k + 1$ and with $\mathsf{nec\text{-}width}(G) \in (n/(k^2 \log n))^{\Omega(k)}$.*

▶ **Lemma 40.** *For each $k \in \mathbb{N}$ with $k > 0$, there exist arbitrary large graphs G of linear tree*-width at most $2k$ and with* nec-width$(G) \in k^{\Omega(k)}$.

▶ **Lemma 41.** *For each $n, k \in \mathbb{N}$ with n sufficiently large, there is a graph $G = (A, B, E)$ on $2n$ vertices such that* empty-width$(G) \leq k$ *and with* nec-width$(G) \in |V(G)|^{\Omega(k)}$.

6 Independent Set parameterized by $\mathcal{F}$-Branchwidth

In this section, we discuss the complexity of Independent Set parameterized by any $\mathcal{F}$-branchwidth measure. On top of that, we strengthen known hardness results for Independent Set and Dominating Set (full version) on graphs of bounded mim-width under the ETH. Concretely, we show the following.

▶ **Theorem 5.** *Unless the Exponential Time Hypothesis fails, neither Independent Set nor Dominating Set can be solved in $f(k)n^{o(k)}$ time for any computable function f, where k is either the chim-width, the mam-width, or the mim-width of a given decomposition of the n-vertex input graph. Moreover, both problems parameterized by either the chim-width or the mam-width are* W[1]*-hard.*

These lower bounds are tight because the algorithm – based on the d-neighbor equivalence – from [7] solves Independent Set and Dominating Set in time $n^{O(w)}$ where w is the mim-width of a given decomposition which is always smaller than its chim-width and mam-width.

It is well-known that Independent Set is FPT parameterized by clique-width, and therefore clique*-width, so for all $\mathcal{F}$-branchwidth measures that are at most as general as clique*-width, Independent Set is FPT.

6.1 Independent Set on graphs of bounded complete-width and chain-width

Since Independent Set is NP-complete on cubic graphs, it is paraNP-hard when parameterized by any width measure in whose obstructions the degree increases proportionally to the size. This is the case for complete bipartite graphs, anti-matchings, and chains.

▶ **Observation 42.** *Independent Set is* NP*-hard on graphs with complete-width at most* 3.

Nevertheless, the previous result does not rule out efficient parameterized algorithms for Independent Set when the solution size is an additional part of the parameter. Note that here, the bounded-degree argument does not apply, since Independent Set parameterized by solution size plus maximum degree can easily be shown to be FPT [10]. However, we show that Independent Set remains W[1]-hard parameterized by solution size on graphs of (the more general) chain-width at most 3. (The same reduction in fact shows W[1]-hardness parameterized by solution size plus chim-width.)

This can be shown by observing that the graphs obtained in the reduction that proved W[1]-hardness for Independent Set parameterized by mim-width plus solution size [13] in fact have a linear order whose mim-width is bounded by a function of the parameter and whose chim-width is at most three.

▶ **Corollary 43.** *Independent Set is W[1]-hard parameterized by*

- *the solution size (plus the linear mim-width) on graphs of linear chain-width* 3*, or*
- *the solution size plus the linear chim-width*

of the input graph, even if a linear order of bounded width is provided with the input.

6.2 Independent Set on graphs of bounded chim-width

To prove Theorem 5 for chim-width and Independent Set, we provide a reduction from Multicolored Independent Set in general graphs to Independent Set in graphs of low chim-width. Briefly, an instance of Multicolored Independent Set consists of a graph whose vertex set is the disjoint union of k sets $V_1, \ldots, V_k$ and the problem asks for a *multicolored* independent set which is an independent set with exactly one vertex in each V_i. Let H a graph with $V(H) = V_1 \uplus \cdots \uplus V_k$ be an instance of Multicolored Independent Set. We denote by N the number of vertices of H and by M the number of edges of H. In the following, we show how to construct in polynomial time a graph G with at most $n = NM$ vertices and a linear order of chim-width at most $2k$ such that H admits a multicolored independent set iff G admits an independent set of size at least kM. This is sufficient to prove Theorem 5 for chim-width. Indeed, with a $f(w)n^{o(w)}$ time algorithm for Independent Set for some computable function f, we would be able to solve Multicolored Independent Set in time $f(k)N^{o(k)}$ which is not possible under the ETH [10]. Futhermore, the W[1]-hardness of Independent Set parameterized by chim-width follows from the W[1]-hardness of Multicolored Independent Set parameterized by k [10].

Intuitively, we start by creating a *selection gadget* S_{uv} for each edge uv of H such that the maximum independent sets of $G[S_{uv}]$ are bijectively mapped to the *potential solution* of H – set of vertices with exactly one vertex in each V_i – which do not contains both u and v. Then we glue the selection gadgets together in such a way that the maximum independent sets in two glued gadgets are non-adjacent if and only if they are associated with the same potential solution of H. See Figure 6 for an illustration of the following construction.

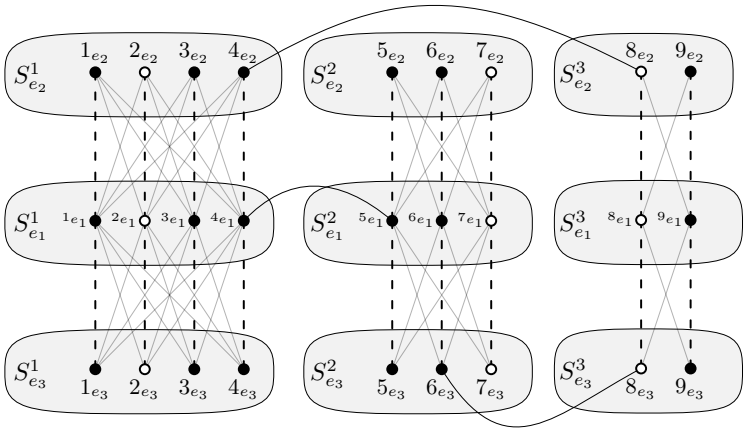


Figure 6 Illustration of the reduction for chim-width with $k = 3$, $V_1 = \{1, 2, 3, 4\}$, $V_2 = \{5, 6, 7\}$ and $V_3 = \{8, 9\}$. Here $M = 3$ and the edges of H consists of $e_1 = 45$, $e_2 = 48$ and $e_3 = 68$. Dashed lines represent the non-edges of the anti-matchings between the selection gadgets. To improve the legibility, we omit the edges of the cliques $S^j_{e_i}$: $i, j \in [3]$. The independent set I consisting of the white filled vertices have size kM and $I_H = \{2, 7, 8\}$ is a multicolored independent set of H.

Selection gadgets. For each edge $e \in E(H)$, we create the selection gadget S_e which consists of the disjoint union of k cliques $S^1_e, \ldots, S^k_e$ such that for each $j \in [k]$, $S^i_e := \{x_e \mid x \in V_i\}$. We finish the construction of S_e by adding the edge between u_e and v_e where u and v are the endpoints of e. Given a set $X \subseteq S_e$, we denote by X_H the set $\{x \in V(H) \mid x_e \in X\}$.

▶ **Observation 44.** *For every edge e of H, X is a maximum independent set I of $G[S_e]$ if and only if X_H has exactly one vertex in each V_i and does not contain both endpoints of e.*

Propagation. The final step of this construction is to add edges between the selection gadgets such that if H admits a multicolored independent sets then every maximum independent set I of G intersects the selections gadgets in the same way, i.e. for every pair of edges (e_1, e_2) of H, we have $(I \cap S_{e_1})_H = (I \cap S_{e_2})_H$. We achieve this without blowing up the chim-width of G as follows. We fix an edge e_1 of H and for every edge $e \neq e_1$ of H and $i \in [k]$, we first add all the edges between $S^i_{e_1}$ and S^i_e and then for each vertex $x \in V_i$, we remove the edge $x_{e_1}x_e$. Consequently, the bipartite graph $G[S^i_{e_1}S^i_e]$ is an anti-matching of half-order $|V_i|$. As $S^i_{e_1}$ and S^i_e are cliques in G, the maximum independent sets of $G[S^i_{e_1} \cup S^i_e]$ are exactly the pairs $\{x_{e_1}, x_e\}$ with $x \in V_i$. This leads to the following observation.

▶ **Observation 45.** *For every edge $e \neq e_1$ of H and subsets X, Y of size k of respectively S_{e_1} and S_e, there is no edge between X and Y in G if and only if we have $X_H = Y_H$.*

We deduce from Observations 44 and 45 the correctness of our reduction.

▶ **Lemma 46.** *The graph G admits an independent set of size at least kM if and only if H admits a multicolored independent set.*

Finally, we prove that we can construct in polynomial time a linear order of G of chim-width at most $2k + 2$. The following proof relies on the fact that induced matchings and chains in anti-matchings have size at most 2, which implies that the chim-width of the cut $(S_{e_1}, V(G) \setminus S_{e_1})$ is at most $2k$.

▶ **Lemma 47.** *We can compute in polynomial time a linear order of G of chim-width at most $2k + 2$.*

6.3 Independent Set on graphs of bounded mam-width

The proof of Theorem 5 for mam-width is similar to the one for chim-width, we start from the same instance of Multicolored Independent Set and construct a graph G. Compared to chim-width, the following construction is a bit more involved and uses skewed chains instead of anti-matchings to glue the selection gadgets together. See Figure 7 for an illustration of the following construction.

Selection gadgets. We reuse the selection gadget $S_{e_1}, S_{e_2} \ldots, S_{e_M}$ from the reduction for chim-width where $e_1, \ldots, e_M$ are the edges of H.

Propagation. First, we add anti-matchings between S_{e_1} and S_{e_2} just as we did in the reduction of chim-width. That is for every $i \in [k]$, we do the following. We add all the edges between $S^i_{e_1}$ and $S^i_{e_2}$. Finally, for every vertex $v \in V_i$, we remove the edge $v_{e_1}v_{e_2}$. Note that $G[S^i_{e_1}, S^i_{e_2}]$ is big anti-matching, but this does not blow-up the mam-width. Indeed, we can insert in the permutation of Lemma 47 each vertex of $S^i_{e_2}$ next to its non-neighbor in $S^i_{e_1}$, this way every cut of the resulting permutation contains at most one non-edge of $G[S^i_{e_1}, S^i_{e_2}]$ and the mam-width stays low. Note that Observation 45 holds for $e = e_2$.

Finally, we add skewed chains between $S_{e_1} \cup S_{e_2}$ and the other selection gadgets, by doing the following. We take a arbitrary strict linear order $<_H$ on the vertices of H such that for each $i, j \in [k]$ with $i < j$, we have $u <_H v$ for each $u \in V_i$ and $v \in V_j$. For every $i \in [M]$ with $i > 2$, $j \in [k]$, and $x, y \in V_j$ such that $x <_H y$, we make x_{e_1} adjacent to y_{e_i} and y_{e_2} adjacent to x_{e_i}.

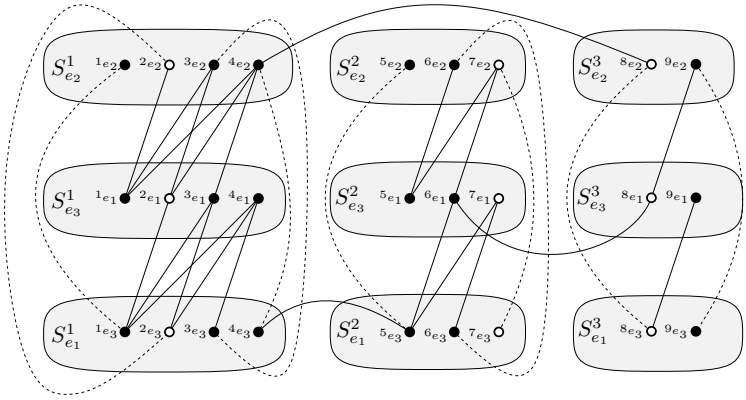


Figure 7 Illustration of the reduction for mam-width with $k = 3$, $V_1 = \{1, 2, 3, 4\}$, $V_2 = \{5, 6, 7\}$ and $V_3 = \{8, 9\}$. Here $M = 3$ and the edges of H consists of $e_1 = 45$, $e_2 = 48$ and $e_3 = 68$. To improve the legibility, we omit the edges of the cliques $S^j_{e_i}$: $i, j \in [3]$. Moreover, we represent only the non-edges (with dashed lines) of the anti-matchings between S_{e_1} and S_{e_2}. The independent set I consisting of the white filled vertices have size kM and $I_H = \{2, 7, 8\}$ is a multicolored independent set of H.

Observe that, for every $i \in [M]$, $j \in [t]$ and $X \in \{R, S\}$, the graphs $G[S^j_{e_1}, S^j_{e_i}]$ and $G[S^j_{e_2}, S^j_{e_i}]$ are skewed chain graphs of half-order $|V_i|$. Moreover, the maximum independent sets of $G[S^j_{e_1} \cup S^j_{e_2} \cup S^j_{e_i}]$ are exactly the sets $\{v_{e_1}, v_{e_2}, v_{e_i}\}$ with $v \in V_i$.

▶ **Observation 48.** *For every $i \in [M]$ and subsets X, Y and Z of respectively S_{e_1}, S_{e_2} and S_{e_i}, each of size k, there is no edge between any two of these subsets in G if and only if $X_H = Y_H = Z_H$.*

References

1 Brage I. K. Bakkane and Lars Jaffke. On the hardness of generalized domination problems parameterized by mim-width. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 3:1–3:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.3.

2 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theor. Comput. Sci.*, 511:54–65, 2013. doi:10.1016/j.tcs.2013.01.011.

3 Benjamin Bergougnoux, Jan Dreier, and Lars Jaffke. A logic-based algorithmic meta-theorem for mim-width. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 3282–3304. SIAM, 2023. doi:10.1137/1.9781611977554.ch125.

4 Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d-neighbor equivalence: Acyclicity and connectivity constraints. *SIAM J. Discret. Math.*, 35(3):1881–1926, 2021. doi:10.1137/20M1350571.

5 Benjamin Bergougnoux, Tuukka Korhonen, and Jesper Nederlof. Tight lower bounds for problems parameterized by rank-width. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.11.

6 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

7 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoret. Comput. Sci.*, 511:66–76, 2013. doi:10.1016/j.tcs.2013.01.009.

8 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

9 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discret. Appl. Math.*, 108(1-2):23–52, 2001. doi:10.1016/S0166-218X(00)00221-3.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

12 Eduard Eiben, Robert Ganian, Thekla Hamm, Lars Jaffke, and O-joung Kwon. A unifying framework for characterizing and computing width measures. In Mark Braverman, editor, *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 63:1–63:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2022.63.

13 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. *Algorithmica*, 82(9):2432–2473, 2020. doi:10.1007/s00453-020-00692-9.

14 Robert Ganian, Petr Hlinený, Daniel Král, Jan Obdrzálek, Jarett Schwartz, and Jakub Teska. FO model checking of interval graphs. *Log. Methods Comput. Sci.*, 11(4), 2015. doi:10.2168/LMCS-11(4:11)2015.

15 Jim Geelen, O-joung Kwon, Rose McCarty, and Paul Wollan. The grid theorem for vertex-minors. *J. Comb. Theory B*, 158(Part):93–116, 2023. doi:10.1016/J.JCTB.2020.08.004.

16 Don R. Lick and Arthur T. White. k-degenerate graphs. *Canadian Journal of Mathematics*, 22(5):1082–1096, 1970. doi:10.4153/CJM-1970-125-1.

17 Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, Norway, 2012.

Weighted Matching in a Poly-Streaming Model

Ahammed Ullah
Purdue University, West Lafayette, IN, USA

S M Ferdous
Pacific Northwest National Laboratory, Richland, WA, USA

Alex Pothen
Purdue University, West Lafayette, IN, USA

Abstract

We introduce the *poly-streaming model*, a generalization of streaming models of computation in which k processors process k data streams containing a total of N items. The algorithm is allowed $\mathcal{O}(f(k) \cdot M_1)$ space, where M_1 is either $o(N)$ or the space bound for a sequential streaming algorithm. Processors may communicate as needed. Algorithms are assessed by the number of passes, per-item processing time, total runtime, space usage, communication cost, and solution quality.

We design a *single-pass* algorithm in this model for approximating the *maximum weight matching* (MWM) problem. Given k edge streams and a parameter $\varepsilon > 0$, the algorithm computes a $(2+\varepsilon)$-approximate MWM. We analyze its performance in a shared-memory parallel setting: for any constant $\varepsilon > 0$, it runs in time $\widetilde{\mathcal{O}}(L_{max} + n)$, where n is the number of vertices and L_{max} is the maximum stream length. It supports $\mathcal{O}(1)$ per-edge processing time using $\widetilde{\mathcal{O}}(k \cdot n)$ space. We further generalize the design to hierarchical architectures, in which k processors are partitioned into r groups, each with its own shared local memory. The total intergroup communication is $\widetilde{\mathcal{O}}(r \cdot n)$ bits, while all other performance guarantees are preserved.

We evaluate the algorithm on a shared-memory system using graphs with trillions of edges. It achieves substantial speedups as k increases and produces matchings with weights significantly exceeding the theoretical guarantee. On our largest test graph, it reduces runtime by nearly two orders of magnitude and memory usage by five orders of magnitude compared to an offline algorithm.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Approximation algorithms

Keywords and phrases Streaming Algorithms, Matchings, Graphs, Parallel Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.17

Related Version *Full Version*: `https://arxiv.org/abs/2507.14114`

Funding *Ahammed Ullah*: Research supported by grant SC-0022260 of the Advanced Scientific Computing Research program of the U. S. Department of Energy.
S M Ferdous: Laboratory Directed Research and Development Program at PNNL.
Alex Pothen: Research supported by grant SC-0022260 of the Advanced Scientific Computing Research program of the U. S. Department of Energy.

1 Introduction

Data-intensive computations arise in data science, machine learning, and science and engineering disciplines. These datasets are often massive, generated dynamically, and, when stored, kept in distributed formats on disks, making them amenable to processing as multiple data streams. The modular feature of these datasets can be exploited by streaming algorithms designed for tightly-coupled shared-memory and distributed-memory multiprocessors to efficiently solve large problem instances that offline algorithms cannot handle due to their high memory requirements. However, the design of parallel algorithms that process multiple data streams concurrently has not yet received much attention.


© Ahammed Ullah, S M Ferdous, and Alex Pothen;
licensed under Creative Commons License CC-BY 4.0



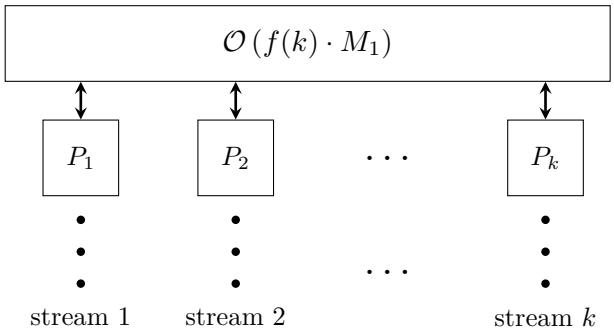


Figure 1 A schematic diagram of the *poly-streaming model* for shared-memory parallel computers. Processors $\{P_l\}_{l \in [k]}$ have access to $\mathcal{O}(f(k) \cdot M_1)$ memory collectively, depicted with the rectangle connected to the processors.

Current multicore shared-memory processors consist of up to a few hundred cores, organized hierarchically to share caches and memory controllers. These cores compute in parallel to achieve speedups over serial execution. With multiple memory controllers, I/O operations can also proceed in parallel, and this feature can be used to process multiple data streams concurrently. These I/O capabilities and the limitations of offline algorithms motivate a model of computation, illustrated in Figure 1 and discussed next.

The streaming model of computation allows $o(N)$ space for a data stream of size N [2, 13]. For graphs, the *semi-streaming model* permits $\mathcal{O}(n \cdot \text{polylog } n)$ space for a graph with n vertices and an edge stream of arbitrary length [8]. Building on these space-constrained models, we introduce the *poly-streaming model*. The key aspects of our model are as follows.

We consider k data streams that collectively contain N items. An algorithm has access to k (abstract) processors, and is allowed $\mathcal{O}(f(k) \cdot M_1)$ total space, where M_1 is either $o(N)$ or the space permitted to a single-stream algorithm. In each pass, each stream is assigned to one of the processors, and each processor independently reads one item at a time from its stream and processes it. Processors may communicate as needed, either via shared or remote memory access. Algorithms are assessed on several metrics: space complexity, number of passes, per-item processing time, total runtime, communication cost, and solution quality.

In the poly-streaming model, we address the problem of approximating a *maximum weight matching* (MWM) in an edge-weighted graph, where the goal is to find a set of vertex-disjoint edges with maximum total weight. We design an algorithm for approximating an MWM when the graph is presented as multiple edge streams. Our design builds on the algorithm of [20] and adds support for handling multiple streams concurrently. We also generalize the design to NUMA (non-uniform memory access) multiprocessor architectures.

We summarize our contributions to the MWM problem as follows. Let L_{max} and L_{min} denote the maximum and minimum lengths of the input streams, respectively, and let n denote the number of vertices in a graph G. For any realization of the CREW PRAM model (such as in Figure 1), we have the following result.

▶ **Theorem 1.** *For any constant $\varepsilon > 0$, there exists a* single-pass *poly-streaming algorithm for the maximum weight matching problem that achieves a $(2+\varepsilon)$-approximation. It admits a CREW PRAM implementation with $\widetilde{\mathcal{O}}(L_{max} + n)$ runtime.*[1] *If $L_{min} = \Omega(n)$, the algorithm*

[1] $\widetilde{\mathcal{O}}(\cdot)$ hides polylogarithmic factors.

achieves $\mathcal{O}(\log n)$ *amortized per-edge processing time using* $\widetilde{\mathcal{O}}(k+n)$ *space. For arbitrarily balanced streams, it uses either* $\widetilde{\mathcal{O}}(k+n)$ *space and* $\widetilde{\mathcal{O}}(n)$ *per-edge processing time, or* $\widetilde{\mathcal{O}}(k \cdot n)$ *space and* $\mathcal{O}(1)$ *per-edge processing time.*

In NUMA architectures, memory access costs depend on a processor's proximity to the target memory. We generalize the algorithm in Theorem 1 to account for these cost differences. In particular, we show that when k processors are partitioned into r groups, each with its own shared local memory, the total number of global memory accesses across all groups is $\widetilde{\mathcal{O}}(r \cdot n)$. This generalization preserves all other performance guarantees from Theorem 1, except that the $\widetilde{\mathcal{O}}(k+n)$ space bound becomes $\widetilde{\mathcal{O}}(k+r \cdot n)$. These results are formalized in Theorem 18 in Section 4. This design gives a memory-efficient algorithm for the NUMA shared memory multiprocessors, on which we report empirical results.

We have evaluated our algorithm on a NUMA machine using graphs with billions to trillions of edges. For most of these graphs, our algorithm uses space that is orders of magnitude smaller than that required by offline algorithms. For example, storing the largest graph in our evaluation would require more than 91,600 GB (≈ 90 TB), whereas our algorithm used less than 1 GB. Offline matching algorithms typically require even more memory to accommodate their auxiliary data structures.

We employ approximate dual variables that correspond to a linear programming relaxation of MWM to obtain *a posteriori* upper bounds on the weights of optimal matchings. These bounds allow us to compare the weight of a matching produced by our algorithm with the optimal weight. Thus, we show that our algorithm produces matchings whose weights significantly exceed the approximation guarantee.

For $k = 128$, our algorithm achieves runtime speedups of 16–83 across all graphs in our evaluation, on a NUMA machine with only 8 memory controllers. This is significant scaling for a poly-streaming algorithm, given that 8 memory controllers are not sufficient to serve the concurrent and random access requests of 128 processors without delays. Nevertheless, these speedups demonstrate the effectiveness of our design, which accounts for a processor's proximity to the target memory. A metric less influenced by memory latency suggests that the algorithm would achieve even better speedups on architectures with more efficient memory access.

Note that Theorem 1 and Theorem 18 both guarantee $\widetilde{\mathcal{O}}(L_{max}+n)$ runtime. For $L_{max} = \Omega(n)$, this is tight up to polylogarithmic factors. However, by using $\widetilde{\mathcal{O}}(k \cdot n)$ space and $\mathcal{O}(1)$ per-edge processing time, we can even achieve $\widetilde{\mathcal{O}}(L_{max}+n/k)$ runtime, which becomes polylogarithmic for large values of k (see the arXiv version).

Organization. Section 2 introduces necessary background. Section 3 presents the design and analyses of our algorithm in Theorem 1. In Section 4, we extend the design to NUMA architectures. Section 5 summarizes the evaluation results. We conclude in Section 6 with a discussion of future research directions.

2 Preliminaries

For a graph $G = (V, E)$, let $n := |V|$ and $m := |E|$ denote the number of vertices and edges, respectively. We denote an edge $e := \{u, v\}$ by the unordered pair of its endpoints. For a weighted graph, let w_e denote the weight of edge e, and for any subset $A \subseteq E$, define $w(A) := \sum_{e \in A} w_e$. For $\ell \in [k]$, let E^ℓ be the set of edges received in the ℓth stream. Define $L_{max} := \max_{\ell \in [k]} |E^\ell|$ and $L_{min} := \min_{\ell \in [k]} |E^\ell|$.

Primal LP

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} w_e x_e \\ \text{subject to} \quad & \sum_{e \in \delta(u)} x_e \leq 1, \quad \text{for all } u \in V \\ & x_e \geq 0, \quad \text{for all } e \in E \end{aligned}$$

Dual LP

$$\begin{aligned} \text{minimize} \quad & \sum_{u \in V} y_u \\ \text{subject to} \quad & \sum_{u \in e} y_u \geq w_e, \quad \text{for all } e \in E \\ & y_u \geq 0, \quad \text{for all } u \in V \end{aligned}$$

Figure 2 The linear programming (LP) relaxations of the MWM problem and its dual.

A *matching* $\mathcal{M}$ in a graph is a set of edges that do not share endpoints. A *maximum weight matching* (MWM) $\mathcal{M}^*$ is a matching with maximum total weight; that is, $w(\mathcal{M}^*) \geq w(\mathcal{M})$ for all matchings $\mathcal{M} \subseteq E$.

A ρ-approximation algorithm computes a solution whose value is within a factor ρ of optimal. The factor ρ is called the (worst-case) *approximation ratio*. We assume $\rho \geq 1$ for both maximization and minimization problems. Thus, for maximization, a ρ-approximation guarantees a solution whose value is at least $\frac{1}{\rho}$ times the optimal.

We use the linear programming (LP) relaxation of the MWM problem, and its dual, shown in Figure 2. In the primal LP, each variable x_e is 1 if edge e is in the matching and 0 otherwise. Each y_u is a dual variable, and $\delta(u)$ denotes the set of edges incident on a vertex u. Let $\{x_e\}_{e \in E}$ and $\{y_u\}_{u \in V}$ be feasible solutions to the primal and dual LPs, respectively. By weak LP duality, we have $\sum_{e \in E} w_e x_e \leq \sum_{u \in V} y_u$. If $\{x_e\}_{e \in E}$ is an optimal solution to the primal LP, then $w(\mathcal{M}^*) \leq \sum_{e \in E} w_e x_e \leq \sum_{u \in V} y_u$. The first inequality holds because the primal LP is a relaxation of the MWM problem.

3 Algorithms for Uniform Memory Access Cost

In this section, we present the design and analyses of our algorithm in Theorem 1 that assumes a uniform memory access cost.

3.1 The Algorithm

Several semi-streaming algorithms have been designed for the MWM problem [3, 4, 6, 8, 10, 11, 18, 20, 23] (see the arXiv version for brief descriptions of these algorithms). In this paper, we focus exclusively on the single-pass setting in the *poly-streaming* model. Our starting point is the algorithm of Paz and Schwartzman [20], which computes a $(2+\varepsilon)$-approximation of MWM. This is currently the best known guarantee in the single-pass setting under arbitrary or adversarial ordering of edges.[2] We extend a primal-dual analysis by Ghaffari and Wajc [11] to analyze our algorithm.

The algorithm of Paz and Schwartzman [20] proceeds as follows. Initialize an empty stack S and set $\alpha_u = 0$ for each vertex $u \in V$. For each edge $e = \{u, v\}$ in the edge stream, skip e if $w_e < (1+\varepsilon)(\alpha_u + \alpha_v)$. Otherwise, compute $g_e = w_e - (\alpha_u + \alpha_v)$, push e onto the stack S, and increase both α_u and α_v by g_e. After processing all edges, compute a matching $\mathcal{M}$ greedily by popping edges from S.

[2] No single-pass algorithm can achieve an approximation ratio better than $1 + \ln 2 \approx 1.7$; see [15].

PS-MWM(V, ℓ, ε)**:**

/* each processor executes this algorithm concurrently */
1. In parallel initialize $lock_u$, and set α_u and $mark_u$ to 0 for all $u \in V$
   /* processor ℓ initializes or sets $\Theta(n/k)$ locks/variables */
2. $S^\ell \leftarrow \varnothing$ /* initialize an empty stack */
3. for each edge $e = \{u, v\}$ in ℓth stream do
   a. Process-Edge(e, S^ℓ, ε)
4. wait for all processors to complete execution of Step 3 /* a barrier */
5. $\mathcal{M}^\ell \leftarrow$ Process-Stack(S^ℓ)
6. return $\mathcal{M}^\ell$

Figure 3 A poly-streaming matching algorithm.

Note that for each edge pushed onto the stack, the increment $g_e = w_e - (\alpha_u + \alpha_v)$ satisfies $g_e \geq \varepsilon(\alpha_u + \alpha_v)$. This ensures that both α_u and α_v increase by a factor of $1 + \varepsilon$. Hence, the number of edges in the stack incident to any vertex is at most $\log_{1+\varepsilon}(W) = \mathcal{O}\left(\frac{\log W}{\varepsilon}\right)$, where W is the (normalized) maximum edge weight. Therefore, the total number of edges in the stack is $\mathcal{O}\left(\frac{n \log W}{\varepsilon}\right) = \mathcal{O}\left(\frac{n \log n}{\varepsilon}\right)$.[3]

To design a poly-streaming algorithm, we begin with a simple version and then refine it. All k processors share a global stack and a set of variables $\{\alpha_u\}_{u \in V}$, and each processor runs the above sequential streaming algorithm on its respective stream. To complete and adapt this setup for efficient execution across multiple streams, we must address two interrelated issues: (1) concurrent edge arrivals across streams may lead to contention for the shared stack or variables, and (2) concurrent updates to the shared variables may lead to inconsistencies in their observed values.

A natural approach to addressing these issues is to enforce a fair sequential strategy, where processors access shared resources in a round-robin order. While this ensures progress, it incurs $\mathcal{O}(k)$ per-edge processing time, which scales poorly with increasing k. Instead, we adopt fine-grained contention resolution that avoids global coordination by allowing processors to operate asynchronously. However, under the initial setup, this leads to $\widetilde{\mathcal{O}}(n/\varepsilon)$ per-edge processing time: a processor may be blocked from accessing shared resources until the stack has accumulated its $\widetilde{\mathcal{O}}(n/\varepsilon)$ potential edges. We address these limitations with the following design choices.

For the first issue, we observe that a global ordering of edges, as used in the single-stack solution, is not necessary; local orderings within multiple stacks suffice. In particular, we can identify a subset of edges (later referred to as *tight edges*) for which maintaining local orderings is sufficient to compute a $(2 + \varepsilon)$-approximate MWM. Hence, we can localize computation using k stacks, assigning one stack to each processor exclusively during the streaming phase. This design eliminates the $\widetilde{\mathcal{O}}(n/\varepsilon)$ contention associated with a shared stack.

[3] Throughout the paper, we assume $W = \mathcal{O}(poly(n))$. For arbitrary weights on edges, we can skip any edge whose weight is less than $\frac{\varepsilon W_{max}}{2(1+\varepsilon)n^2}$, where W_{max} denotes the maximum edge weight observed so far in the stream. This ensures that the (normalized) maximum weight the algorithm sees is $\mathcal{O}\left(n^2/\varepsilon\right)$, while maintaining a $2(1 + \mathcal{O}(\varepsilon))$ approximation ratio (see [11] for details).

Process-Edge$(e = \{u, v\}, S^\ell, \varepsilon)$**:**

/* Assumes access to global variables $\{\alpha_u\}_{u \in V}$ and locks $\{lock_u\}_{u \in V}$ */

1. if $w_e \leq (1+\varepsilon)(\alpha_u + \alpha_v)$ then return
2. repeatedly try to acquire $lock_u$ and $lock_v$ in lexicographic order of u and v as long as $w_e > (1+\varepsilon)(\alpha_u + \alpha_v)$
3. if $w_e > (1+\varepsilon)(\alpha_u + \alpha_v)$ then
   a. $g_e \leftarrow w_e - (\alpha_u + \alpha_v)$
   b. increment α_u and α_v by g_e
   c. add e to the top of S^ℓ along with g_e
4. release $lock_u$ and $lock_v$, and return

Figure 4 A subroutine used in algorithms PS-MWM and PS-MWM-LD.

However, contention still arises when updating the variables $\{\alpha_u\}_{u \in V}$. It is unclear how to resolve this contention without using additional space. Hence, we consider two strategies for processing edge streams that illustrate the trade-off between space and per-edge processing time. In the first, which we call the *non-deferrable strategy*, the decision to include an edge in a stack is made immediately during streaming. In the second, which we call the *deferrable strategy*, this decision may be deferred to post-processing. The latter strategy requires more space but achieves $\mathcal{O}(1)$ per-edge processing time.

To address the second issue, which concerns the potential for inconsistencies due to concurrent updates to the variables $\{\alpha_u\}_{u \in V}$, we observe that the variables are monotonically increasing and collectively require only $\widetilde{\mathcal{O}}(n/\varepsilon)$ updates. Thus, for most edges that are not eligible for the stacks, decisions can be made by simply reading the current values of the relevant variables. However, for the $\widetilde{\mathcal{O}}(n/\varepsilon)$ edges that are included in the stacks, we must update the corresponding variables. To ensure consistency of these updates, we associate a lock with each variable in $\{\alpha_u\}_{u \in V}$. We maintain $|V|$ exclusive locks and allow a variable to be updated only after acquiring its corresponding lock.[4]

We now outline the non-deferrable strategy of our poly-streaming algorithm for the MWM problem (for the deferrable strategy see the arXiv version). For simplicity, we assume that if a processor attempts to release a lock it did not acquire, the operation has no effect. We also assume that any algorithmic step described with the "in parallel" construct includes an implicit barrier (or synchronization primitive) at the end, synchronizing the processors participating in that step.

The non-deferrable strategy is presented in Algorithm PS-MWM, with two subroutines used by PS-MWM described in Process-Edge (Figure 4) and Process-Stack (Figure 5). In PS-MWM, Steps 1–2 form the preprocessing phase, Steps 3–4 the streaming phase, and Step 5 the post-processing phase. Each processor $\ell \in [k]$ executes PS-MWM asynchronously, except that all processors begin the post-processing phase simultaneously (due to Step 4) and then resume asynchronous execution.

In the subroutine Process-Edge, Step 2 ensures that all edges are processed using the non-deferrable strategy: a processor repeatedly attempts to acquire the locks corresponding to the endpoints of an edge $e = \{u, v\}$ until it succeeds, or the edge becomes ineligible for

[4] This corresponds to the concurrent-read exclusive-write (CREW) paradigm of the PRAM model.

Process-Stack(S^ℓ)**:**

/* Assumes access to global variables $\{\alpha_u\}_{u \in V}$; and $\{mark_u\}_{u \in V}$, initialized in Algorithm PS-MWM (or PS-MWM-LD). */

1. $\mathcal{M}^\ell \leftarrow \varnothing$
2. while $S^\ell \neq \varnothing$ do
   a. remove the top edge $e = \{u, v\}$ of S^ℓ
   b. if $w_e + g_e < \alpha_u + \alpha_v$ then wait for e to be a tight edge
      /* e is a *tight edge* if $w_e + g_e = \alpha_u + \alpha_v$ */
   c. if both $mark_u$ and $mark_v$ are set to 0 then
      /* no locking is needed since e is a tight edge */
      i. $\mathcal{M}^\ell \leftarrow \mathcal{M}^\ell \cup \{e\}$
      ii. set $mark_u$ and $mark_v$ to 1
   d. decrement α_u and α_v by g_e
3. return $\mathcal{M}^\ell$

Figure 5 A subroutine used in algorithms PS-MWM and PS-MWM-LD.

inclusion in a stack. As a result, a processor executing Step 3, has a consistent view of the variables α_u and α_v. In Step 3(c), we store the *gain* g_e of an edge e along with the edge itself for use in the post-processing phase.

When all k processors are ready to execute Step 5 of PS-MWM, the k stacks collectively contain all the edges needed to construct a $(2+\varepsilon)$-approximate MWM, which can be obtained in several ways. In the subroutine Process-Stack, we outline a simple approach based on local edge orderings. We define an edge $e = \{u, v\}$ in a stack to be a *tight edge* if $w_e + g_e = \alpha_u + \alpha_v$. Equivalently, an edge is tight if and only if all of its neighboring edges that were included after it in any stack have already been removed. Any set of tight edges can be processed concurrently, regardless of their positions in the stacks. In Process-Stack, we simultaneously process the tight edges that appear at the tops of the stacks.

3.2 Analyses

We now formally characterize several correctness properties of the algorithm and analyze its performance. These correctness properties include the absence of deadlock, livelock, and starvation. The performance metrics are space usage, approximation ratio, per-edge processing time, and total runtime.

To simplify the analysis, we assume that processors operate in a quasi-synchronous manner. In particular, to analyze Step 3 of Algorithm PS-MWM, we define an algorithmic *superstep* as a unit comprising a constant number of elementary operations.

▶ **Definition 2** (Superstep). *A processor takes one* superstep *for an edge if it executes Process-Edge with at most one iteration of the loop in Step 2 (i.e., without contention), requiring $\mathcal{O}(1)$ elementary operations. Each additional iteration of Step 2 due to contention adds one superstep, with each such iteration also requiring $\mathcal{O}(1)$ operations.*

▶ **Definition 3** (Effective Iterations). Effective iterations *is the maximum number of supersteps taken by any processor during the execution of Step 3 of Algorithm PS-MWM.*

Note that for $k = 1$, the effective iterations is equal to the number of edges in the stream. Using this notion, we align the supersteps of different processors and define the following directed graph.

▶ **Definition 4** ($G^{(t)}$). *For the* tth *effective iteration, consider the set of edges processed across all* k *streams. Let* $e_\ell = (u_\ell, v_\ell)$ *denote an edge processed in* ℓ*th stream, where* u_ℓ *precedes* v_ℓ *in the lexicographic ordering of the vertices. If processor* ℓ *is idle in the* tth *iteration, then* $e_\ell = \varnothing$. *Define* $G^{(t)} := \left(V^{(t)}, E^{(t)}\right)$, *where* $E^{(t)} := \{e_\ell \mid \ell \in [k]\}$ *and* $V^{(t)} := \bigcup_{(u_\ell, v_\ell) \in E^{(t)}} \{u_\ell, v_\ell\}$.

The following property of $G^{(t)}$ is straightforward to verify.

▶ **Proposition 5.** *$G^{(t)}$ is a directed acyclic graph.*

We show that Algorithm PS-MWM is free from deadlock, livelock, and starvation. *Deadlock* occurs when a set of processors forms a cyclic dependency, with each processor waiting for a resource held by another. *Livelock* occurs when a set of processors repeatedly form such a cycle, where each processor continually acquires and releases resources without making progress. *Starvation* occurs when a processor waits indefinitely for a resource because other processors repeatedly acquire it first. The following lemma shows that the streaming phase of Algorithm PS-MWM is free from deadlock, livelock, and starvation.

▶ **Lemma 6.** *The concurrent executions of the subroutine Process-Edge are free from deadlock, livelock, and starvation.*

Proof. Since the variables $\{\alpha_u\}_{u \in V}$ are updated only while holding their corresponding locks, we treat the locks $\{lock_u\}_{u \in V}$ as the only shared resources in Process-Edge.

Let $G^{(t)}$ be the graph defined in Definition 4. By Proposition 5, $G^{(t)}$ is a directed acyclic graph (DAG), and hence each of its components is also a DAG.

To reason about cyclic dependencies, we focus on components of $G^{(t)}$ involving processors executing Step 2 of Process-Edge. Every DAG contains at least one vertex with no outgoing edges. Thus, each such component includes an edge $e_\ell = (u_\ell, v_\ell)$ such that only processor ℓ requests $lock_{v_\ell}$. This precludes the possibility of cyclic dependencies; that is, the concurrent executions of Process-Edge is free from deadlock and livelock.

To show that starvation does not occur, suppose an edge appears in every effective iteration $t \in [a, b]$, that is, $e_\ell = (u_\ell, v_\ell) \in \bigcap_{t \in [a,b]} E^{(t)}$. We show that $b - a = \widetilde{\mathcal{O}}(n/\varepsilon)$, which bounds the number of supersteps that processor ℓ may spend attempting to acquire locks for e_ℓ. Step 2 requires one superstep per iteration, while all other steps collectively require at most one. For each $t \in (a, b]$, the component of $G^{(t-1)}$ containing e_ℓ has at least one vertex with no outgoing edge. This guarantees that at least one edge in that component acquires its locks and completes Step 3 during the $(t-1)$th effective iteration. Since Step 3 can increment the values in $\{\alpha_u\}_{u \in V}$ for at most $\mathcal{O}\left(n \log_{1+\varepsilon} W\right) = \widetilde{\mathcal{O}}(n/\varepsilon)$ edges over the entire execution, the number of iterations for which e_ℓ may remain blocked is also bounded by $\widetilde{\mathcal{O}}(n/\varepsilon)$. ◀

To analyze Step 5 of Algorithm PS-MWM, we adopt the same simplification: processors are assumed to operate in a quasi-synchronous manner. Accordingly, we define $\mathcal{U}^{(t)}$ as the set of edges present in the stacks $\bigcup_{\ell \in [k]} S^\ell$ at the beginning of iteration t of Step 2 in Process-Stack. The following definition is useful for characterizing tight edges via an equivalent notion.

▶ **Definition 7** (Follower). *An edge $e_j \in \mathcal{U}^{(t)}$ is a* follower *of an edge $e_i \in \mathcal{U}^{(t)}$ if $e_i \cap e_j \neq \varnothing$ and e_j is added to some stack S^j after e_i is added to some stack S^i. We denote the set of followers of an edge e by $\mathcal{F}(e)$.*

The proofs of the following four lemmas are included in the arXiv version. The fourth lemma establishes that the post-processing phase of PS-MWM is free from deadlock, livelock, and starvation.

▶ **Lemma 8.** *An edge e is a tight edge if and only if $\mathcal{F}(e) = \varnothing$.*

▶ **Lemma 9.** *Let $\mathcal{T}^{(t)}$ be the set of top edges in the stacks at the beginning of iteration t of Step 2 of Process-Stack. Then $\mathcal{T}^{(t)}$ contains at least one tight edge.*

▶ **Lemma 10.** *The set of tight edges in $\mathcal{U}^{(t)}$ is vertex-disjoint.*

▶ **Lemma 11.** *The concurrent executions of the subroutine Process-Stack are free from deadlock, livelock, and starvation.*

We now analyze the performance metrics of the algorithm.

▶ **Lemma 12.** *For any constant $\varepsilon > 0$, the space complexity and per-edge processing time of Algorithm PS-MWM are $\mathcal{O}(k + n \log n)$ and $\mathcal{O}(n \log n)$, respectively. Furthermore, for $L_{min} = \Omega(n)$, the amortized per-edge processing time of the algorithm is $\mathcal{O}(\log n)$.*

Proof. The claimed space bound follows from three components: $\mathcal{O}(n)$ space for the variables and locks, $\mathcal{O}(n \log n)$ space for the stacked edges, and $\mathcal{O}(1)$ space per processor.

The worst-case per-edge processing time follows from the second part of the proof of Lemma 6.

Processor ℓ processes $|E^\ell|$ edges, each requiring at least one distinct effective iteration (see Definition 3). Additional iterations may arise when it repeatedly attempts to acquire locks in Step 2 of Process-Edge. From the second part of the proof of Lemma 6, the total number of such additional iterations is bounded by $\mathcal{O}(n \log n)$. This implies that to process $|E^\ell|$ edges, a processor ℓ uses $\mathcal{O}\left(|E^\ell| + n \log n\right)$ supersteps. Therefore, the amortized per-edge processing time is

$$\mathcal{O}\left(\frac{|E^\ell| + n \log n}{|E^\ell|}\right) = \mathcal{O}\left(\frac{n \log n}{|E^\ell|}\right) = \mathcal{O}\left(\frac{n \log n}{L_{min}}\right) = \mathcal{O}(\log n). \quad \blacktriangleleft$$

Note that the amortized per-edge processing time is computed over the edges of an individual stream, not over the total number of edges across all streams. While both forms of amortization are meaningful for poly-streaming algorithms, our analysis is more practically relevant, as it reflects the cost incurred per edge arrival within a single stream.

▶ **Lemma 13.** *For any constant $\varepsilon > 0$, Algorithm PS-MWM takes $\mathcal{O}(L_{max} + n \log n)$ time.*

Proof. The preprocessing phase (Steps 1–2) takes $\Theta(n/k)$ time.

To process $|E^\ell|$ edges, processor ℓ takes $\mathcal{O}\left(|E^\ell| + n \log n\right)$ supersteps (see the proof of Lemma 12). Since $|E^\ell| \leq L_{max}$ for all $\ell \in [k]$, the time required for Step 3 is $\mathcal{O}(L_{max} + n \log n)$.

At the beginning of Step 5, the total number of edges in the stacks is $\mathcal{U}^{(1)} = \mathcal{O}(n \log n)$. By Lemma 9, iteration t of Process-Stack removes at least one edge from $\mathcal{U}^{(t)}$. Hence, the time required for Step 5 is $\mathcal{O}(n \log n)$.

The claim now follows by summing the time spent across all three phases. ◀

Now, using the characterizations of tight edges, we extend the duality-based analysis of [11] to our algorithm. Let Δ_α^e denote the change in $\sum_{u\in V}\alpha_u$ resulting from processing an edge $e \in E^\ell$ in Step 3 of Process-Edge. If an edge $e \in E^\ell$ is not included in a stack S^ℓ, then $\Delta_\alpha^e = 0$, either because it fails the condition in Step 1 or Step 3 of Process-Edge. It follows that $\sum_{e\in\bigcup_{\ell\in L} E^\ell}\Delta_\alpha^e = \sum_{u\in V}\alpha_u$. For an edge e that is included in some stack S^i, let $\mathcal{P}(e)$ denote the set of edges that share an endpoint with e and are included in some stack S^j no later than e including e itself. The following two results are immediate from Observation 3.2 and Lemma 3.4 of [11].

▶ **Proposition 14.** *Any edge e added to some stack S^ℓ satisfies the inequality*

$$w_e \geq \sum_{e'\in\mathcal{P}(e)} g_{e'} = \frac{1}{2}\left(\sum_{e'\in\mathcal{P}(e)}\Delta_\alpha^{e'}\right).$$

▶ **Proposition 15.** *After all processors complete Step 3 of Algorithm PS-MWM, the variables $\{\alpha_u\}_{u\in V}$, scaled by a factor of $(1+\varepsilon)$, form a feasible solution to the dual LP in Figure 2.*

▶ **Lemma 16.** *Let $\mathcal{M}^*$ be a maximum weight matching in G. The matching $\mathcal{M} := \bigcup_{l\in[k]}\mathcal{M}^\ell$ returned by Algorithm PS-MWM satisfies $w(\mathcal{M}) \geq \frac{1}{2(1+\varepsilon)}w(\mathcal{M}^*)$.*

Proof. We only process tight edges in Process-Stack. By Lemma 10 tight edges are vertex disjoint, and hence their independent processing does not interfere with their inclusion in $\mathcal{M}$.

By Lemma 8, an edge e included in $\mathcal{M}$ must satisfy $\mathcal{F}(e) = \varnothing$. Consider any edge $e' \in \mathcal{P}(e)\backslash\{e\}$. Since $e \in \mathcal{F}(e')$, we have $\mathcal{F}(e') \neq \varnothing$, which means e' is not a tight edge before e is processed. Thus, when e is selected for inclusion in $\mathcal{M}$, none of the edges in $\mathcal{P}(e)\backslash\{e\}$ is tight. Hence, all edges of $\mathcal{P}(e)$ are in the stacks when we are about to process e. Therefore, the total gain contributed by edges of $\mathcal{P}(e)$ can be attributed to the weight of e, and by Proposition 14, we have

$$\begin{aligned} w(\mathcal{M}) = \sum_{e\in\mathcal{M}} w_e &\geq \frac{1}{2}\left(\sum_{e\in\mathcal{M}}\sum_{e'\in\mathcal{P}(e)}\Delta_\alpha^{e'}\right) \geq \frac{1}{2}\left(\sum_{e\in\bigcup_{\ell\in[k]} S^\ell}\Delta_\alpha^e\right) \\ &= \frac{1}{2}\left(\sum_{e\in\bigcup_{\ell\in[k]} E^\ell}\Delta_\alpha^e\right) = \frac{1}{2}\left(\sum_{u\in V}\alpha_u\right). \end{aligned}$$

Let $\{x_e^*\}_{e\in E}$ be an optimal solution to the primal LP in Figure 2. By Proposition 15 and LP duality we have

$$w(\mathcal{M}^*) \leq \sum_{e\in E} w_e x_e^* \leq (1+\varepsilon)\left(\sum_{u\in V}\alpha_u\right) \leq 2(1+\varepsilon)w(\mathcal{M}). \qquad \blacktriangleleft$$

Algorithm PS-MWM uses only one pass over the streams. Theorem 1 now follows by combining the results in Lemma 12, Lemma 13, Lemma 16, and the analysis of the *deferrable strategy* sketched in the arXiv version.

4 Algorithms for Non-Uniform Memory Access Costs

In this section, we extend the algorithm from Section 3 to account for the non-uniform memory access (NUMA) costs present in real-world machines.

PS-MWM-LD(V, l, j, ε)**:**

1. In parallel initialize $lock_u$, and set α_u and $mark_u$ to 0 for all $u \in V$
   /* processor ℓ initializes or sets $\Theta(n/k)$ locks/variables */
2. In parallel initialize $lock_u^j$, and set α_u^j to 0 for all $u \in V$
   /* processor ℓ initializes or sets $\Theta(n/(k/r))$ locks / variables */
3. In parallel initialize $glock^j$ /* one processor initializes for group j */
4. $S^\ell \leftarrow \varnothing$ /* initialize an empty stack */
5. for each edge $e = \{u, v\}$ in ℓth stream do
   a. Process-Edge-LD(e, S^ℓ, ε)
6. wait for all processors to complete execution of Step 4 /* a barrier */
7. $\mathcal{M}^\ell \leftarrow$ Process-Stack(S^ℓ)
8. return $\mathcal{M}^\ell$

Figure 6 A generalization of Algorithm PS-MWM using local dual variables.

In a poly-streaming algorithm, each processor may receive an arbitrary subset of the input, making it difficult to maintain memory access locality. Modern shared-memory machines, as illustrated in Figure 1, have non-uniform memory access costs and far fewer memory controllers than processors [12]. As a result, memory systems with such limitations would struggle to handle the high volume of concurrent, random memory access requests generated by poly-streaming algorithms, leading to significant delays.

We now describe a generalization of the algorithm from Section 3 that localizes a significant portion of each processor's memory access to its near memory. This generalization applies to both edge-processing strategies introduced in Section 3.1. We focus on the *non-deferrable strategy*. (The deferrable strategy generalizes in the same way, following the same relationship between the two strategies as in the specialized case.)

The runtime of Process-Edge is dominated by the time to access the dual variables $\{\alpha_u\}_{u \in V}$. By assigning a dedicated stack to each processor, we have substantially localized accesses associated with edges in that stack. However, since a large fraction of edges is typically not included in the stacks, the runtime remains dominated by accesses to dual variables associated with these discarded edges. We therefore describe an algorithm that localizes these accesses to memory near the processor.

To localize accesses to the dual variables $\{\alpha_u\}_{u \in V}$, we observe that these variables increase monotonically during the streaming phase. This observation motivates a design in which a subset of processors maintains local copies of the variables and can discard a substantial number of edges without synchronizing with the global copy. When a processor includes an edge in its stack, it increments the corresponding dual variables in the global copy by the gain of the edge and synchronizes its local copy accordingly. As a result, some local copies may lag behind the global copy, but they can be synchronized when needed.

A general scheme for allocating dual variables is as follows. The set of k processors is partitioned into r groups. For simplicity, we assume that k is a multiple of r, so each group contains exactly k/r processors. For $r > 1$, in addition to a global copy of dual variables, we maintain r *local copies* $\{\alpha_u^j\}_{u \in V}$, one for each $j \in [r]$. Group j consists of the processors $\{\ell \in [k] \mid \lfloor \ell/(k/r) \rfloor = j\}$, and uses $\{\alpha_u^j\}_{u \in V}$ as its local copy of the dual variables. Algorithm PS-MWM is the special case $r = 1$, where all processors operate using only the global copy of the dual variables.

Process-Edge-LD$(e = \{u, v\}, S^{\ell}, \varepsilon)$**:**

/* Assumes access to $\{\alpha_u\}_{u \in V}$, $\{lock_u\}_{u \in V}$, $\{\alpha_u^j\}_{u \in V}$, $\{lock_u^j\}_{u \in V}$, and $glock^j$ */

1. if $w_e \leq (1+\varepsilon)(\alpha_u^j + \alpha_v^j)$ then return
2. repeatedly try to acquire $lock_u^j$ and $lock_v^j$ in lexicographic order of u and v as long as $w_e > (1+\varepsilon)(\alpha_u^j + \alpha_v^j)$
3. if $w_e \leq (1+\varepsilon)(\alpha_u^j + \alpha_v^j)$ then release $lock_u^j$ and $lock_v^j$, and return
4. repeatedly try to acquire $glock^j$
5. Process-Edge$(e, S^{\ell}, \varepsilon)$
6. $\alpha_u^j \leftarrow \alpha_u$ and $\alpha_v^j \leftarrow \alpha_v$ /* synchronization of local and global dual variables */
7. release $lock_u^j$, $lock_v^j$, $glock^j$ and return

Figure 7 A subroutine used in Algorithm PS-MWM-LD.

Algorithm PS-MWM-LD, along with its subroutine Process-Edge-LD, incorporates local dual variables in addition to global ones. In Step 2, processors in each group $j \in [r]$ collectively initialize their group's local copies of dual variables and locks, followed by initializing a group lock in Step 3. All other steps of the algorithm are identical to those in PS-MWM.

In the subroutine Process-Edge-LD, Step 5 implements the non-deferable strategy. Steps 1–3 and Step 6 enforce the localization of access to dual variables. Steps 2–3 ensure that, at any given time, each global dual variable is accessed by at most one processor per group; we refer to this processor as the *delegate* of the group for that variable. Thus, a processor executing Steps 4–6 serves as a delegate of its group for that time. In Step 6, after completing updates to the global variables, the delegate synchronizes its group's local copy in $\mathcal{O}(1)$ time. As a result, the waiting time on a local variable in Step 2 is bounded by the total time spent by the corresponding delegates, up to constant factors.

The delegates in each group handle vertex-disjoint edges, so concurrent executions of Step 6 would have been safe. However, the lock in Step 4 ensures that at most one delegate per group executes Step 5. Regardless of these design choices, the behavior of the delegates executing Step 5 concurrently mirrors that of processors competing for exclusive access to global dual variables in PS-MWM.

The following lemma highlights the benefit of using Algorithm PS-MWM-LD.

▶ **Lemma 17.** *For any constant $\varepsilon > 0$, in the streaming phase of Algorithm PS-MWM-LD, processors in all r groups collectively access global variables a total of $\mathcal{O}(r \cdot n \log n)$ times.*

In contrast to the result in Lemma 17, the streaming phase of Algorithm PS-MWM accesses global variables $\Omega(m)$ times or up to $\mathcal{O}(m + k \cdot n \log n)$ times.

Algorithm PS-MWM-LD, together with the generalization of the deferrable strategy, leads to the following result (a proof is included in the arXiv version).

▶ **Theorem 18.** *Let k processors be partitioned into r groups, each with its own shared local memory. For any constant $\varepsilon > 0$, there exists a* single-pass *poly-streaming algorithm for the maximum weight matching problem that achieves a $(2+\varepsilon)$-approximation. It admits a CREW PRAM implementation with $\widetilde{\mathcal{O}}(L_{max} + n)$ runtime. If $L_{min} = \Omega(n)$, the algorithm achieves $\mathcal{O}(\log n)$ amortized per-edge processing time using $\widetilde{\mathcal{O}}(k + r \cdot n)$ space. For arbitrarily balanced streams, it uses either $\widetilde{\mathcal{O}}(k + r \cdot n)$ space and $\widetilde{\mathcal{O}}(n)$ per-edge processing time, or $\widetilde{\mathcal{O}}(k \cdot n)$ space and $\mathcal{O}(1)$ per-edge processing time. The processors collectively access the global memory $\widetilde{\mathcal{O}}(r \cdot n)$ times.*

Table 1 Summary of datasets. Each collection contains eight graphs.

Graph Collection	# of Edges (in billions)
The *SSW graphs*	1.36 – 127.4
The *BA graphs*	4.64 – 550.1
The *ER graphs*	256 – 4096
The *UA-dv graphs*	275.2 – 550.1
The *UA graphs*	8.93 – 1100
The *ER-dv graphs*	32 – 4096

5 Empirical Evaluation

This section summarizes our evaluation results for Algorithm PS-MWM-LD. Detailed datasets, experimental setup, and additional comparisons (including with PS-MWM) are provided in the arXiv version.

5.1 Datasets

Table 1 summarizes our datasets. Each collection consists of eight graphs, with edge counts ranging from one billion to four trillion. To the best of our knowledge, these represent some of the largest graphs for which matchings have been reported in the literature. Exact and approximate offline MWM algorithms (see [22]) would exceed available memory on the larger graphs. The first class (SSW) consists of six of the largest graphs from the SuiteSparse Matrix collection [5] and two from the Web Data Commons [19], which includes the largest publicly available graph dataset. Other classes include synthetic graphs generated from the Barabási–Albert (BA), Uniform Attachment (UA), and Erdős–Rényi (ER) models [1, 7, 21].

5.2 Experimental Setup

We ran all experiments on a community cluster called Negishi [17], where each node has an AMD Milan processor with 128 cores running at 2.2 GHz, 256–1024 GB of memory, and the Rocky Linux 8 operating system version 8.8. The cores are organized in a hierarchy: groups of eight cores constitute a core complex that share an L3 cache. Eight core complexes form a socket, and they share four dual-channel memory controllers; two sockets constitute a Milan node [12]. Memory access within a socket is approximately three times faster than across sockets.

We implemented the algorithms in `C++` and compiled the code using the *gcc* compiler (version 12.2.0) with the `-O3` optimization flag. For shared-memory parallelism, we used the OpenMP library (version 4.5). All experiments used $\varepsilon = 1e - 6$. Reported values are the average over five runs.

5.3 Space

Figure 8 summarizes the space usage of our algorithm. For $k = 1$, the algorithm of Paz and Schwartzman [20], we store one copy of the dual variables, stack, and matching. For $k > 1$, our algorithm stores $r + 1$ copies of the dual variables (global and local), stacks, matching, and locks. We choose the values of r based on the number of memory controllers and the number of streams.

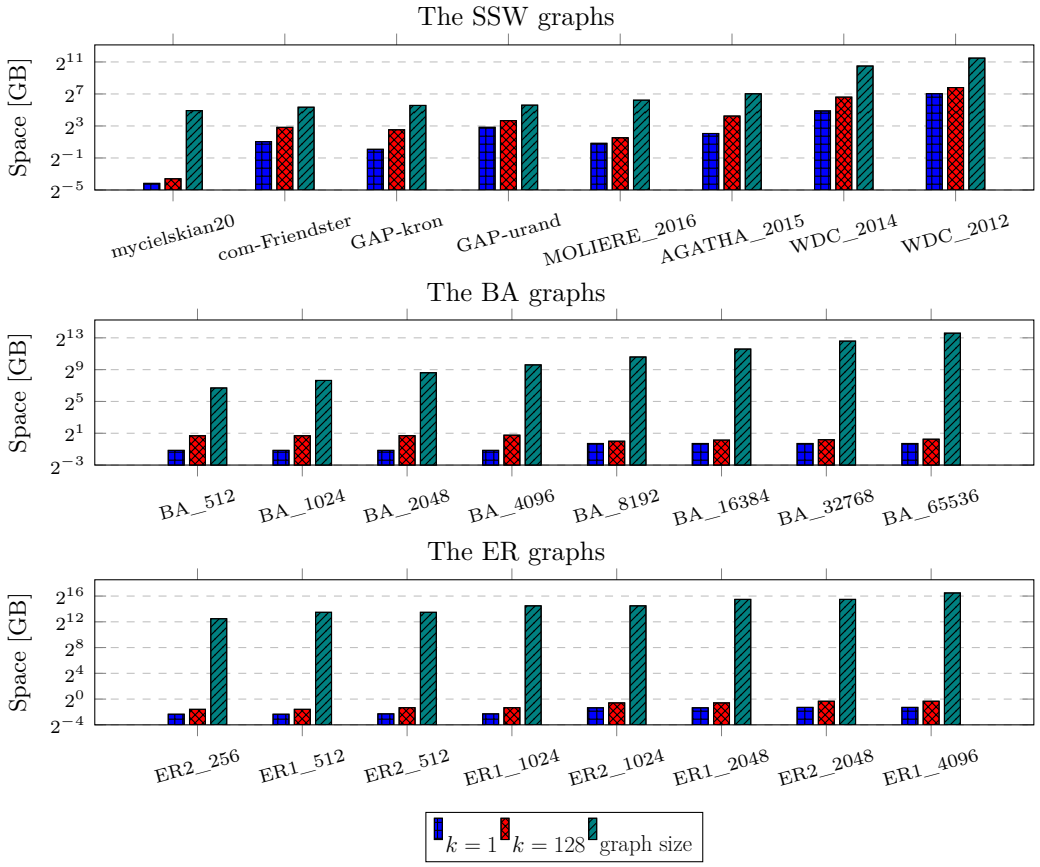


Figure 8 Memory used by the algorithm and the corresponding *graph size* (space needed to store the entire graph in CSR format). Note that the y-axes are in a logarithmic scale.

The maximum space used by our algorithm is 223 GB, for the web graph WDC_2012. In comparison, storing this graph in compressed sparse row (CSR) format would require over 2800 GB. Storing the largest graph in our datasets (ER1_4096) in CSR would require more than 91,600 GB (89.45 TB), for which our algorithm used less than 0.8 GB.

5.4 Solution Quality

min-OPT percent. In the arXiv version, we describe different ways to get *a posteriori* upper bounds on the weight of a MWM $w(M^*)$, using the values of the dual variables. Let Y_{min} denote the minimum value of these upper bounds. If $\mathcal{M}$ is a matching in the graph returned by any algorithm, then we have $\frac{w(\mathcal{M})}{w(\mathcal{M}^*)} \geq \frac{w(\mathcal{M})}{Y_{min}}$. Hence, $\frac{w(\mathcal{M})}{Y_{min}} \times 100$ gives a lower bound on the percentage of the maximum weight $w(\mathcal{M}^*)$ obtained by $\mathcal{M}$. We use *min-OPT percent* to denote the fraction $\frac{w(\mathcal{M})}{Y_{min}} \times 100$.

Figure 9 shows min-OPT percent obtained by different algorithms. In the arXiv version, we describe four dual update rules as alternatives to the default rule used in Steps 3(a)–(b) of Process-Edge. The values under $k = 1$ and $k = 128$ use the default rule, and the values under *ALG-d* use the best result among the four new dual update rules. For perspective, we include min-OPT percent obtained by the sequential 6-approximate streaming algorithm of Feigenbaum et al. [8], denoted *ALG-s*.

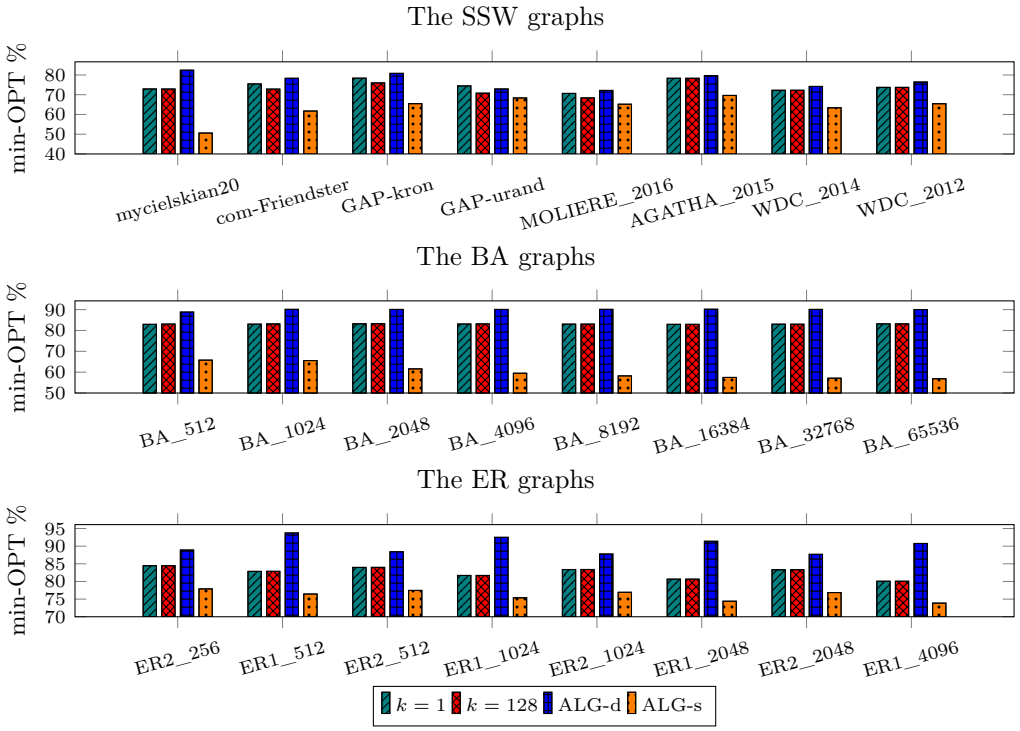


Figure 9 Comparisons of *min-OPT percent* obtained by different algorithms. *ALG-d* denotes the best results from four dual update rules, and *ALG-s* denotes the algorithm of Feigenbaum et al. [8].

The results under $k = 1$ and $k = 128$ show that, in terms of solution quality, our poly-streaming algorithm is on par with the single-stream algorithm of [20]. The values under *ALG-d* indicate further potential improvements using alternative dual update rules. The comparison with *ALG-s* supports our choice of the algorithm from [20] over other simple algorithms, such as that of [8]. The arXiv version contains comparisons with an offline algorithm and details on the dual update rules.

5.5 Runtime

We report runtime-based speedups, computed as the total time across all three phases of PS-MWM-LD (preprocessing, streaming, and post-processing). Figure 10 shows these speedups. For $k = 128$, we have speedups of $16 - 60$, $37 - 73$, and $68 - 83$ for the SSW graphs, the BA graphs, and the ER graphs, respectively.

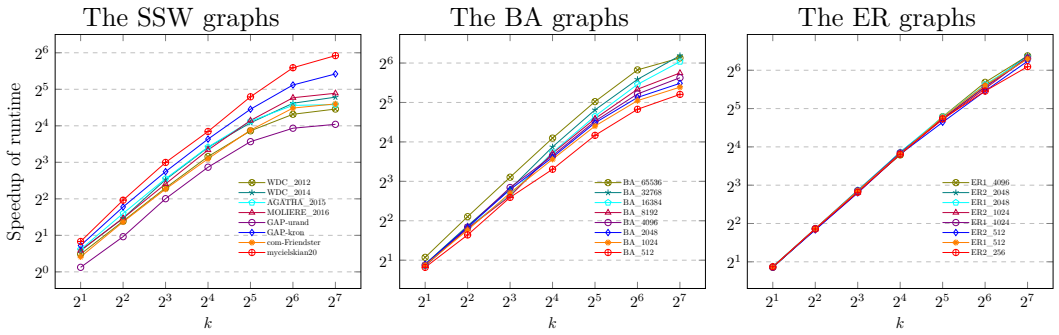


Figure 10 Speedup in runtime vs. k. Note that both axes are in a logarithmic scale.

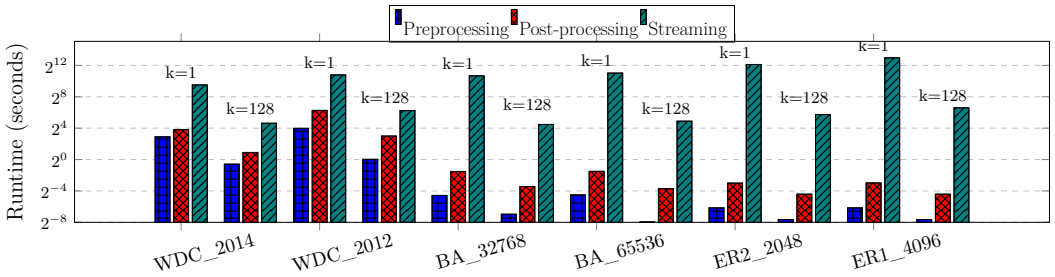


Figure 11 Breakdown of runtime into three phases for $k = 1$ and $k = 128$. Note that the y-axis is in a logarithmic scale.

Due to the significant memory bottlenecks (discussed in Section 4), we also report speedups w.r.t. *effective iterations* (Definition 3), which are less affected by such bottlenecks. The speedup w.r.t. effective iterations is the ratio of the metric for one stream to that for k streams. Now for $k = 128$, we obtain speedups of $112 - 127$, $121 - 127$, and $124 - 128$ for the SSW graphs, the BA graphs, and the ER graphs, respectively. These results indicate that shared variable access incurs no noticeable contention among processors. As a result, we expect even better runtime improvements on systems with more memory controllers or better support for remote memory access.

Figure 11 shows the runtimes for different graphs, decomposed into three phases, for $k = 1$ and $k = 128$. The plots report the absolute time savings achieved by processing multiple streams concurrently. For $k = 1$ and $k = 128$, the geometric means of the runtimes for these graphs are over 2350 seconds and under 45 seconds, respectively. For the largest graph (ER1_4096), single-stream processing took over 8000 seconds, whereas poly-stream processing reduced the time to under 100 seconds.

6 Conclusion

While numerous studies have focused on optimizing time (in parallel computing) or space (in streaming algorithms) in isolation, the poly-streaming model offers a *practically relevant paradigm* for exploring how time and space can be jointly optimized. It fills a gap by providing a formal framework for analyzing algorithmic design choices and their associated time-space trade-offs. Our study of matchings illustrates both the benefits of this paradigm and its practical relevance.

The simplicity of our matching algorithm and its generalization reflect our choice to adopt the design of [20]. We believe this principle will inspire the development of other poly-streaming algorithms. To this end, we note that [20] has also motivated simple algorithms for related problems, such as matchings with submodular objectives [16], b-matchings [14], and collections of disjoint matchings [9].

References

1 Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47, 2002.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, pages 20–29, 1996. doi:10.1145/237814.237823.

3 Sepehr Assadi. A simple (1—ε)-approximation semi-streaming algorithm for maximum (weighted) matching. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 337–354. SIAM, 2024. doi:10.1137/1.9781611977936.31.

4 Michael Crouch and Daniel M Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, pages 96–104. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICS.APPROX-RANDOM.2014.96.

5 Timothy A Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011. doi:10.1145/2049662.2049663.

6 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011. doi:10.1137/100801901.

7 Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5:17–60, 1960.

8 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005. doi:10.1016/J.TCS.2005.09.013.

9 S M Ferdous, Bhargav Samineni, Alex Pothen, Mahantesh Halappanavar, and Bala Krishnamoorthy. Semi-streaming algorithms for weighted k-disjoint matchings. In *32nd Annual European Symposium on Algorithms (ESA 2024)*, pages 53–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.

10 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 491–500, 2019. doi:10.1145/3293611.3331603.

11 Mohsen Ghaffari and David Wajc. Simplified and space-optimal semi-streaming $(2+\varepsilon)$-approximate matching. In *Symposium on Simplicity in Algorithms*, volume 69, 2019.

12 NASA High-End Computing Capability (HECC). AMD Milan Processors, 2024. Accessed: 2025-07-07. URL: https://www.nas.nasa.gov/hecc/support/kb/amd-milan-processors_688.html.

13 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External Memory Algorithms*, 50:107–118, 1998. doi:10.1090/DIMACS/050/05.

14 Chien-Chung Huang and François Sellier. Semi-streaming algorithms for submodular function maximization under b-matching, matroid, and matchoid constraints. *Algorithmica*, 86(11):3598–3628, 2024. doi:10.1007/S00453-024-01272-X.

15 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1874–1893. SIAM, 2021. doi:10.1137/1.9781611976465.112.

16 Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1914–1933. SIAM, 2021. doi:10.1137/1.9781611976465.114.

17 Gerry McCartney, Thomas Hacker, and Baijian Yang. Empowering faculty: A campus cyberinfrastructure strategy for research communities. *Educause Review*, 2014.

18 Andrew McGregor. Finding graph matchings in data streams. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 170–181. Springer, 2005. doi:10.1007/11538462_15.

19 Robert Meusel, Sebastiano Vigna, Oliver Lehmberg, and Christian Bizer. The graph structure in the web–analyzed on different aggregation levels. *The Journal of Web Science*, 1, 2015. doi:10.1561/106.00000003.

20 Ami Paz and Gregory Schwartzman. A $(2+\varepsilon)$-approximation for maximum weight matching in the semi-streaming model. *ACM Transactions on Algorithms (TALG)*, 15(2):1–15, 2018. doi:10.1145/3274668.

21 Erol A Peköz, Adrian Röllinn, and Nathan Ross. Total variation error bounds for geometric approximation. *Bernoulli*, 19(2):610–632, 2013.

22 Alex Pothen, S M Ferdous, and Fredrik Manne. Approximation algorithms in combinatorial scientific computing. *Acta Numerica*, 28:541–633, 2019. doi:10.1017/S0962492919000035.

23 Mariano Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, 62(1-2):1–20, 2012. doi:10.1007/S00453-010-9438-5.

Efficiency of Learned Indexes on Genome Spectra

Md. Hasin Abrar[†]
Department of Computer Science and Engineering, Penn State, University Park, PA, USA

Paul Medvedev[†]
Department of Computer Science and Engineering, Penn State, University Park, PA, USA
Department of Biochemistry and Molecular Biology, Penn State, University Park, PA, USA
Huck Institutes of the Life Sciences, Penn State, University Park, PA, USA

Giorgio Vinciguerra[†]
Department of Computer Science, University of Pisa, Italy

Abstract

Data structures on a multiset of genomic k-mers are at the heart of many bioinformatic tools. As genomic datasets grow in scale, the efficiency of these data structures increasingly depends on how well they leverage the inherent patterns in the data. One recent and effective approach is the use of learned indexes that approximate the rank function of a multiset using a piecewise linear function with very few segments. However, theoretical worst-case analysis struggles to predict the practical performance of these indexes.

We address this limitation by developing a novel measure of piecewise-linear approximability of the data, called CaPLa (Canonical Piecewise Linear approximability). CaPLa builds on the empirical observation that a power-law model often serves as a reasonable proxy for piecewise linear-approximability, while explicitly accounting for deviations from a true power-law fit. We prove basic properties of CaPLa and present an efficient algorithm to compute it. We then demonstrate that CaPLa can accurately predict space bounds for data structures on real data. Empirically, we analyze over 500 genomes through the lens of CaPLa, revealing that it varies widely across the tree of life and even within individual genomes. Finally, we study the robustness of CaPLa as a measure and the factors that make genomic k-mer multisets different from random ones.

2012 ACM Subject Classification Applied computing → Bioinformatics; Applied computing → Computational biology; Theory of computation → Data structures design and analysis

Keywords and phrases Genome spectra, piecewise linear approximation, learned index, k-mers

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.18

Related Version A complete version of this work, including full proofs, is available but omitted here due to space limits. A preliminary version appeared in a preprint but is subsumed by this paper.

Full Version: `https://www.biorxiv.org/content/10.1101/2025.07.10.664199v1` [3]
Previous Version: `https://www.biorxiv.org/content/10.1101/2024.02.08.579510v1` [1]

Supplementary Material *Software (Source Code)*: `https://github.com/medvedevgroup/CaPLa` archived at `swh:1:dir:da45f156bdafa582fd16f04690ee49e184bf3590`

Funding This material is based upon work supported by the NSF under Grants No. DBI2138585 and OAC1931531. Research reported in this publication was supported by the National Institute Of General Medical Sciences of the NIH under Award Number R01GM146462. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH. GV was supported by the NextGenerationEU – National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) – Project: "SoBigData.it - Strengthening the Italian RI for Social Mining and Big Data Analytics" – Prot. IR0000013 – Avviso n. 3264 del 28/12/2021.

Acknowledgements We thank anonymous reviewers for helping improve various versions of this paper.

[†] The authors are listed in alphabetical order.

© Md. Hasin Abrar, Paul Medvedev, and Giorgio Vinciguerra;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Data structures on k-mers (fixed-length strings) originating from genomes form a crucial component of many bioinformatics tools [22]. Their space usage is now a major bottleneck at the forefront of biological discovery. For example, the Logan project needed two petabytes to index the 31-mers from the Sequence Read Archive [7]. Existing data structures often meet the space challenge by exploiting the non-uniform structure of genomic k-mer sets [6]. In doing so, they circumvent [2, 17, 23, 28] the theoretical worst- and average-case lower bounds, which offer a more pessimistic perspective [25].

A popular approach is based on learned indexes, a recent research direction that takes advantage of the internal patterns in the rank curve of a dataset [5, 10, 18, 32]. The rank of an element x in a multiset S is defined as the position of the first occurrence of x in the sorted list of S. A learned index is constructed from S and, given a query k-mer, returns an error-bounded prediction for its rank. Initial approaches used machine learning to construct the index [13, 14, 15], but it later turned out that in this setting, it is more effective to model the rank curve using a piecewise linear approximation [2, 4, 17], as depicted in Figure 1.

Learned indexes based on piecewise linear approximations (PLAs) of the rank curve have resulted in practically groundbreaking time/space tradeoffs, both on genomic [2, 8, 17] and other kinds of data [8, 11, 12, 16, 19, 20, 21, 31]. However, existing theoretical worst-case analyses only show improved performance under assumptions that are unrealistic for genomic datasets, such as the independence and identical distribution of gaps between k-mers [9]. This leaves a gap between the theoretical analysis of k-mer-based learned indexes and their much better performance on real data [2, 17]. By better understanding and characterizing the structural properties of genomic k-mer sets, we can not only better predict the performance of learned indexes but also drive the development of new PLA-based tools that maximally exploit the properties of real data.

Our approach in this paper is to develop a measure of piecewise linear (PL) approximability which can be used to parameterize the theoretical analysis of PLA-based methods. Parameterizing string complexity is an active area of research [26], but none of the existing measures capture the PL-approximability of a genome's rank curve. Previous papers have proposed modeling the PL-approximability of S using a power-law [9, 11], and have implicitly used the two parameters of a power-law fit as proxies for the PL-approximability of S. However, the interpretation of such parameters is limited when the rank curve strays from a power-law, as (we will show) occurs in real genomic data.

We develop a novel measure of PL-approximability, which we call CaPLa (Canonical PL-approximability) (Section 2). It builds on the idea of a power-law fit but adds the uncertainty of the fit as part of the measure. We prove basic properties of CaPLa, including its existence, and give an algorithm to compute it (Section 3). The algorithm is exact under certain conditions of the data, which we empirically show are nearly always met in practice. We demonstrate how CaPLa can be applied to derive space bounds that accurately reflect properties of real data (Section 4).

Finally, we apply CaPLa to analyze genome spectra, where a spectrum is the multiset of all k-mers appearing in a string.[1] We use a dataset of more than 500 genomes and find that CaPLa varies greatly across the tree of life and even within individual genomes (Section 5). We also show that CaPLa is a good predictor of the space usage of a learned index data structure (the PLA-index [2]). We use controlled experiments to elucidate what

[1] Our tool for computing CaPLa is available at `https://github.com/medvedevgroup/CaPLa`.

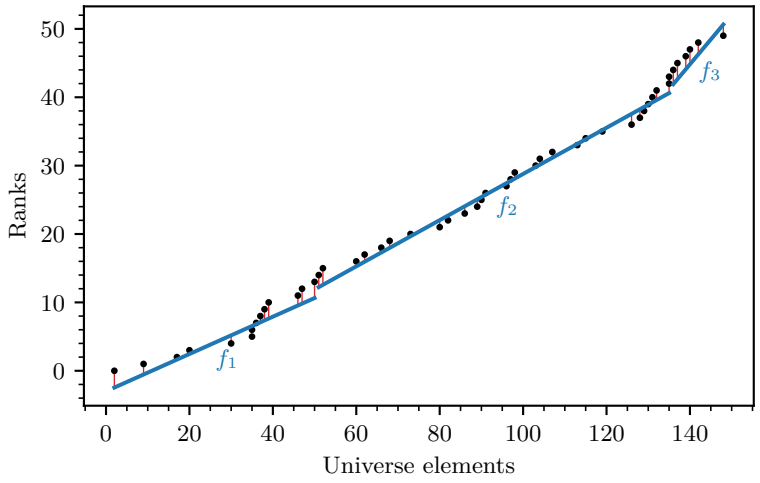


Figure 1 An example multiset S of size $N = 50$ with $n = 48$ distinct elements drawn from a universe of size $u = 150$, plotted as points $\{(x, \text{RANK}_S(x)\}_{x \in S}$ on the plane. The points are approximated using a PLA with error bound $\varepsilon = 2$, where the vertical orange lines show the error. The PLA uses 3 segments, which is the smallest possible for S at $\varepsilon = 2$, so $b(\varepsilon) = 3$.

makes genome spectra different from random k-mer multisets. Finally, we verify that CaPLa has some of the desired properties of a measure of genome complexity, namely robustness to variation in genome or k-mer size. Our work helps reduce the gap between theory and practice and aims to deepen our understanding of when and how to best apply PLA-based indexes to real genomic data.

2 PL-approximability definitions and motivation

In this section, we present the key definitions of PL-approximability and the motivation behind them. We start with a multiset S of elements from an integer universe $[u] = \{0, 1, \ldots, u-1\}$. In the case of k-mers, $u = 4^k$, though our definitions work for any u. We will use the notation that n is the number distinct and N is the total number of elements in S. The function we will be approximating is defined as $\text{RANK}_S(x) = |\{y \in S \mid y \leq x\}|$, for all $x \in [u]$. The key tool in designing PLA-based learned data structures [2, 8, 11, 12, 17] is the following:

▶ **Definition 1** (Piecewise linear ε-approximation). *For a given positive integer ε, a* piecewise linear ε-approximation (PLA) *of S is a partition of $[u]$ into subintervals such that, for each subinterval $[a_i, b_i]$, there exists a segment (i.e., a linear function) f_i such that $|f_i(x) - \text{RANK}_S(x)| \leq \varepsilon$ for all $x \in [a_i, b_i]$.*

Figure 1 shows an example of a PLA. The usefulness of a PLA comes from the fact that a data structure can avoid storing the full rank function and instead just store the PLA, as long as it has a way to handle the uncertainty due to the error of the approximation. To this end, the PLA with the smallest number of segments is the most space-efficient representation. Such a PLA can also be computed efficiently [27]. As a result, minimum-sized PLAs have formed the basis of several data structures [2, 8, 11, 19, 21, 31]. We formalize it with the following definition.

▶ **Definition 2** (PLA-size). *The* PLA-size *of S is the function $b\colon \mathbb{Z}^+ \to \mathbb{Z}^+$ mapping a positive integer ε to the smallest integer $b(\varepsilon)$ such that there exists a piecewise linear ε-approximation of S using exactly $b(\varepsilon)$ segments.*

In this paper, we introduce a closely related measure, but one that factors out the effect of the data size.

▶ **Definition 3** (PL-approximability). *The* PL-approximability *of S is the function mapping a positive integer ε to $\frac{n}{b(\varepsilon)}$, where $b(\varepsilon)$ is the PLA-size of S.*

The PL-approximability of S can be interpreted as the longest average number of elements spanned by a segment in a PLA of S, so it directly quantifies the efficiency of a PLA in capturing the structure and distribution of the underlying data.

The space and time performance of learned data structures based on PLAs are often bounded in terms of the PLA-size [2, 8, 11]. Typically, smaller values of ε lead to faster query times, but require a larger number of segments $b(\varepsilon)$ and thus increased space. The PLA-size and PL-approximability can then be tabulated for a given dataset and plugged into these space- and time-bounds to assess the resulting space-time trade-off. However, this approach provides little insight into broader trends and differences among data distributions. Ideally, the PL-approximability could be modeled as a parametrized family of functions, such that fitting this family to S yields parameters that fully capture the PL-approximability of S.

A key difficulty is that the shape of the PLA-size function varies significantly depending on the structure and distribution of the underlying data. In general, the PLA-size is bounded between $b(\varepsilon) = 1$ and $b(\varepsilon) = N/(2\varepsilon)$ for all ε [11]. For sets where the gaps between elements are random and independently drawn, the PLA-size has been shown to be $b(\varepsilon) = \mathcal{O}(N/\varepsilon^2)$ [9]. For real data, the PLA-size rarely admits a closed-form expression, but power-law relations of the form $b(\varepsilon) = n/(\beta\varepsilon^\alpha)$ have been empirically observed [9, 11]. Here, β and α are dataset-specific constants found through fitting to the tabulated PLA-size. This power-law modeling provides a degree of generality, but when the PLA-size does not exactly follow a power-law, it has several drawbacks. Figure 2 demonstrates that this type of modeling 1) has a high variability in the error, 2) is not robust to the choice of fitting algorithm, and 3) is unpredictable in whether it over- or under-estimates the true PLA-size. Such drawbacks make it challenging to incorporate this type of model into downstream analyses of data structures, especially since its predictions are neither worst- nor average-case bounds.

To address this challenge, we propose bounding the PLA-size using two power laws instead of relying on a single fitted curve. We first fix a finite set $\mathcal{E} \subset \mathbb{Z}^+$ of ε values for which the PL-approximability is evaluated. This reflects practical applications, where ε is chosen from a set of values according to the desired space-time trade-off.

▶ **Definition 4** (Bounded PL-approximability). *The PL-approximability of S is power-law bounded over $\mathcal{E}$ with parameters $(\alpha, \beta_{\text{low}}, \beta_{\text{high}})$ if*

$$\beta_{\text{low}}\varepsilon^\alpha \leq \frac{n}{b(\varepsilon)} \leq \beta_{\text{high}}\varepsilon^\alpha \quad \text{for all } \varepsilon \in \mathcal{E}. \tag{1}$$

When $\beta_{\text{low}} = \beta_{\text{high}}$, this is equivalent to a perfect power-law fit. The role of α is to capture the rate of segment growth as a function of ε. If $\beta_{\text{low}} = \beta_{\text{high}}$, then when ε doubles, the average segment length increases by a factor of 2^α. There is similarly an intuitive interpretation of β_{low} and β_{high}: when $\varepsilon = 1$, the average segment length is between β_{low} and β_{high}.

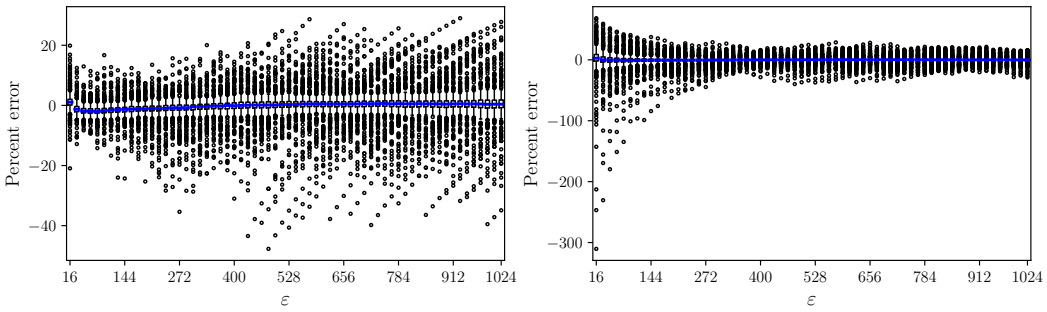


Figure 2 Errors due to fitting a single power-law curve to the PLA-size of a set of 513 genomes (dataset details are in Section 5.1). The left panel models the relationship as $n/b(\varepsilon) = \beta\varepsilon^{\alpha}$ and uses Python's *scipy.optimize.curve_fit* with the Levenberg–Marquardt algorithm to estimate α and β. The right panel uses the model $\log(n/b(\varepsilon)) = \alpha \log \varepsilon + \log \beta$ and applies Python's *scipy.stats.linregress* to estimate α and β. For each $\varepsilon \in \{16, 32, 48, 64, \ldots, 1024\}$, we calculate the percent error between $b(\varepsilon)$ and the prediction $n/(\beta\varepsilon^{\alpha})$ and show a box plot of the distribution aggregated over the 513 genomes (with the median line colored in blue).

Definition 4 captures a full spectrum of data distributions and their PLA-size, from the best case $b(\varepsilon) = 1$ (with $\alpha = 0$ and $\beta_{\text{low}} = \beta_{\text{high}} = n$), to the worst case $b(\varepsilon) = N/(2\varepsilon)$ from [11] (with $\alpha = 1$ and $\beta_{\text{low}} = \beta_{\text{high}} = 2n/N$), to the case of random gaps from [9] (with $\alpha = 2$ and $\beta_{\text{low}} = \beta_{\text{high}} = cn/N$ for a constant $c > 0$). More importantly, it enables bounding the PL-approximability of real datasets where an exact power-law fitting is not possible.

The PL-approximability of a single dataset can be power-law bounded for infinitely many values of $\alpha, \beta_{\text{low}}$, and β_{high}. To capture a single parameterization that can be used as a proxy for the PL-approximability of a dataset, we introduce the following notion of *canonical PL-approximability*.

▶ **Definition 5** (CaPLa). *A pinch point of S is any α^* that satisfies*

$$\alpha^* = \operatorname*{argmin}_{\alpha \geq 0} \big(H(\alpha) - L(\alpha)\big), \quad \textit{where} \quad L(\alpha) = \min_{\varepsilon \in \mathcal{E}} \frac{n}{\varepsilon^{\alpha} b(\varepsilon)}, \quad H(\alpha) = \max_{\varepsilon \in \mathcal{E}} \frac{n}{\varepsilon^{\alpha} b(\varepsilon)}. \tag{2}$$

If a unique pinch point α^* exists, then we let $\beta^*_{\text{low}} = L(\alpha^*)$ and $\beta^*_{\text{high}} = H(\alpha^*)$ and say that S has a canonical PL-approximability (CaPLa) of $(\alpha^*, \beta^*_{\text{low}}, \beta^*_{\text{high}})$. Otherwise, we say that the CaPLa of S is undefined.

We note that if S has CaPLa $(\alpha^*, \beta^*_{\text{low}}, \beta^*_{\text{high}})$, then Equation 1 immediately implies that S is power-law bounded with parameters $(\alpha^*, \beta^*_{\text{low}}, \beta^*_{\text{high}})$.

3 Existence and computation of the canonical PL-approximability

In this section, we show under which conditions the canonical PL-approximability exists and how it can be efficiently computed. We do so by introducing a geometric object, the *twisted ribbon*, which captures the entire family of valid power-law bounds for S. We then study the structural properties of this ribbon to identify a *pinch point*, which yields the canonical PL-approximability (Definition 5). Finally, we describe a two-phase algorithm that efficiently locates a pinch point based on the twisted ribbon's geometry.

We first observe that, for any given α, the tightest possible power-law bound on the PL-approximability of S is given by choosing β_{low} and β_{high} as $L(\alpha)$ and $H(\alpha)$, respectively. The proof follows easily from the definitions of L and H.

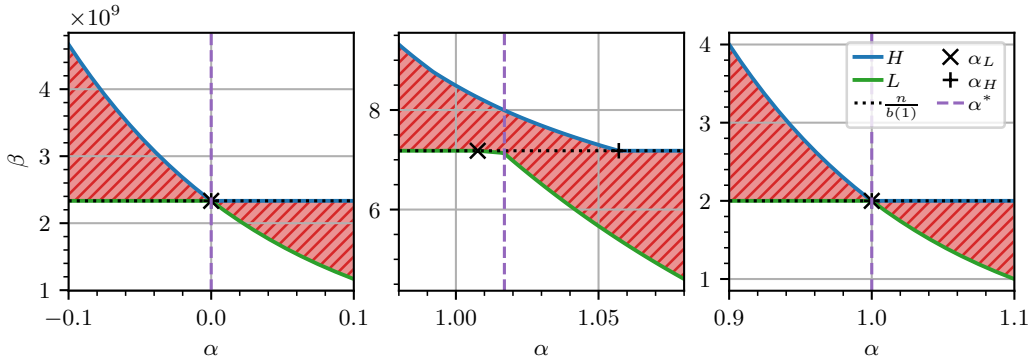


Figure 3 On the left, a twisted ribbon corresponding to a set S giving the best case $b(\varepsilon) = 1$ for any $\varepsilon \in \{1, \ldots, 1024\}$. On the center, a twisted ribbon derived from the human genome (T2TCHM13v2.0). On the right, a twisted ribbon corresponding to a set S giving the worst case $b(\varepsilon) = N/(2\varepsilon)$ for any ε. For symmetry with the other plots, the left plot extends to values of $\alpha < 0$ despite the ribbon being formally defined for $\alpha \geq 0$.

▶ **Lemma 6.** *For any $\alpha \geq 0$ and $\delta > 0$, we have that (i) S is power-law bounded over $\mathcal{E}$ with parameters $(\alpha, L(\alpha), H(\alpha))$, and (ii) S is not power-law bounded over $\mathcal{E}$ with parameters $(\alpha, L(\alpha) + \delta, H(\alpha) - \delta)$.*

We can plot the functions H and L on a Cartesian plane with α along the horizontal axis and β along the vertical axis. The region enclosed between these two curves forms the aforementioned twisted ribbon, depicted in Figure 3 and formally defined as follows.

▶ **Definition 7** (Twisted ribbon)**.** *The twisted ribbon of S is the region in the α-β space enclosed between the curves $L(\alpha)$ and $H(\alpha)$ for any $\alpha > 0$. Formally, the twisted ribbon of S is the set $\mathcal{R}_S = \{(\alpha, \beta) \in \mathbb{R}^2_{\geq 0} \mid L(\alpha) < \beta < H(\alpha), \alpha \geq 0\}$.*

Observe that choosing β_{low} or β_{high} inside of the twisted ribbon (that is, setting $\beta_{\text{low}} > L(\alpha)$ or $\beta_{\text{high}} < H(\alpha)$ for some α) would violate Equation 1, and therefore would not yield a valid bound on the PL-approximability of S. This reduces the space of relevant power-law bounds to that of $\{(\alpha, L(\alpha), H(\alpha))\}_{\alpha \geq 0}$.

We further aim to summarize this entire family of bounds into a single representative instance. Intuitively, this means identifying a value of α that yields the tightest bound. This corresponds to a value α^* that minimizes the twisted ribbon width $W(\alpha) = H(\alpha) - L(\alpha)$, i.e., what we call a pinch point (Definition 5).

To study the existence and computation of a pinch point, we first show some structural properties of the twisted ribbon. For technical convenience, we assume that the (nonincreasing integer) function $b(\varepsilon)$ has been extended to a continuous, nonincreasing, and differentiable function on the reals, e.g., via monotone cubic interpolation.

▶ **Lemma 8.** *If $1 \in \mathcal{E}$ and $|\mathcal{E}| > 1$, then:*

- **(i)** *H and L are continuous functions.*
- **(ii)** *For any $\alpha \geq 0$, it holds $0 \leq L(\alpha) \leq \frac{n}{b(1)} \leq H(\alpha) \leq \frac{n}{b(\varepsilon_{\max})}$, where $\varepsilon_{\max} = \max \mathcal{E}$.*
- **(iii)** *There exists a value α_L such that $L(\alpha) = \frac{n}{b(1)}$ for $\alpha \leq \alpha_L$, and $L(\alpha_L + \delta)$ decreases as $\delta > 0$ increases. We call α_L a flattening point of S.*
- **(iv)** *There exists a value α_H such that $H(\alpha) = \frac{n}{b(1)}$ for $\alpha \geq \alpha_H$, and $H(\alpha_H - \delta)$ increases as $\delta > 0$ increases. We also call α_H a flattening point of S.*
- **(v)** *The flattening points satisfy $\alpha_L \leq \alpha_H$.*

Note that the lemma holds for any dataset, as its conditions are related to a parameter ($\mathcal{E}$) chosen by the user. The following result shows that a pinch point exists and is confined between (or at) the flattening points mentioned in Lemma 8 and depicted in Figure 3.

▶ **Theorem 9.** *If $1 \in \mathcal{E}$ and $|\mathcal{E}| > 1$, then a pinch point α^* of S exists and is guaranteed to lie in the interval $[\alpha_L, \alpha_H]$, where α_L and α_H are the flattening points of S.*

We now focus on finding a pinch point, that is, on solving the problem $\operatorname{argmin}_{\alpha \geq 0} W(\alpha)$. Observe W is non-differentiable due to the presence of min and max operations. Moreover, we assume that W is *unimodal*, namely, it has a unique minimum α^*, is strictly decreasing for all $\alpha < \alpha^*$, and strictly increasing for $\alpha > \alpha^*$. While we do not have any general theoretical guarantee on the unimodality of W (except in simple cases such as those shown in the left and right plots of Figure 3), our empirical observation suggests that unimodality often holds in practice (see Section 5.5). If this is not the case, our algorithm finds a local minimum instead. We also assume $b(\varepsilon)$ has been precomputed for all $\varepsilon \in \mathcal{E}$, allowing $W(\alpha)$ to be evaluated in $\mathcal{O}(|\mathcal{E}|)$ time via a direct computation based on Equation 2. This precomputation can be done in $\mathcal{O}(N|\mathcal{E}|)$ time [27].

The algorithm works in two phases. In the first phase, we identify the flattening points α_L and α_H by observing from Lemma 8 that α_L is the smallest value of α such that $\frac{n}{\varepsilon^\alpha b(\varepsilon)} = \frac{n}{b(1)}$ for some $\varepsilon \in \mathcal{E} \setminus \{1\}$. This equality simplifies to $\alpha = \log_\varepsilon \frac{b(1)}{b(\varepsilon)}$, so we obtain $\alpha_L = \min_{\varepsilon \in \mathcal{E} \setminus \{1\}} \log_\varepsilon \frac{b(1)}{b(\varepsilon)}$, which can be computed in $\mathcal{O}(|\mathcal{E}|)$ time. A similar derivation with max instead of min yields α_H.

In the second phase, we run a derivative-free minimization algorithm (golden-section search) within the interval $[\alpha_L, \alpha_H]$, which is guaranteed to contain α^* due to Theorem 9. It converges in $\lceil \log_\phi \frac{\alpha_H - \alpha_L}{\tau} \rceil$ iterations, where τ is the desired tolerance, and $\phi \approx 1.618$ is the golden ratio [29]. Since the cost of each iteration is dominated by the evaluation of $W(\alpha)$, the overall time complexity of finding the pinch point is $\mathcal{O}(|\mathcal{E}| \log \frac{\alpha_H - \alpha_L}{\tau})$. This gives the following theorem.

▶ **Theorem 10.** *Suppose the ribbon width $W(\alpha) = H(\alpha) - L(\alpha)$ is unimodal, $1 \in \mathcal{E}$, and $|\mathcal{E}| > 1$. Suppose the value $b(\varepsilon)$ has been precomputed for all $\varepsilon \in \mathcal{E}$ in $\mathcal{O}(N|\mathcal{E}|)$ time. Then, there exists an algorithm that finds the pinch point α^* in $\mathcal{O}(|\mathcal{E}| \log \frac{\alpha_H - \alpha_L}{\tau})$ time, where τ is the desired tolerance.*

Section 5.1 will show that the runtime of this algorithm is on the order of a few milliseconds in practice.

4 Using PL-approximability bounds in downstream applications

There are a number of PLA-based learned data structures whose space-time performance is bounded in terms of the PLA-size, such as the PGM-index [11, Theorem 1], LeMonHash [8, Theorem 2], and the PLA-index [2, Theorem 1]. In this section, we showcase how our PL-approximability framework can be used to reinterpret such bounds, focusing on the PLA-index as a concrete example particularly relevant in genomic applications. The PLA-index has been applied to substantially improve the run time of suffix array search and decrease the space of a read aligner and direct-access lookup tables [2].

The PLA-index uses a piecewise linear ε-approximation to a multiset of k-mers S to estimate the rank of a query k-mer in S [2]. The tuning parameter ε represents a time/space trade-off, with smaller ε values resulting in faster index query times but more segments.

Theorem 1 in [2] describes the size of the PLA-index as a function of the number of segments used by the PLA index (denoted by s_ε), N, and ε:

$$s_\varepsilon(2k - \lg \frac{s_\varepsilon^{31/16}}{N(1+4\varepsilon)} + \mathcal{O}(1)). \tag{3}$$

As s_ε can be high in the worst case, Equation 3 does not capture the fact that the space is very low in practice.

Here, we show that if S has a power-law bounded PL-approximability, then Equation 3 can be turned into a useful predictor of PLA-index space usage. One of the technical difficulties is that PL-approximability gives bounds on the PLA size $b(\varepsilon)$, whereas Equation 3 is stated in terms of s_ε. Though the PLA-index uses a segmentation based on the optimal one, it requires consecutive segments to share an endpoint and in some rare cases breaks segments. This dependence of s_ε on $b(\varepsilon)$ is difficult to quantify in terms of S. Instead, we isolate the dependence by expressing our results in terms of $\gamma_\varepsilon \triangleq \frac{s_\varepsilon}{b(\varepsilon)}$. Though γ_ε is theoretically unknown (except that $\gamma_\varepsilon \geq 1$), empirical results show that it is close to one on genome spectra. For example, for the 21-spectrum of chromosome 1 of the human genome, γ_ε values are below 1.21 for all tested ε up to $\varepsilon = 1023$.

▶ **Theorem 11.** *Let S be a multiset of k-mers, with n distinct elements, N total elements. Suppose the PL-approximability of S is $(\alpha, \beta_{\text{low}}, \beta_{\text{high}})$ power-law bounded over $\mathcal{E}$. Let B_ε be the number of bits used by the PLA-index on S with tuning parameter ε. Then, for all $\varepsilon \in \mathcal{E}$, we have that*

$$f(\beta_{\text{high}}, \beta_{\text{low}}) \leq B_\varepsilon \leq f(\beta_{\text{low}}, \beta_{\text{high}}),$$

where

$$f(\beta_1, \beta_2) = \frac{n\gamma_\varepsilon}{\beta_1 \varepsilon^\alpha}\left(2k - \frac{15 \lg n}{16} + \left(\frac{31 \log_\varepsilon(\beta_2 \varepsilon^\alpha)}{16} + 1\right)\lg \varepsilon + C\right), \tag{4}$$

and $C = c - \frac{31 \lg \gamma_\varepsilon}{16} + \lg N/n$ and, for large enough $\frac{n}{\beta_{\text{high}}\varepsilon^\alpha}$, c is a number between 5 and 7.

We have isolated and combined some terms into a variable C, in order to convey that they can be thought of as constants in practice. First, N/n is in practice smaller than 2 when avoiding small values of k, for which a suffix array would not make sense as an application anyway. Second, as we mentioned earlier, γ_ε is in practice a number close to 1.

Interpreting Eq. 4 is helped by the observation that given the type of values we see in practice, the dominating term is the leading one, i.e. $1/(\beta_1 \varepsilon^\alpha)$. This intuition is confirmed by Figure 4AB, which plots Eq. 4 using real numbers from human chromosome 1. As suggested by the leading term, increasing the α value of a dataset by δ results in the leading term decreasing by a factor of ε^δ, with the plots confirming this type of exponential decay for Eq. 4 overall. Note that for $\varepsilon = 63$ (Panel A), the decay is slower than for $\varepsilon = 1023$ (Panel B), as expected from the leading term. Figure 4CD shows how the gap between β_{low} and β_{high} affects the gap between the upper and lower bounds of Theorem 11 and gives the scale of the gap at which our bounds become meaningless.

5 Experimental results

The goal of our experiments is threefold. First, we want to understand how much PL-approximability varies among the tree of life. Second, we want to understand the properties of genome spectra that make them different from random k-mer multisets. Finally, we want to evaluate the suitability of CaPLa as a measure of PL-approximability. Our tool for computing CaPLa, as well as reproducibility information, is available through GitHub at `https://github.com/medvedevgroup/CaPLa`.

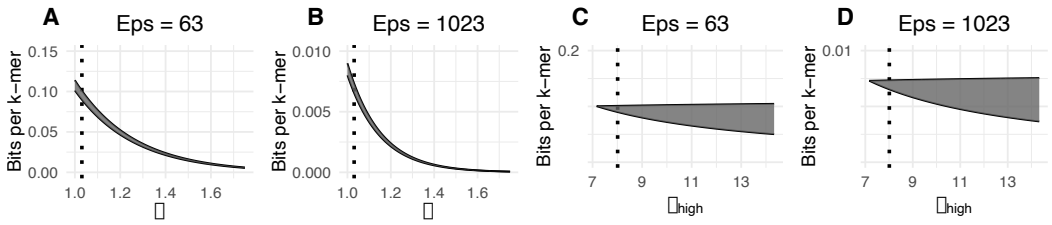


Figure 4 Analysis of space usage of PLA index, as predicted by Theorem 11, plugging in $C = 6$. In each panel, the shaded area is bounded from above by the upper bound curve and below by the lower bound curve. We use values corresponding to human chromosome 1 with $k = 21$: $n = 195,735,278$, $\alpha^* = 1.03$, $\beta^*_{\text{low}} = 7.2$, and $\beta^*_{\text{high}} = 8.0$, $\gamma_{63} = 1.13$ and $\gamma_{1023} = 1.15$. In panels A and B, we fix $\beta_{\text{low}} = \beta^*_{\text{low}}$ and $\beta_{\text{high}} = \beta^*_{\text{high}}$ while varying α. The vertical dotted line corresponds to $\alpha = \alpha^*$. In panels C and D, we fix the values of $\alpha = \alpha^*$ and $\beta_{\text{low}} = \beta^*_{\text{low}}$ and vary β_{high}. The vertical dotted line corresponds to $\beta_{\text{high}} = \beta^*_{\text{high}}$.

5.1 Experimental setup

Data. We downloaded a sample of RefSeq genomes that are complete and full, do not have any missing bases, and are longer than 10,000nt. We further filtered out any genomes for which the number of segments at $\varepsilon = 1024$ was 1 (i.e. $b(1024) = 1$). This excludes genomes that are too trivial to offer meaningful insight and for which the PL-approximability becomes trivially dominated by n. We process the genomes to make everything uppercase, remove any letters that are not A, C, G, or T, and concatenate all contigs into a single string. The resulting dataset contains 513 genomes, representing the kingdoms of virus (91 genomes), bacteria (200 genomes), archaea (200 genomes), and fungi (22 genomes). The median genome length was 2.98 mil and the maximum was 63 mil . We also downloaded six other larger genomes: *C. elegans*, *P. yoelii*, *M. commoda*, *O. lucimarinus*, and the latest high-quality assemblies of the human (T2TCHM13v2.0) and Gorilla genomes .

Tool. Our tool to compute CaPLa takes a genome, a k-mer size, and $\mathcal{E}$, builds a suffix array of the genome using libdivsufsort [24], computes the values $\{b(\varepsilon)\}_{\varepsilon\in\mathcal{E}}$ using O'Rourke's algorithm [27] on the genome's k-mers (extracted in sorted order using the suffix array), and then computes CaPLa using the algorithm of Theorem 10 with tolerance $\tau = 1.49 \times 10^{-8}$ (i.e., the square root of the machine epsilon). We use $\mathcal{E}$ as the set of all integers from 1 to 1024, and $k = 21$ by default. We also construct the PLA-index [2] for $\varepsilon \in \{32, 64\}$ for all genomes, using $k = 21$ as in previous works [2, 17]. Full details, including the accession number of the genomes, their CaPLa values, PLA-index sizes, and other associated information are available at our GitHub repository.

Runtime. For runtime analysis, we used a machine with a 2.10 GHz Intel Xeon CPU E5-2683 v4 processor with 512 GB of memory. To give a sense of the execution time of our pipeline, for a RefSeq genome of median length (GCF_001870125, 2.98M bases), it takes 0.318 s to construct the suffix array, 730.979 s to compute the values $\{b(\varepsilon)\}_{\varepsilon\in\mathcal{E}}$, and 0.006 seconds to run the algorithm of Theorem 10. Thus, the runtime of the algorithm of Theorem 10 is negligible and we do not investigate it further. A more detailed analysis of the runtime of computing $b(\varepsilon)$ values is covered in other papers (e.g. [2]); since it is not a contribution of our work, we do not expand on those studies here.

CaPLa comparison. At first glance, comparing CaPLa values across genomes may seem challenging, as α^* has a larger impact on the PL-approximability than β^*_{low} or β^*_{high}, due to its presence in the exponent of ε in Equation 1. However, the values are easier to compare when the user has a desired query time, i.e. when we think of ε as being fixed. In this case, we define the *ε-mapped CaPLa* as the following two values: $\rho_{\text{low}} = \alpha + \log_\varepsilon \beta_{\text{low}}$ and $\rho_{\text{high}} = \alpha + \log_\varepsilon \beta_{\text{high}}$. The PL-approximability power-law bound (Equation 1) can then be rewritten in terms of ρ_{low} and ρ_{high}, i.e. $\varepsilon^{\rho_{\text{low}}} \leq n/b(\varepsilon) \leq \varepsilon^{\rho_{\text{high}}}$. This replaces the three variables of CaPLa with two variables which, moreover, are now directly comparable between genomes. We can further average ρ_{low} and ρ_{high} to get the *averaged ε-mapped CaPLa*, which becomes useful for looking at CaPLa distributions across large datasets.

Table 1 CaPLa of the RefSeq dataset and of six additional genomes. The first row corresponds to all of the RefSeq dataset, while the later four rows separate the dataset into Kingdoms. Note that percentiles are not reported for the last six rows because each row corresponds to a single genome.

	α^*			β^*_{low}			β^*_{high}		
Category	5th%	Mean	95th%	5th%	Mean	95th%	5th%	Mean	95th%
RefSeq dataset	1.03	1.09	1.16	4.8	7.4	8.6	7.1	9.0	11.1
Virus	1.06	1.12	1.19	3.4	5.6	7.9	7.1	8.6	10.8
Bacteria	1.03	1.08	1.12	6.4	7.8	8.6	7.0	9.1	11.2
Archaea	1.03	1.07	1.12	6.7	7.8	8.6	7.2	8.9	10.7
Fungi	1.08	1.12	1.17	6.5	8.2	9.0	7.3	10.3	12.5
C. elegans		1.06			7.4			8.2	
P. yoelii		1.00			5.5			6.1	
M. commoda		1.06			7.4			8.2	
O. lucimarinus		1.06			7.4			8.1	
Human		1.02			7.1			8.0	
Gorilla		1.00			7.1			7.9	

5.2 Understanding the scale of CaPLa

Table 1 shows the CaPLa values for our data. But, to make sense of the variability, we need to first understand how variations in CaPLa values translate into variations in the space of downstream data structures. The theoretical predictions of Theorem 11 allow such an analysis for the case of the PLA-index. To simplify the analysis, we consider the case that PL-approximability follows a perfect power law, i.e. there is a β^* such that $\beta^* = \beta^*_{\text{low}} = \beta^*_{\text{high}}$. As Theorem 11 and Figure 4 indicated, the total bits per k-mer is approximately $\frac{\gamma_\varepsilon}{\beta^* \varepsilon^{\alpha^*}}$. When α^* increases by an additive factor of δ, then the approximate space decreases by a multiplicative factor of $1 - \varepsilon^{-\delta}$; when β^* increases by a multiplicative factor of δ, then the approximate space decreases by a multiplicative factor of $\frac{\delta}{\delta+1}$. Table 2 shows these numbers for some concrete values that we encounter in our data. For example, in the bacterial genomes, the α^* values increase by 0.1 from the 5th and 95th percentiles (Table 1); Table 2 shows that this corresponds to a decrease in space of 32% for the PLA index with $\varepsilon = 64$. Thus the variability in CaPLa across RefSeq corresponds to a substantial variability in predicted bits per k-mer of the associated PLA-index.

5.3 Variation of CaPLa among the tree of life

To better understand the distributions of CaPLa among different kingdoms, Figure 5AC shows the distributions of the averaged ε-mapped CaPLa. The fungi kingdom has a noticeably higher CaPLa. The difference is substantial, as is reflected in the lower PLA-index space

Table 2 The effect of CaPLa on PLA-index space. We show the decrease in bits per distinct k-mer when adding δ to α (left columns) and multiplying β by $(1+\delta)$ (right columns). The numbers without parenthesis indicate the predicted decrease using $\frac{\gamma_\varepsilon}{\beta^* \varepsilon \alpha^*}$, the leading term of Equation 4. The numbers in parenthesis indicate the change predicted by the full form of Theorem 11, using $C = 6$ and the real values from human chromosome 1 with $k = 21$ ($n = 195,735,278$, $\alpha^* = 1.0309$, $\beta^*_{\text{low}} = 7.1645$, and $\beta^*_{\text{high}} = 8.0119$, $\gamma_{63} = 1.13$ and $\gamma_{32} = 1.13$).

	$\alpha \leftarrow \alpha + \delta$			$\beta \leftarrow \beta(1+\delta)$	
	$\delta = 0.01$	$\delta = 0.05$	$\delta = 0.1$	$\delta = 0.1$	$\delta = 0.5$
$\varepsilon = 32$	3% (3%)	16% (15%)	29% (28%)	9% (9%)	33% (32%)
$\varepsilon = 64$	4% (4%)	19% (18%)	34% (32%)	9% (9%)	33% (32%)
$\varepsilon = 1024$	7% (6%)	29% (28%)	50% (48%)	9% (9%)	33% (32%)

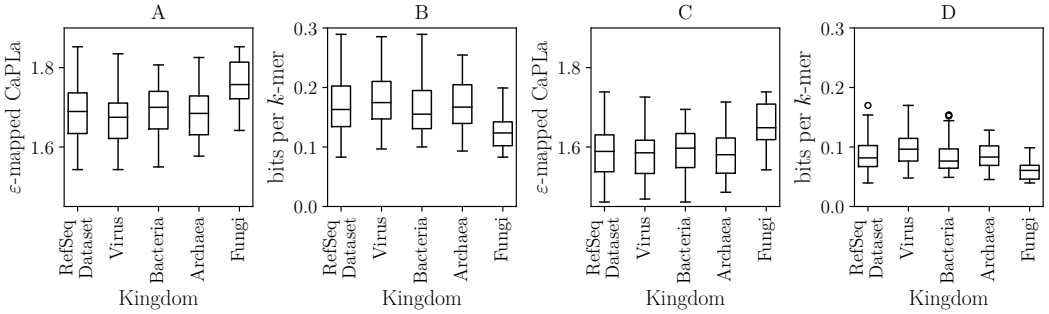


Figure 5 Distribution of averaged ε-mapped CaPLa across RefSeq. Panels A and C show the values for $\varepsilon = 32$ and $\varepsilon = 64$, respectively. Panels B and D show the distribution of bits per distinct k-mer in the constructed PLA indexes. In each panel, the left box plot is for our whole RefSeq dataset, while the remaining four box plots correspond to subsets according to Kingdoms.

usage of fungi genomes compared to the other kingdoms (Figure 5BD). The reasons for this could be due to biological factors but could also be due to biases in the type of genomes that have been sequenced and/or deposited into RefSeq.

Figure 6 shows the space usage of the PLA index and breaks down the relationship between bits per distinct k-mer and the CaPLa separately for each genome. The variation is substantial, with space differences as big as threefold between different RefSeq genomes. Furthermore, the figure shows a strong correlation between CaPLa and bits per distinct k-mer of the PLA-index.

We also split the human genome into its 24 constituent chromosome sequences. Figure 7 shows that CaPLa varies substantially across the chromosomes, indicating that PL-approximability varies not only between different species but also within the genome of a single species. Figure 7 also shows a similar pattern for the chromosomes of the Gorilla genome. Chromosome Y, which is known for being highly repetitive, is an outlier in both species, with substantially lower CaPLa than other chromosomes.

In spite of the CaPLa variability, our results give a general range of CaPLa for genomes that are not part of our dataset. This gives a user with a new genome sequence a robust estimate of the expected efficiency of a PLA-based learned index. This is in contrast with datasets from other domains that have been shown to roughly fit a power-law curve with significantly different α values, such as $\alpha = 1.8$ on book popularity data, $\alpha = 1.4$ on Facebook user IDs, and $\alpha = 2$ on random gaps [9].

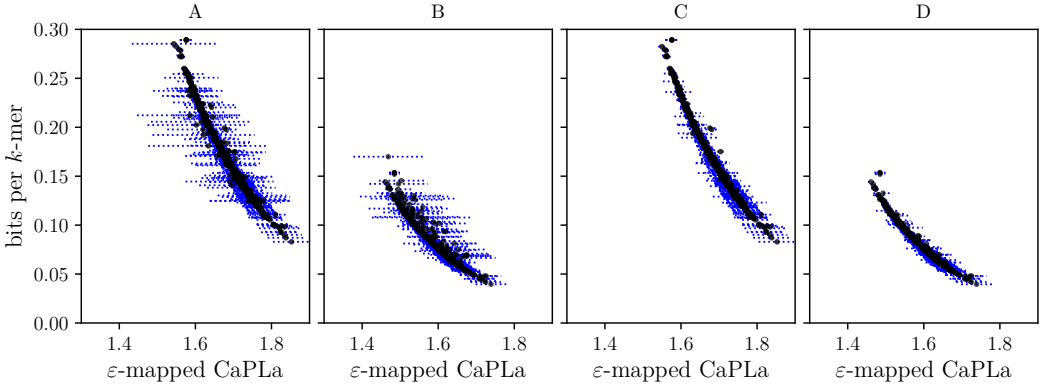


Figure 6 Relation between the bits per distinct k-mer of the PLA-index to the ε-mapped CaPLa for the RefSeq dataset. Panel A shows the result for $\varepsilon = 32$ and Panel B for $\varepsilon = 64$. Each dashed horizontal line represents the range of the ε-mapped CaPLa for a single genome and each black dot represents an averaged ε-mapped CaPLa for a single genome. Panels C and D show the same information as panels A and B, but we additionally filter out all genomes that are shorter than $80,000$nt. For such shorter genomes, the constant overhead of the PLA-index data structures starts to dominate the part that makes use of the PL-approximation; as the plot shows, most of the widest horizontal lines in panels A and B are gone after filtering out the shorter genomes.

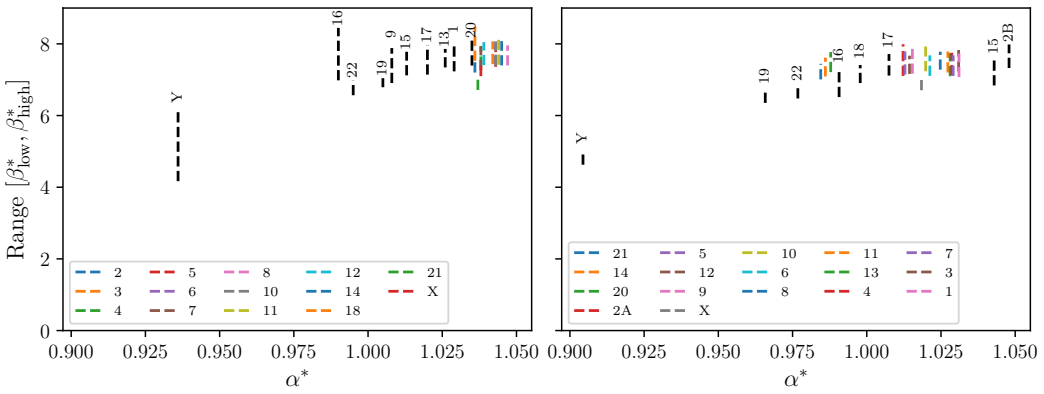


Figure 7 CaPLa of Human (left) and Gorilla (right) chromosomes. Each vertical segment is labeled with the corresponding chromosome, either in the legend or directly above the segment.

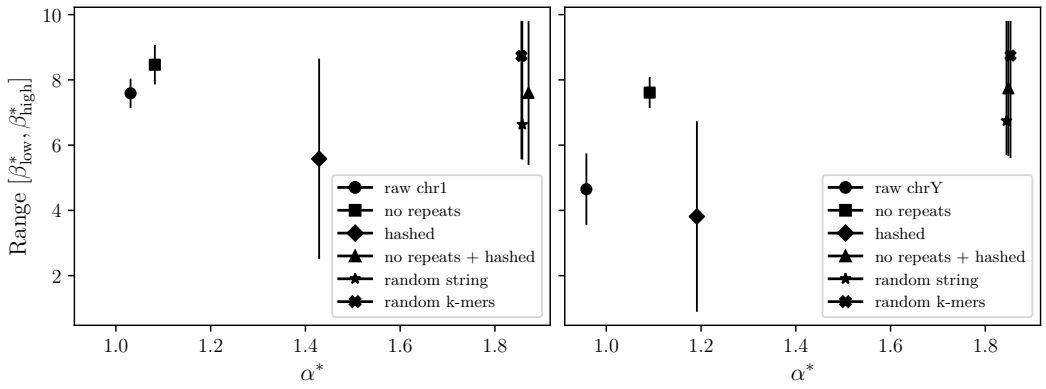


Figure 8 PL-approximability of a genome spectrum versus a random k-mer multisets. The left and right panels show human chromosomes 1 and Y, respectively, for $k = 21$. The y-axis shows the range from β^*_{low} to β^*_{high}. The random string and the random k-mers are generated so as to match the number of k-mers in the raw sequence.

5.4 What makes a genome spectrum special?

Figure 8 shows a huge difference between the CaPLa of human chromosome 1 ($\alpha^* = 1.03$) and a random multiset of k-mers of the same size ($\alpha^* = 1.86$). Theorem 11 predicts a 26-fold difference in the size of the corresponding PLA indexes, at $\varepsilon = 64$. We therefore wanted to tease out the effect of several factors which distinguish a genome spectrum from a random multiset. The first factor is that the k-mers of a genome spectrum must be overlapping, i.e. not all multisets of k-mers can be put together into a continuous genome sequence, even allowing for chromosome breaks. To test the effect of this, we create a random string with the same length as chr1 and find that it also had $\alpha^* = 1.86$. Thus, this factor does not seem to play a role, at least in isolation.

A second effect is that chr1 is not random but is evolved to have a biological function. One consequence is that the genome has k-mers which appear more than once, which is rare in a random string of the same length (for $k = 21$). To test the effect of this, we make a linear scan of chr1 and clip out any encountered k-mer which we have previously seen. In this way, we create a genome-like string which is nearly repeat-free. We found $\alpha^* = 1.08$ for this low-repeat string, suggesting that repeats do negatively affect the PL-approximability.

Another consequence of having a biological function is that the spectrum of a genome has a non-uniform distribution. To test the effect of this, we process the k-mers of chr1 with a hash function and use the hashed value as the basis for the rank function. This has the effect of making the distribution uniform. The resulting $\alpha^* = 1.43$ shows that the non-uniform distribution of the k-mers in a spectrum substantially decreases the PL-approximability.

Finally, we combined the two effects by removing repeats and then hashing the k-mers, getting an $\alpha^* = 1.87$, the same as a random multiset. This indicates that the two factors have an interplay which drives down the PL-approximability a lot when combined.

We repeated the same experiments on the Y chromosome (Figure 8). The effects were similar, though the effect of removing repeats was more pronounced (α^* rose from 0.96 to 1.09). This was expected, as the Y is known to be more repetitive than chromosome 1 [30].

5.5 Stability of CaPLa as a measure

Uniqueness of pinch point. We conducted brute-force tests to investigate whether in our data the pinch point is unique (i.e. CaPLa is well defined), and to verify whether the unimodality assumption required by our algorithm in Theorem 10 holds in practice. Specifically, for each genome in the RefSeq dataset, we sampled the twisted ribbon width $W(\alpha)$ at uniformly spaced α-values within the interval $[\alpha_L, \alpha_H]$ containing a pinch point (Theorem 9), using a step size of 10^{-6}. We then checked for the presence of multiple minima within a numerical tolerance of $\tau = 1.49 \times 10^{-8}$, and verified unimodality by looking for a decreasing and increasing trend before and after the minimum, respectively.

We found that the pinch point was unique in all of the 513 RefSeq genomes. Furthermore, the unimodality assumption was not met in only 11 datasets. Even in those 11 cases, the algorithm of Theorem 10 avoided local minima and matched the brute-force pinch point in all but one instance, where the actual pinch point was nonetheless close (within 3.23×10^{-2}). In summary, up to the tested step size, CaPLa was defined for all genomes and our efficient algorithm returned the correct value in all but one case.

Genome length. A desirable property of a measure of PL-approximability is that it is not causally dependent on genome length and is only an intrinsic property of a genome's rank curve. To verify this, we compare the lengths of our RefSeq genomes with their α^*, β^*_{low}, and β^*_{high} values, respectively, as well as to the ε-mapped CaPLa, for $\varepsilon \in \{2, 4, 8, \ldots, 1024\}$. The Pearson correlation coefficient (R^2) is less than 0.02 for all comparisons, providing evidence that the relationship between CaPLa and genome length is negligible.

***k*-mer size.** The structure of the rank curve is unavoidably dependent on k, because for small k, the number of repeats and the density of S with respect to the universe gets high. Ideally, when k is above the threshold where spurious repeats are rare, CaPLa should stabilize. To validate this, we looked at the behavior of CaPLa with respect to k for the RefSeq dataset and for bigger mammalian genomes. Figure 9 shows six genomes, which we manually selected to capture the phylogenetic diversity of the data. We measure the averaged ε-mapped CaPLa for $\varepsilon = 64$. The plots show that indeed CaPLa plateaus at some point between $k = 16$ and $k = 21$, depending on the genome. The difference between the averaged ε-mapped CaPLa values at $k = 26$ and $k = 31$ was less than 0.01 for all six genomes, both for $\varepsilon = 64$ (Figure 9) and $\varepsilon = 32$ (data not shown). Similarly, between $k = 21$ and $k = 26$, the differences in α^* values were less than 0.4% and the difference in the β^*_{low} and β^*_{high} values were less than 4%. These results provide strong evidence that the CaPLa is relatively stable for large enough values of k, and, in particular, for values of $k \geq 21$.

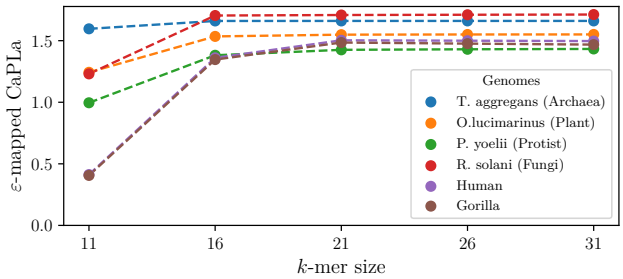


Figure 9 Effect of k-mer size on CaPLa, for $\varepsilon = 64$, using averaged ε-mapped CaPLa

Accuracy of prediction. We compare the space range predicted by plugging in CaPLa into Theorem 11 with the true space taken by the PLA-index (Figure 10). In the majority of cases, the true space falls within the predicted range, as expected. However, for most cases of $\varepsilon = 1024$ and one case of $\varepsilon = 16$, the true space is higher than the predicted range. This is likely due to one of two reasons: 1) the PLA-index uses an implementation of Elias-Fano encoding that has a constant overhead per character, which is not the theoretically optimal data structure used for theoretical analysis, and 2) the results of Theorem 11 hold only asymptotically and do not account for constant factors that contribute to the space, especially for higher ε where the total space is small.

We also compare the true space with the space predicted by fitting a single power-law model to the tabulated PLA-sizes (as we previously discussed in Figure 2) and plugging the fitted parameters into Theorem 1 from [2], with $c = 4$. As Figure 10 shows, the fit-based predictions are often significantly under-predicting the true space, especially at $\varepsilon = 16$.

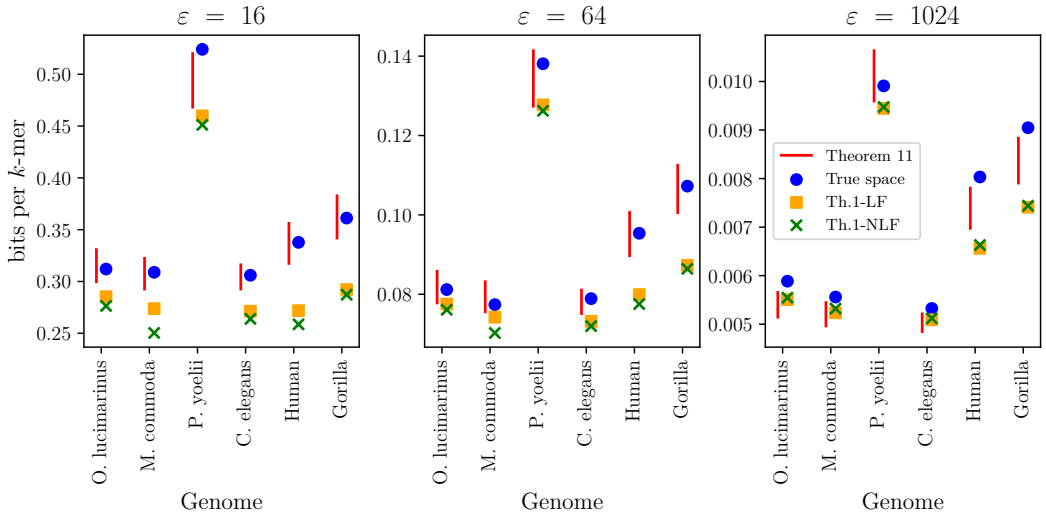


Figure 10 The true space of the PLA-index compared with various forms of predictions, for $\varepsilon \in \{16, 64, 1024\}$. The true space is measured using a version of the PLA-index implementation which encodes the X-array using Elias-Fano, which is the version that is theoretically analyzed in [2] and in Theorem 11. The predicted range (red) corresponds to plugging in the CaPLa into Theorem 11, using $c = 7$. The Th.1-LF and Th.1-NLF correspond to fitting a power-law model with a linear and non-linear fit, respectively, and plugging in the fitted parameters into Theorem 1 in [2] (with $c = 4$). The fitted and non-fitted models are described in the caption of Figure 2.

6 Conclusion

In this paper, we introduced the notion of PL-approximability of a multiset and a corresponding measure called CaPLa (Section 2). Our theoretical results show that CaPLa is well-defined and can be computed efficiently, under reasonable assumptions which we demonstrate hold in almost all tested data (Section 3). Our experimental results show that CaPLa varies substantially not only across genomes and kingdoms, but also within individual genomes, and this variability can lead to differences of up to 50% in the space required by a PLA-based data structure (Sections 5.2 and 5.3). Within our CaPLa framework, the key factors distinguishing a genome spectrum from a random k-mer multiset are the presence of

repeats and the non-uniform distribution of k-mers. The combined effect of these two factors substantially reduces PL-approximability (Section 5.4). CaPLa has several desirable properties, namely, it is well-defined and efficient to compute across our experimented genomes, it is not trivially correlated with genome length thus indicating it captures the intrinsic properties of the genome, and it is robust with respect to k-mer size when $k \geq 21$ (Section 5.5).

Our result has a concrete application to parameter selection in PLA-based indexes such as the PLA-index. When a target space is specified, an approximate estimate of the dataset's CaPLa can be used with Theorem 11 to compute a suitable value for the tuning parameter ε. This is particularly advantageous when the CaPLa is known for a similar dataset (e.g., from another individual of the same species), as it avoids the need to recompute the full set of $(\varepsilon, b(\varepsilon))$ pairs. In the absence of prior CaPLa information, future work could also explore efficient estimation of CaPLa over a small sample of the dataset itself (e.g., a contiguous region of the suffix array).

Our paper presents several future directions. From an application perspective, it will be interesting to connect the variability of PL-approximability to biological properties. Our preliminary results show that PL-approximability is higher in fungi and lower in repeat-rich chromosomes like the Y, but this raises more questions than it answers. From a theory perspective, it would be interesting to find the conditions under which the pinch point is unique. In our experiments, we could not find any instances of non-unique pinch points, suggesting that a theoretical proof may be possible. Finally, Theorem 9 requires that $1 \in \mathcal{E}$ in order to guarantee the existence of a pinch point. This condition is necessary, as in fact one can prove that a pinch point does not exist if $1 \notin \mathcal{E}$. This raises an interesting open question if there is some alternative notion of CaPLa that can be used when $1 \notin \mathcal{E}$.

References

1 Md. Hasin Abrar and Paul Medvedev. PLA-complexity of k-mer multisets. *bioRxiv*, 2024. doi:10.1101/2024.02.08.579510.

2 Md. Hasin Abrar and Paul Medvedev. PLA-index: A k-mer Index Exploiting Rank Curve Linearity. In *Proc. 24th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 312 of *LIPIcs*, pages 13:1–13:18, 2024. doi:10.4230/LIPIcs.WABI.2024.13.

3 Md. Hasin Abrar, Paul Medvedev, and Giorgio Vinciguerra. Efficiency of learned indexes on genome spectra, 2025. doi:10.1101/2025.07.10.664199.

4 Domenico Amato, Giosuè Lo Bosco, and Raffaele Giancarlo. Neural networks as building blocks for the design of efficient learned indexes. *Neural Computing and Applications*, 35(29):21399–21414, 2023. doi:10.1007/S00521-023-08841-1.

5 Naiyong Ao, Fan Zhang, Di Wu, Douglas S. Stones, Gang Wang, Xiaoguang Liu, Jing Liu, and Sheng Lin. Efficient parallel lists intersection and index compression algorithms using graphics processing units. *PVLDB*, 4(8):470–481, 2011. doi:10.14778/2002974.2002975.

6 Rayan Chikhi, Jan Holub, and Paul Medvedev. Data structures to represent a set of k-long DNA sequences. *ACM Computing Surveys*, 54(1):1–22, 2021. doi:10.1145/3445967.

7 Rayan Chikhi, Brice Raffestin, Anton Korobeynikov, Robert Edgar, and Artem Babaian. Logan: Planetary-scale genome assembly surveys life's diversity. *bioRxiv*, July 2024. doi:10.1101/2024.07.30.605881.

8 Paolo Ferragina, Hans-Peter Lehmann, Peter Sanders, and Giorgio Vinciguerra. Learned monotone minimal perfect hashing. In *Proc. 31st Annual European Symposium on Algorithms (ESA)*, volume 274 of *LIPIcs*, pages 46:1–46:17, 2023. doi:10.4230/LIPICS.ESA.2023.46.

9 Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. On the performance of learned data structures. *Theoretical Computer Science*, 871:107–120, 2021. doi:10.1016/j.tcs.2021.04.015.

10 Paolo Ferragina and Giorgio Vinciguerra. Learned data structures. In Luca Oneto, Nicolò Navarin, Alessandro Sperduti, and Davide Anguita, editors, *Recent Trends in Learning From Data*, pages 5–41. Springer International Publishing, 2020. doi:10.1007/978-3-030-43883-8_2.

11 Paolo Ferragina and Giorgio Vinciguerra. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *PVLDB*, 13(8):1162–1175, 2020. doi:10.14778/3389133.3389135.

12 Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. FITing-Tree: A data-aware index structure. In *Proc. 45th International Conference on Management of Data (SIGMOD)*. ACM, 2019. doi:10.1145/3299869.3319860.

13 Darryl Ho, Saurabh Kalikar, Sanchit Misra, Jialin Ding, Vasimuddin Md, Nesime Tatbul, Heng Li, and Tim Kraska. LISA: Learned indexes for sequence analysis. *bioRxiv*, 2021. doi:10.1101/2020.12.22.423964.

14 Youngmok Jung and Dongsu Han. BWA-MEME: BWA-MEM emulated with a machine learning approach. *Bioinformatics*, 38(9):2404–2413, 2022. doi:10.1093/bioinformatics/btac137.

15 Saurabh Kalikar, Chirag Jain, Md Vasimuddin, and Sanchit Misra. Accelerating minimap2 for long-read sequencing applications on modern CPUs. *Nature Computational Science*, 2(2):78–83, 2022. doi:10.1038/s43588-022-00201-8.

16 Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. RadixSpline: a single-pass learned index. In *Proc. 3rd International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM)*, pages 5:1–5:5, 2020. doi:10.1145/3401071.3401659.

17 Melanie Kirsche, Arun Das, and Michael C Schatz. Sapling: Accelerating suffix array queries with learned data models. *Bioinformatics*, 37(6):744–749, 2021. doi:10.1093/BIOINFORMATICS/BTAA911.

18 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proc. 45th International Conference on Management of Data (SIGMOD)*. ACM, 2018. doi:10.1145/3183713.3196909.

19 Meng Li, Huayi Chai, Siqiang Luo, Haipeng Dai, Rong Gu, Jiaqi Zheng, and Guihai Chen. VEGA: an active-tuning learned index with group-wise learning granularity. *Proc. ACM Manag. Data*, 3(1):86:1–86:26, 2025. doi:10.1145/3709736.

20 Liang Liang, Guang Yang, Ali Hadian, Luis Alberto Croquevielle, and Thomas Heinis. SWIX: A memory-efficient sliding window learned index. *Proc. ACM Manag. Data*, 2(1):41:1–41:26, 2024. doi:10.1145/3639296.

21 Chaohong Ma, Xiaohui Yu, Yifan Li, Xiaofeng Meng, and Aishan Maoliniyazi. FILM: a fully learned index for larger-than-memory databases. *PVLDB*, 16(3):561–573, 2022. doi:10.14778/3570690.3570704.

22 Camille Marchet, Christina Boucher, Simon J. Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing data sets. *Genome Research*, 31(1):1–12, 2020. doi:10.1101/gr.260604.119.

23 Paul Medvedev. Theoretical analysis of sequencing bioinformatics algorithms and beyond. *Communications of the ACM*, 66(7):118–125, 2023. doi:10.1145/3571723.

24 Yuta Mori. libdivsufsort. https://github.com/y-256/libdivsufsort/.

25 Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, 2016.

26 Gonzalo Navarro. Indexing highly repetitive string collections, Part I. *ACM Computing Surveys*, 54(2):1–31, 2021. doi:10.1145/3434399.

27 Joseph O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Communications of the ACM*, 24(9):574–578, 1981. doi:10.1145/358746.358758.

28 Giulio Ermanno Pibiri, Yoshihiro Shibuya, and Antoine Limasset. Locality-preserving minimal perfect hashing of k-mers. *Bioinformatics*, 39(Supplement_1):i534–i543, 2023. doi:10.1093/bioinformatics/btad219.

29 William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

30 Arang Rhie, Sergey Nurk, Monika Cechova, Savannah J Hoyt, Dylan J Taylor, Nicolas Altemose, Paul W Hook, Sergey Koren, Mikko Rautiainen, Ivan A Alexandrov, et al. The complete sequence of a human Y chromosome. *Nature*, 621(7978):344–354, 2023. doi:10.1038/s41586-023-06457-y.

31 Geoffrey X. Yu, Markos Markakis, Andreas Kipf, Per-Åke Larson, Umar Farooq Minhas, and Tim Kraska. TreeLine: an update-in-place key-value store for modern storage. *PVLDB*, 16(1):99–112, 2022. doi:10.14778/3561261.3561270.

32 Sepanta Zeighami and Cyrus Shahabi. On distribution dependent sub-logarithmic query time of learned indexing. In *Proc. 40th International Conference on Machine Learning (ICML)*, volume 202 of *PMLR*, pages 40669–40680, 2023. URL: https://proceedings.mlr.press/v202/zeighami23a.html.

On Finding ℓ-Th Smallest Perfect Matchings

Nicolas El Maalouly
Department of Computer Science, ETH Zurich, Switzerland

Sebastian Haslebacher
Department of Computer Science, ETH Zurich, Switzerland

Adrian Taubner
Department of Computer Science, ETH Zurich, Switzerland

Lasse Wulf
Section of Algorithms, Logic, and Graphs, Technical University of Denmark, Lyngby, Denmark
IT University, Copenhagen, Denmark

Abstract

Given an undirected weighted graph G and an integer k, Exact-Weight Perfect Matching (EWPM) is the problem of finding a perfect matching of weight exactly k in G. In this paper, we study EWPM and its variants. The EWPM problem is famous, since in the case of unary encoded weights, Mulmuley, Vazirani, and Vazirani showed almost 40 years ago that the problem can be solved in randomized polynomial time. However, up to this date no derandomization is known.

Our first result is a simple deterministic algorithm for EWPM that runs in time $n^{\mathcal{O}(\ell)}$, where ℓ is the number of distinct weights that perfect matchings in G can take. In fact, we show how to find an ℓ-th smallest perfect matching in any weighted graph (even if the weights are encoded in binary, in which case EWPM in general is known to be NP-complete) in time $n^{\mathcal{O}(\ell)}$ for any integer ℓ. Similar next-to-optimal variants have also been studied recently for the shortest path problem.

For our second result, we extend the list of problems that are known to be equivalent to EWPM. We show that EWPM is equivalent under a weight-preserving reduction to the Exact Cycle Sum problem (ECS) in undirected graphs with a conservative (i.e. no negative cycles) weight function. To the best of our knowledge, we are the first to study this problem. As a consequence, the latter problem is contained in RP if the weights are encoded in unary. Finally, we identify a special case of EWPM, called BCPM, which was recently studied by El Maalouly, Steiner and Wulf. We show that BCPM is equivalent under a weight-preserving transformation to another problem recently studied by Schlotter and Sebő as well as Geelen and Kapadia: the Shortest Odd Cycle problem (SOC) in undirected graphs with conservative weights. Finally, our $n^{\mathcal{O}(\ell)}$ algorithm works in this setting as well, identifying a tractable special case of SOC, BCPM, and ECS.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Design and analysis of algorithms

Keywords and phrases Exact Matching, Perfect Matching, Exact-Weight Perfect Matching, Shortest Odd Cycle, Exact Cycle Sum, l-th Smallest Solution, l-th Largest Solution, k-th Best Solution, Derandomization

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.19

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2506.22619

Funding *Lasse Wulf*: Supported by Carlsberg Foundation CF21-0302 "Graph Algorithms with Geometric Applications"

1 Introduction

A big open question in complexity theory concerns the role of randomness in devising polynomial-time algorithms. While it is conjectured that every problem in RP (and even BPP) should also lie in P (see e.g. Chapter 20 in [1] for more details), there exist several


© Nicolas El Maalouly, Sebastian Haslebacher, Adrian Taubner, and Lasse Wulf;
licensed under Creative Commons License CC-BY 4.0



natural problems for which efficient randomized algorithms, but no efficient deterministic algorithms, are known. Thus, instead of attacking the conjecture RP = P directly, it might be more reasonable to first try to devise efficient deterministic algorithms for those problems.

A famous example of a problem in RP, that is not known to lie in P, is the unary version of Exact-Weight Perfect Matching (EWPM): Given an undirected weighted graph G and an integer k, EWPM is the problem of deciding whether G admits a perfect matching of weight exactly k. Note that this problem is often called Exact Matching (EM) if we allow only weights 0 and 1, and it is not hard to see (by replacing single edges by paths corresponding to their weight) that unary EWPM is equivalent to EM.

EWPM was introduced by Papadimitriou and Yannakakis [27] in 1982, and it is known to be NP-hard in general (for a short proof, see [14]). Papadimitriou and Yannakakis also conjectured that EWPM remains NP-hard if weights are encoded in unary. However, Mulmuley, Vazirani and Vazirani [24] proved in 1987 that unary EWPM is in RP. This makes it very unlikely for the problem to be NP-hard. Hence, the question became whether one could also find an efficient deterministic algorithm for unary EWPM.

This question has been open for almost 40 years now. In fact, despite considerable effort, no deterministic algorithm is known that runs in time that is subexponential in the number of vertices of the graph. Instead, progress has been made mostly by focusing on restricted graph classes, approximation variants of EWPM, and finding other natural problems that are equivalent to unary EWPM (see Section 1.3 for more details).

Our main result is a new parameterized algorithm for EWPM that works on general graphs, even if weights are encoded in binary: Assume that $k_1 < k_2 < \cdots < k_\ell$ are all the distinct weights up to k_ℓ that perfect matchings in G take. We prove that it is possible to find a perfect matching of weight k_ℓ in time $n^{\mathcal{O}(\ell)}$. Our algorithm is very simple and relies on our main theorem (Theorem 9), which says that there must exist $2(\ell-1)$ edges such that any minimum-weight perfect matching that includes all of these $2(\ell-1)$ edges must have weight k_ℓ. The obtained bound of $2(\ell-1)$ is tight.

1.1 Next-to-Optimal Problems

Our main result can be equivalently stated in a different way: Consider the following problem, which we call ℓ-th Smallest Perfect Matching (SPM_ℓ): Given a weighted graph G, an integer k, and a natural number ℓ, decide whether G contains an ℓ-th smallest perfect matching of weight exactly k. Here, the notion of ℓ-th smallest perfect matchings is defined recursively: 1st-smallest perfect matchings are exactly the minimum-weight perfect matchings, and for $\ell \geq 2$, ℓ-th smallest perfect matchings are the perfect matchings of minimum weight among all perfect matchings that are not ℓ'-th smallest for any $\ell' < \ell$. In this language, our result is an algorithm that solves SPM_ℓ in time $n^{\mathcal{O}(\ell)}$.

Similar next-to-optimal problems have been studied in the literature before. In particular, there is a line of work on finding next-to-shortest paths in directed or undirected graphs. In these problems, we want to find a next-to-shortest (s,t)-path in a weighted graph G, i.e. the path should have minimum weight among all (s,t)-paths that are not shortest paths. This problem is NP-hard on directed graphs [22], but admits polynomial-time algorithms on undirected graphs [18, 21, 23, 31, 33]. Note that the NP-hardness result in [22] heavily relies on allowing edges to have weight zero. Indeed, the so-called detour problem – decide whether all (s,t)-shortest paths in a given unweighted directed graph have the same length or not – is very similar but does not allow for zero-weight edges, and its complexity status is still unresolved [15].

Instead of asking for a perfect matching of ℓ-th smallest weight, one could instead ask for ℓ distinct perfect matchings while minimizing their combined weight. This is a conceptually different problem which is equivalent to repeatedly (ℓ times) finding the best solution distinct from previously obtained solutions. Such k-best enumeration problems have been studied for a number of combinatorial optimization problems: Murty [26] first showed a polynomial-time algorithm for the k-best bipartite perfect matching problem. Later, the time complexity was improved by Chegireddy and Hamacher [2]. For more information, we refer the reader to the survey by Eppstein [11].

1.2 Related Cycle Problems

Our second contribution in this paper is to strengthen the connection between EWPM and related cycle problems, as we will outline here.

Consider the following problem, which we call Exact Cycle Sum (ECS): Given an undirected graph G with conservative (i.e. no negative cycles) edge-weights and an integer k, decide whether G contains a set of disjoint cycles of weight exactly k. A directed version of ECS with general weights has been studied e.g. by Papadimitriou and Yannakakis [27]. Concretely, they proved that the directed version of ECS is equivalent to EWPM on bipartite graphs. We extend this result by proving that the undirected problem ECS as defined above is equivalent to EWPM on general graphs. Our reductions preserve the encoding of the weights. Hence, we obtain as a result that ECS is in RP if weights are encoded in unary, and NP-complete if weights are encoded in binary.

Another interesting problem on cycles in graphs is the so-called Shortest Odd Cycle problem (SOC): Given an undirected graph G with conservative edge-weights and an integer k, decide whether G contains a cycle of odd weight at most k. SOC was recently studied by Schlotter and Sebő [28] (they use slightly different but equivalent definition, see Section 2.3 for more details), who proved that it is equivalent to finding minimum-weight odd T-joins. They also point out that SOC is contained in RP for unary weights, due to a result by Geelen and Kapadia [13], and therefore conjecture that the problem should be contained in P as well. Interestingly, the algorithm in [13] follows from a reduction to a problem on perfect matchings that can then be solved using the techniques of Mulmuley, Vazirani, and Vazirani [24]. In fact, this problem on perfect matchings has independently been studied by El Maalouly, Steiner, and Wulf [8] who called it Bounded Correct-Parity Perfect Matching (BCPM): Given a weighted graph G and an integer k, decide whether G contains a perfect matching M of weight $w(M)$ at most k and with $w(M) \equiv_2 k$ (where $\equiv_2$ is used to denote mod 2 equivalence). El Maalouly, Steiner, and Wulf use BCPM on bipartite graphs as a subroutine for a parameterized algorithm for EM on bipartite graphs. To this end, they prove that BCPM on bipartite graphs is in P, by reducing it to the directed version of SOC (which can be solved via dynamic programming). In this paper, we extend this set of results by proving that BCPM on general graphs is equivalent to SOC as defined above. Unfortunately, we are not able to give deterministic polynomial-time algorithms for either problem, but any such deterministic polynomial-time algorithm for SOC or BCPM would immediately imply that the parameterized algorithm from [8] also works on general graphs. Furthermore, our results highlight that the two equivalent problems SOC and BCPM can be seen as a stepping stone for solving unary EWPM, and that it might be reasonable to tackle them first.

Concretely, some interesting further questions that remain unresolved are: Can we solve SOC and BCPM in deterministic polynomial time? The conjecture RP = P suggests that for unary weights, one should be able to do it. Interestingly, we know very little about the case of weights encoded in binary: If the weights are encoded in binary, it is not known whether SOC or BCPM are in RP or NP-hard (since the randomized algorithm in [13] works only for unary weights). Note that the same questions were also asked by Schlotter and Sebő [28].

While in the general case, derandomization of unary SOC and BCPM remains open, combining our algorithm for EWPM with our reductions also implies that BCPM, SOC, and ECS can be deterministically solved in time $n^{O(\ell)}$ if at most ℓ many different possible weights of perfect matchings (cycle sums, respectively) appear in the instance.

1.3 Further Related Work on EM

As mentioned before, EWPM with weights restricted to 0 and 1 is called the Exact Matching Problem (EM). Derandomizing the known randomized algorithm is an important open problem. Solution approaches have mainly been focused on three directions: Finding deterministic approximation algorithms, finding deterministic algorithms on restricted graph classes, and identifying other polynomial-time equivalent problems.

- In terms of approximation algorithms for EM, Yuster [32] proved that one can find an almost exact matching (a matching of weight exactly k that may fail to cover two vertices of the graph) in deterministic polynomial time if an exact matching exists in the graph. One could interpret this as an approximation result that relaxes the perfect matching constraint. If we relax the exactness constraint instead, recent progress includes a two-sided 2-approximation algorithm on bipartite graphs by El Maalouly [5], and a one-sided 3-approximation algorithm on bipartite graphs by Dürr, El Maalouly, and Wulf [3].
- EM has been studied on various restricted graph classes. In fact, Karzanov [19] proved already in 1987 that EM can be solved in deterministic polynomial time on complete and complete bipartite graphs. This has been generalized in different ways: El Maalouly and Steiner [7] gave deterministic polynomial-time algorithms on graphs with bounded independence number (and bipartite graphs of bounded bipartite independence number). In the case of bipartite graphs, this was improved to an FPT-algorithm by El Maalouly, Steiner, and Wulf [8]. In the general case, Murakami and Yamaguchi [25] gave an FPT-algorithm that is paramatrized by the independence number and the minimum size of an odd cycle transversal.

  El Maalouly, Haslebacher, and Wulf [6] used Karzanov's original techniques to prove that EM restricted to chain graphs, unit-interval graphs, and complete r-partite graphs is in P. They also showed that a local search algorithm works in deterministic polynomial time on complete r-partite graphs, graphs of bounded neighborhood diversity, and graphs without complete bipartite t-holes.

  Finally, it is also possible to derandomize the randomized algorithm on some graph classes, e.g. by finding so-called Pfaffian orientations. Concretely, Vazirani [30] showed how to do this for $K_{3,3}$-free graphs, and Galluccio and Loebl [12] obtained Pfaffian orientations for graphs of bounded genus.
- Previously, some effort has also been made to identify other natural problems that are polynomial-time equivalent to EM. We already mentioned some of these results in Section 1.2. Besides the mentioned cycle problems, EM is also known to be equivalent to a problem called Top-k Perfect Matching [9].
- Finally, Jia, Svensson, and Yuan [16] recently proved that the bipartite exact matching polytope has exponential extension complexity. This stands in contrast to the bipartite perfect matching polytope, which is known to be integral.

We remark that many of these algorithmic results only hold for 0-1-weighted EWPM, and do not even translate to unary EWPM since they often involve very specific graph structures or parameters. In contrast, our parameterized algorithm works even for binary encoding of weights in EWPM.

1.4 Outline

We formally define EWPM and related problems in Section 2, where we also recall some of the most important concepts from the literature. In Section 3, we introduce the ℓ-th Smallest Perfect Matching (SPM$_\ell$) problem, and we prove that it can be solved deterministically in time $n^{\mathcal{O}(\ell)}$. In Section 4, we explain our reductions between cycle and matching problems. Full proofs of all statements in Section 4 can be found in the Appendix of the full version.

2 Preliminaries

We start by introducing notation and afterwards recall important facts from the literature. First of all, all our graphs are considered to be simple, i.e. they do not contain self-loops or parallel edges. If not specified, graphs are also considered to be undirected. Now, given a graph $G = (V, E)$, we will frequently use the standard notions of paths, cycles, and matchings, and we usually think of those objects as subsets of edges. In particular, a perfect matching M of G is a subset of edges $M \subseteq E$ such that every vertex of G is covered exactly once by M. We denote by $\mathcal{M}_G$ the set of all perfect matchings of G.

Occasionally, we will explicitly use directed graphs. In particular, we will write $(u, v) \in E$ (instead of $\{u, v\} \in E$) for directed edges in a directed graph $G = (V, E)$ with $u, v \in V$.

In some problems, edge-weights are given for a graph $G = (V, E)$ in terms of a function $w : E \to \mathbb{Z}$ or $w : E \to \mathbb{N}$. We use the convention that $w(X)$ denotes the sum of all weights of edges in $X \subseteq E$.

For two decision problems A and B, we use $A \leq_p B$ to denote that there exists a polynomial-time many-one reduction from A to B. We also write $A \equiv_p B$ to say that both $A \leq_p B$ and $B \leq_p A$ are true.

2.1 Symmetric Difference and Alternating Cycles

An important and common tool for working with perfect matchings is the *symmetric difference*. The symmetric difference of two sets X, Y is defined as $X \Delta Y \coloneqq (X \cup Y) \setminus (X \cap Y)$. It is a well-known fact that the symmetric difference $M \Delta M'$ of two perfect matchings M, M' in a graph $G = (V, E)$ always consists of a set of vertex-disjoint even cycles. In other words, there exist vertex-disjoint even cycles $C_1, \dots, C_p \subseteq E$ such that $M \Delta M' = \bigsqcup_{i=1}^{p} C_i$. We say that C_i is a cycle from $M \Delta M'$, and we sometimes use the letter p to refer to the number of cycles in $M \Delta M'$. Further, we recall that every cycle C in $M \Delta M'$ is alternating in both M and M' (an even cycle C of G is called *M-alternating* if and only if it alternates between edges in M and edges not in M). The set $M \Delta C$ is again a matching of the same cardinality as M. We often refer to this property by saying that we *switch* M along C to obtain the new matching $M \Delta C$. Naturally, the matchings $M \Delta C$ and M differ exactly in C, i.e. $(M \Delta C) \Delta M = C$.

Recall again that $M \Delta M'$ is a collection of vertex-disjoint even cycles $C_1, \dots, C_p \subseteq E$, and that each of the cycles $C_1, \dots, C_p$ is both M-alternating and M'-alternating. Hence, given both M, M', some new perfect matchings of G can be obtained by switching either M or M' along a subset of the cycles $C_1, \dots, C_p$. In particular, we can perform the switching operations independently of each other because the cycles are vertex-disjoint.

2.2 Minimum-Weight and Exact-Weight Perfect Matching

Finding a minimum-weight perfect matching is a classical problem that has been studied extensively. We will define it as follows.

▶ **Definition 1** (MWPM). *Given a graph $G = (V, E)$, edge-weights $w : E \to \mathbb{Z}$, and an integer k, Minimum-Weight Perfect Matching (MWPM) is the problem of deciding whether G admits a perfect matching of total weight at most k.*

It is well-known that MWPM is in P (even if weights are encoded in binary, see e.g. [20]). Moreover, finding a maximum-weight perfect matching is analogous to finding a minimum-weight perfect matching, and hence that can also be done in deterministic polynomial time. Since both of these problems are in P, it then seems natural to ask whether we can also test for a perfect matching of weight exactly k in polynomial time.

▶ **Definition 2** (EWPM). *Given a graph $G = (V, E)$, edge-weights $w : E \to \mathbb{Z}$, and an integer k, Exact-Weight Perfect Matching (EWPM) is the problem of deciding whether G admits a perfect matching of weight exactly k.*

As mentioned in Section 1, the variant of EWPM with weights from $\{0, 1\}$ is also known as Exact Matching (EM), and it is contained in RP [24]. Moreover, this also implies that EWPM with weights encoded in unary is contained in RP. In contrast, EWPM with weights encoded in binary is NP-complete [27, 14]. Note that this constitutes a major difference to MWPM, where both the unary and binary variant admit efficient algorithms.

The following problem BCPM is at most as hard as EWPM, and it was used in [8] as part of an algorithm for EM.

▶ **Definition 3** (BCPM). *Given a graph $G = (V, E)$, edge-weights $w : E \to \mathbb{Z}$, and an integer k, Bounded Correct-Parity Perfect Matching (BCPM) is the problem of deciding whether G admits a perfect matching M of weight $w(M)$ at most k satisfying $k \equiv_2 w(M)$.*

The randomized algorithm for unary EWPM also implies a randomized algorithm for unary BCPM. Interestingly, the complexity of binary BCPM remains open. In particular, it is not known whether binary BCPM is NP-hard or not.

2.3 Conservative Weights and Cycle Problems

Many problems on finding shortest paths and cycles in graphs with non-negative edge-weights can be solved efficiently. On the other hand, allowing for negative weights usually makes those problems NP-hard as they now include problems such as finding Hamiltonian paths or cycles. An interesting compromise is provided by conservative edge-weights.

▶ **Definition 4** (Conservative Weights). *Edge-weights $w : E \to \mathbb{Z}$ of a (possibly directed) graph $G = (V, E)$ are conservative if and only if there are no negative (directed) cycles in G with respect to w.*

Using established techniques (see e.g. Johnson's algorithm [17] or Suurballe's algorithm [29]), conservative weights on directed graphs can be turned into non-negative weights while maintaining the total weight of each cycle. Hence, cycle problems in directed graphs with conservative edge-weights are usually not harder than their non-negative variants. However, in the case of undirected graphs, the same technique is not applicable, and it turns out that there are interesting cycle problems on undirected graphs with conservative edge-weights that relate to EWPM and BCPM. Concretely, we will work with the two problems Exact Cycle Sum (ECS) and Shortest Odd Cycle (SOC), defined on graphs with conservative weights.

▶ **Definition 5** (ECS). *Given a graph $G = (V, E)$ with conservative edge-weights $w : E \to \mathbb{Z}$ and an integer k, Exact Cycle Sum (ECS) is the problem of deciding whether there exists a set of vertex-disjoint cycles of total weight exactly k in G.*

▶ **Definition 6** (SOC). *Given a graph $G = (V, E)$ with conservative edge-weights $w : E \to \mathbb{Z}$ and an integer k, Shortest Odd Cycle (SOC) is the problem of deciding whether there exists a cycle of odd weight at most k in G.*

Note that the definition of SOC by Schlotter and Sebő [28] as well as Geelen and Kapadia [13] slightly differs from ours: They require the cycle to have odd length (i.e. an odd number of edges) instead of odd weight. However, it is not hard to see that the two versions are polynomial-time equivalent. Indeed, by splitting even-weight edges, we can ensure to get a graph with only odd-weight edges, yielding a reduction from our odd-weight version to the odd-length version. To reduce the odd-length version to the odd-weight version, it suffices to manipulate the weights, replacing the weight $w(e)$ of each edge e with $2|V|w(e) + 1$.

As mentioned in Section 1.2, El Maalouly et al. [8] gave a polynomial-time algorithm for BCPM on bipartite graphs by reducing it to a directed version of SOC. An analogous connection between EWPM on bipartite graphs and the directed version of ECS has been observed by Papadimitriou and Yannakakis [27] already in 1982. In Section 4, we will extend these connections between cycle and matching problems by proving equivalence of ECS (undirected) and EWPM (on general graphs), and SOC (undirected) and BCPM (on general graphs), respectively.

3 ℓ-th Smallest Perfect Matchings

As discussed in Section 1, the problem of finding a deterministic polynomial-time algorithm for unary EWPM has been attacked from various directions in recent years. We attack it from yet another direction that involves so-called ℓ-th smallest perfect matchings.

▶ **Definition 7.** *Let $G = (V, E)$ be a graph with edge-weights $w : E \to \mathbb{Z}$. We define ℓ-th smallest perfect matchings for all $\ell \in \mathbb{N}$ recursively. The 1st-smallest perfect matchings of G are exactly the minimum-weight perfect matchings. Moreover, for fixed $\ell \geq 2$, a perfect matching M is ℓ-th smallest if and only if it has minimum weight among all perfect matchings that are not ℓ'-th smallest for any $\ell' < \ell$.*

▶ **Definition 8** (SPM_ℓ). *Given a graph $G = (V, E)$, edge-weights $w : E \to \mathbb{Z}$, an integer k, and a natural number ℓ, ℓ-th Smallest Perfect Matching (SPM_ℓ) is the problem of deciding whether G admits an ℓ-th smallest perfect matching of weight exactly k.*

The goal of this section is to give a deterministic algorithm that can decide SPM_ℓ in time $n^{\mathcal{O}(\ell)}$, where n is the size of the instance. Naturally, this has interesting consequences for EWPM: Assume that we are given an instance of EWPM where k is such that any perfect matching M with $w(M) = k$ is an ℓ'-th smallest perfect matching. Then we can verify existence of such a matching in time $n^{\mathcal{O}(\ell')}$ by simply calling the SPM_ℓ-algorithm repeatedly for $\ell = 1, \ldots, \ell'$.

To the best of our knowledge, we are the first to observe a result of this kind. Our algorithm is very simple and follows a natural approach: For a given ℓ and for every set $F \subseteq E$ of size at most $2(\ell - 1)$, compute a minimum-weight perfect matching that includes all of F. We prove in Theorem 9 that there is always a 1st-smallest, a 2nd-smallest, …, an ℓ-th smallest perfect matching among the set of perfect matchings that were computed. The proof of Theorem 9 is less straight-forward. It relies on the complementary slackness theorem (compare e.g. [20]) and the perfect matching polytope.

▶ **Theorem 9** (ℓ-th Smallest PM in General Graphs). *Let $G = (V, E)$ be a graph with edge-weights $w : E \to \mathbb{Z}$. Let $\ell \in \mathbb{N}$ be arbitrary. Assume that G contains an ℓ-th smallest perfect matching M. Then there exists a subset of edges $F \subseteq E$ of size at most $2(\ell - 1)$ such that*

$$w(M) = \min_{M' \in \mathcal{M}_G : F \subseteq M'} w(M').$$

It turns out that one can slightly improve the above theorem in the case of bipartite graphs.

▶ **Theorem 10** (ℓ-th Smallest PM in Bipartite Graphs). *The ℓ-th smallest perfect matching in a weighted bipartite graph can be obtained the same way as in Theorem 9, except that F ranges only over edge sets of size at most $\ell - 1$.*

In this paper, we omit the proof of Theorem 10, since its proof is identical to that of Theorem 9, except that the general matching polytope is swapped with the bipartite matching polytope.

One might ask whether the number of edges one needs to "fix" in these two theorems is tight. Indeed, this is the case, i.e. in order to find the ℓ-th smallest perfect matching, one needs to fix $\ell - 1$ edges in a bipartite graph and $2(\ell - 1)$ edges in a general graph. To see this, consider Figure 1, where dashed edges have a weight of 1 while solid edges have a weight of 0. Consider first the bipartite four-cycle on the left. It has exactly two perfect matchings, one of weight 0 and one of weight 2. Fixing one edge is necessary and sufficient to find the 2nd-smallest perfect matching. Taking multiple disjoint copies of this four-cycle generalizes this to show that it is sometimes necessary to fix $\ell - 1$ edges to find the ℓ-th smallest perfect matching in a bipartite graph. Analogously, the graph on the right has perfect matchings of weight 1 and 3, but fixing a single edge does not suffice to find the perfect matching of weight 3. In other words, we need to fix 2 edges to find the second-smallest perfect matching here. Again, generalizing this argument by taking multiple copies of this graph proves that the bound $2(\ell - 1)$ is best possible in general graphs.

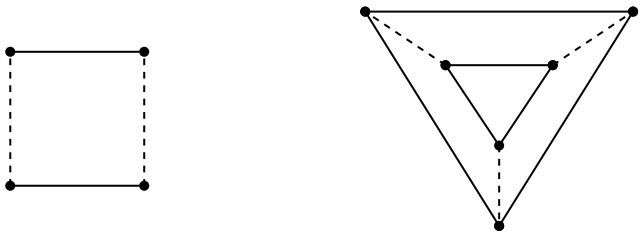


Figure 1 Example that Theorems 9 and 10 are tight in terms of the number of fixed edges. Dashed edges have a weight of 1 and solid edges have a weight of 0.

3.1 Proof of Theorem 9

The goal of this section is to prove Theorem 9. We start by recalling some useful facts about the perfect matching polytope from Chapter 11 of the standard textbook by Korte and Vygen [20]. For this, it will be convenient to use the notation $\delta(v)$ to denote the set of edges incident to a vertex $v \in V$. Similarly, for all $A \subseteq V$, we use $\delta(A)$ to denote the set of edges with one endpoint in A and the other in $V \setminus A$. Moreover, we denote by $\mathcal{A}$ the set of all odd-cardinality sets $A \subseteq V$, i.e. $\mathcal{A} := \{A \subseteq V : |A| \equiv_2 1\}$.

For the remainder of this section, let $G = (V, E)$ be a graph with edge-weights $w : E \to \mathbb{Z}$. The integer linear program

$$\begin{aligned} &\min \sum_{e \in E} x_e w(e) && \\ \text{s.t.} \quad &\sum_{e \in \delta(v)} x_e = 1 && \forall v \in V \\ \text{and } &x_e \in \{0, 1\} && \forall e \in E \end{aligned}$$

with variables $\{x_e \mid e \in E\}$ captures the problem of finding a minimum-weight perfect matching in G. Now consider the following LP relaxation

$$\begin{aligned} &\min \sum_{e \in E} x_e w(e) && \\ \text{s.t.} \quad &\sum_{e \in \delta(v)} x_e = 1 && \forall v \in V \\ &\sum_{e \in \delta(A)} x_e \geq 1 && \forall A \in \mathcal{A} \text{ with } |A| > 1 \\ &x_e \geq 0 && \forall e \in E \end{aligned}$$

where additional odd-set constraints were added. By the famous result of Edmonds [4], adding these additional odd-set constraints is necessary and sufficient to make the LP integral. In other words, the basic feasible solutions to the LP are exactly the perfect matchings of G. The dual LP is now given by

$$\begin{aligned} &\max \sum_{A \in \mathcal{A}} z_A && \\ \text{s.t.} \quad &\sum_{A \in \mathcal{A} \,:\, e \in \delta(A)} z_A \leq w(e) && \forall e \in E \\ &z_A \geq 0 && \forall A \in \mathcal{A} \text{ with } |A| > 1 \end{aligned}$$

where we introduced a variable z_A for all $A \in \mathcal{A}$. By strong duality, the optimal value of the primal and the dual coincide.

▶ **Definition 11** (Residual Graph). *Let $G = (V, E)$ be a graph with edge-weights $w : E \to \mathbb{Z}$. Let z be an optimal solution to the dual LP associated with G. The graph $G_z := (V, E_z)$ with*

$$e \in E_z \iff w(e) = \sum_{A \in \mathcal{A} \,:\, e \in \delta(A)} z_A$$

is called residual graph with respect to z.

Unsurprisingly, it turns out that all minimum-weight perfect matchings are contained in the residual graph with respect to a given optimal solution z. However, there might exist perfect matchings in the residual graph that are not minimum-weight perfect matchings of G. Hence, we need an extra condition to exactly characterize which perfect matchings in the residual graph are minimum-weight in G. Eppstein [10] shows such a result for bipartite graphs. We present here the natural generalization to all graphs. This essentially follows from the complementary slackness theorem [20].

▶ **Lemma 12.** *Let $G = (V, E)$ be a graph with edge-weights $w : E \to \mathbb{Z}$. Let z be an optimal solution to the dual LP associated with G, and let $G_z = (V, E_z)$ be its residual graph. A perfect matching M of G has minimum weight if and only if $M \subseteq E_z$ and $|M \cap \delta(A)| = 1$ holds for all $A \in \mathcal{A}$ with $z_A > 0$.*

Proof. For the sake of clarity, we prove both directions individually.
(⇒) We provide an indirect proof. Assume first that there exists $f \in M$ such that

$$\sum_{A \in \mathcal{A} \colon f \in \delta(A)} z_A < w(f)$$

holds and hence $f \notin E_z$. Observe that, by feasibility of z, we have

$$\sum_{A \in \mathcal{A} \colon e \in \delta(A)} z_A \leq w(e)$$

for all $e \in E$. Moreover, we have $z_A \geq 0$ for all $A \in \mathcal{A}$ with $|A| > 1$. Thus, we get

$$w(M) - \sum_{A \in \mathcal{A}} z_A \geq \sum_{e \in M} \left(w(e) - \sum_{A \in \mathcal{A} \colon e \in \delta(A)} z_A \right) > 0$$

since every set $A \in \mathcal{A}$ with $|A| = 1$ is associated with exactly one edge in M, and every other set $A \in \mathcal{A}$ (with $|A| > 1$) satisfies $z_A \geq 0$. In other words, we have

$$w(M) > \sum_{A \in \mathcal{A}} z_A$$

which, by strong duality, implies that M does not have minimum weight. Thus, assume now that $M \subseteq E_z$, but there exists $A \in \mathcal{A}$ such that $z_A > 0$ and $|M \cap A| > 1$. Again, we calculate

$$w(M) - \sum_{A \in \mathcal{A}} z_A > \sum_{e \in M} \left(w(e) - \sum_{A \in \mathcal{A} \colon e \in \delta(A)} z_A \right) = 0$$

where the strict inequality now comes from the fact that we needed to duplicate the variable z_A in the first step. Again, by strong duality, we conclude that M does not have minimum weight.
(⇐) Assume now that we have $M \subseteq E_z$ and $|M \cap A| = 1$ for all $A \in \mathcal{A}$ with $z_A > 0$. In this case, we get

$$w(M) - \sum_{A \in \mathcal{A}} z_A = \sum_{e \in M} \left(w(e) - \sum_{A \in \mathcal{A} \colon e \in \delta(A)} z_A \right) = 0$$

as every set $A \in \mathcal{A}$ with $z_A > 0$ is associated with exactly one edge in M. By strong duality, we can use the fact that z is optimal to conclude that M has minimum weight. ◀

Using Lemma 12, we can prove the following lemma, which captures the special case $\ell = 2$ of Theorem 9.

▶ **Lemma 13.** *Let $G = (V, E)$ be a graph with edge-weights $w : E \to \mathbb{Z}$. Assume that G contains a 2nd-smallest perfect matching M. Then there exist edges $e, f \in E$ such that*

$$w(M) = \min_{M' \in \mathcal{M}_G \colon \{e,f\} \subseteq M'} w(M').$$

Proof. Let z be a fixed optimal dual solution. Since M is not a minimum-weight perfect matching, by Lemma 12, there are two cases: Either there exists an edge $e \in M$ with $e \notin E_z$ (in this case, take as f any edge from M and we are done), or there exists some odd set $A \in \mathcal{A}$ with $z_A > 0$, distinct edges $e, f \in M$, and both $e, f \in \delta(A)$. In the latter case, by the same lemma, no minimum-weight perfect matching contains both e and f. Hence, we get

$$\text{OPT} < \min_{M' \in \mathcal{M}_G : \{e,f\} \subseteq M'} w(M') = w(M)$$

where OPT denotes the weight of a minimum-weight perfect matching in G. ◀

It remains to use Lemma 13 repeatedly to obtain Theorem 9.

Proof of Theorem 9. We proceed by induction over ℓ. The case $\ell = 1$ is trivial and Lemma 13 already proves the case $\ell = 2$. Thus, let now $\ell \geq 3$ be arbitrary, and assume as induction hypothesis that the statement holds for all $\ell' < \ell$. Recall that M is an ℓ-th smallest perfect matching. We want to prove that there is a subset $F \subseteq E$ of edges of size at most $2(\ell - 1)$ such that

$$w(M) = \min_{M' \in \mathcal{M}_G : F \subseteq M'} w(M').$$

Let $M_{\min}$ be an arbitrary minimum-weight perfect matching of G. From the argument in Lemma 13, we know that there exist edges $e, f \in M$ such that

$$w(M_{\min}) < \min_{M' \in \mathcal{M}_G : \{e,f\} \subseteq M'} w(M') \leq w(M).$$

Consider now the graph G' obtained from G by removing e and f, their incident vertices, and all their incident edges. Observe that $M \setminus \{e, f\}$ is an ℓ'-th smallest perfect matching in G' for some $\ell' < \ell$. By the induction hypothesis, this means that there is a set $F' \subseteq E \setminus \{e, f\}$ of size at most $2(\ell' - 1)$ such that

$$w(M \setminus \{e, f\}) = \min_{M' \in \mathcal{M}_{G'} : F' \subseteq M'} w(M').$$

Define $F := F' \cup \{e, f\}$ and observe that

$$w(M) = w(\{e, f\}) + \min_{M' \in \mathcal{M}_{G'} : F' \subseteq M'} w(M') = \min_{M' \in \mathcal{M}_G : F \subseteq M'} w(M')$$

and that the set F has size at most $2(\ell - 1)$. ◀

4 Related Cycle Problems

In this section, we present the ideas behind Theorem 14 below. The full proof of Theorem 14 and the involved lemmata can be found in the Appendix of the full version.

▶ **Theorem 14** (EWPM $\equiv_p$ ECS & BCPM $\equiv_p$ SOC)**.** *EWPM is polynomial-time equivalent to ECS. BCPM is polynomial-time equivalent to SOC.*

As we will see, both equivalences use the same base construction and only differ in some minor configurations. The encoding of the weights is preserved in all reductions and hence ECS is equivalent to EM for weights encoded in unary, and NP-complete for weights encoded in binary.

From Matchings to Cycles

We start with the direction of EWPM $\leq_p$ ECS and BCPM $\leq_p$ SOC, i.e. we give reductions starting from matching-type problems going to cycle-type problems. We use roughly the same construction for both reductions. We remark that in the case of matching-type problems in *bipartite graphs*, a similar reduction already exists, and it goes to cycle-type problems in *directed* graphs [27, 8]. In these existing reductions, the bipartition of the graph was used to define some edge orientation, which in turn ensured that cycles in the new graph would correspond to M-alternating cycles (where M is an arbitrary minimum-weight perfect matching) in the old graph (see [27] and [8] for more details). The same trick does not apply in general graphs, which is why we need to reduce to undirected cycle problems instead. In particular, the new trick will be to manipulate the weights such that the weight of any non-alternating cycle exceeds some threshold. The following lemma captures the key properties of this construction.

▶ **Lemma 15.** *Let $G = (V, E)$ be a graph with non-negative edge-weights $w : E \to \mathbb{N}$ that admits a minimum-weight perfect matching M. Consider the new edge-weights $w' : E \to \mathbb{Z}$ defined as*

$$w'(e) := \begin{cases} -w(e) - r - 1, & e \in M \\ w(e) + r + 1, & e \notin M \end{cases}$$

for some integer $r \geq w(M)$. Then every cycle C with $w'(C) \leq r - w(M)$ is M-alternating and w' is conservative.

Using Lemma 15, we can now informally describe the reductions: Assume first that we are given an EWPM-instance I consisting of $G = (V, E)$, $w : E \to \mathbb{Z}$, and $k \in \mathbb{Z}$. Without loss of generality, we can assume that the weights w are non-negative. We first compute a minimum-weight perfect matching M in G. Note that if no such matching exists or its weight exceeds k, I is a NO-instance and we can produce a simple NO-instance of ECS. Otherwise, we construct w' as in Lemma 15 (setting $r = k$) and return the ECS-instance I' consisting of G, w' and $k' := k - w(M)$. By Lemma 15, w' is conservative. Setting $r = k$ crucially guarantees that solutions to the ECS-instance can be translated back to solutions to the EWPM-instance I.

The only difference in the reduction from BCPM to SOC is that we need to manually check the parity of M and use a trivial construction if it already has correct parity.

From Cycles to Matchings

We will now turn our attention to the reverse reductions ECS $\leq_p$ EWPM and SOC $\leq_p$ BCPM. The general idea is again inspired by the existing reductions between bipartite matching and directed cycle problems ([27, 8]): We want to duplicate every vertex and add edges between duplicates in order to ensure the existence of a canonical perfect matching M. Additionally, we want to ensure that the existence of any other perfect matching M' in the new graph implies existence of cycles in the original graph by looking at the symmetric difference $M \Delta M'$. Unfortunately, since our original graph is not directed anymore, we cannot use the ideas from [27, 8] to achieve this. Instead, we use the construction described in Lemma 16 and Figure 2.

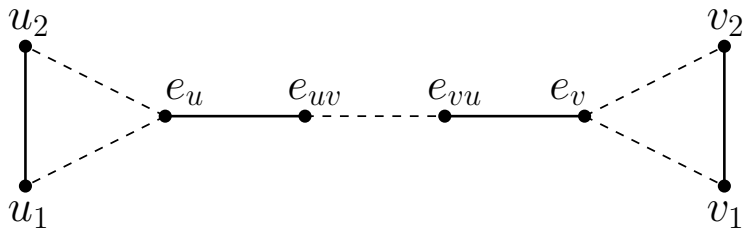


Figure 2 This figure shows the construction from Lemma 16 for a single edge $e = \{u, v\}$. Solid edges indicate the perfect matching M from Lemma 16. The edge $\{e_{uv}, e_{vu}\}$ receives the weight of e while all other edges get a weight of zero.

▶ **Lemma 16.** *Let $G = (V, E)$ be a graph with conservative edge-weights $w : E \to \mathbb{Z}$. We define*

$$\begin{aligned}
V' \coloneqq& \{v_1, v_2 \mid v \in V\} \cup \{e_u, e_{uv}, e_{vu}, e_v \mid e = \{u, v\} \in E\} \\
E' \coloneqq& \{\{v_1, v_2\} \mid v \in V\} \\
&\cup \{\{u_1, e_u\}, \{u_2, e_u\}, \{e_u, e_{uv}\}, \{e_{uv}, e_{vu}\}, \\
&\quad \{e_{vu}, e_v\}, \{e_v, v_1\}, \{e_v, v_2\} \mid e = \{u, v\} \in E\} \\
w' \coloneqq& e' \in E' \mapsto \begin{cases} w(e) & \text{if } e' = \{e_{uv}, e_{vu}\} \text{ for some } e = \{u, v\} \in E \\ 0 & \text{otherwise} \end{cases} \\
M \coloneqq& \{\{v_1, v_2\} \mid v \in V\} \cup \{\{e_u, e_{uv}\}, \{e_{vu}, e_v\} \mid e = \{u, v\} \in E\}
\end{aligned}$$

and consider the graph $G' = (V', E')$ with edge-weights w' and perfect matching M. Then there exist vertex-disjoint cycles $C_1, \ldots, C_p$ of total weight k (w.r.t. w) in G if and only if there exist vertex-disjoint M-alternating cycles $C'_1, \ldots, C'_p$ of total weight k (w.r.t. w') in G'.

Let us now briefly explain how this yields the desired reductions. Assume that we are given an ECS-instance I consisting of $G = (V, E)$, $w : E \to \mathbb{Z}$, and $k \in \mathbb{Z}$. We use the construction from Lemma 16 to obtain a graph G' with edge-weights w' and return the EWPM-instance I' consisting of $G' = (V', E')$, w', and $k' \coloneqq k$. The symmetric difference $M \Delta M'$ of M with any perfect matching M' in G' yields a set of vertex disjoint cycles that can be translated back to G using Lemma 16. The reduction from SOC to BCPM uses the same construction, but we need to additionally observe that we can get a single odd-weight cycle that translates through Lemma 16.

5 Final Remarks

In Section 3, we proved that we can deterministically find an ℓ-th smallest perfect matching in time $n^{\mathcal{O}(\ell)}$. This also allows us to solve certain instances of EWPM efficiently: Assume e.g. that we are given an EWPM instance with k such that $k_1 < k_2 < \cdots < k_\ell = k$ are all the distinct weights up to k that perfect matchings in G can take. Then our algorithm can find a perfect matching of weight $k = k_\ell$ efficiently if ℓ is small.

Clearly, this also works in the case of BCPM. Moreover, our reductions from Section 4 also allow us to get similar guarantees for ECS and SOC, respectively. Indeed, assume that we are given an instance of ECS or SOC with k such that $k_1 < k_2 < \cdots < k_\ell = k$ are all the distinct weights up to k that cycle sums in G can take. We apply our reduction to build a corresponding EWPM or BCPM instance. By Lemma 16, our reductions have a one-to-one correspondence between ℓ-th smallest perfect matchings and ℓ-th smallest cycle sums. Hence, the parameterized runtime guarantees of the SPM$_\ell$-algorithm carry through the reduction.

▶ **Corollary 17.** *BCPM can be solved in time $n^{\mathcal{O}(\ell)}$ if $k_1 < k_2 < \cdots < k_\ell = k$ are all the distinct weights up to k that perfect matchings in G can take. ECS and SOC can be solved in time $n^{\mathcal{O}(\ell)}$ if $k_1 < k_2 < \cdots < k_\ell = k$ are all the distinct weights up to k that cycle sums in G can take.*

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge ; New York, 2009.

2 Chandra R. Chegireddy and Horst W. Hamacher. Algorithms for finding k-best perfect matchings. *Discrete applied mathematics*, 18(2):155–165, 1987. doi:10.1016/0166-218X(87)90017-5.

3 Anita Dürr, Nicolas El Maalouly, and Lasse Wulf. An Approximation Algorithm for the Exact Matching Problem in Bipartite Graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, volume 275 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:21, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.18.

4 Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69:125–130, 1965.

5 Nicolas El Maalouly. Exact Matching: Algorithms and Related Problems. In *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.29.

6 Nicolas El Maalouly, Sebastian Haslebacher, and Lasse Wulf. On the Exact Matching Problem in Dense Graphs. In *41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.STACS.2024.33.

7 Nicolas El Maalouly and Raphael Steiner. Exact Matching in Graphs of Bounded Independence Number. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.MFCS.2022.46.

8 Nicolas El Maalouly, Raphael Steiner, and Lasse Wulf. Exact Matching: Correct Parity and FPT Parameterized by Independence Number. In *34th International Symposium on Algorithms and Computation (ISAAC 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ISAAC.2023.28.

9 Nicolas El Maalouly and Lasse Wulf. Exact Matching and the Top-k Perfect Matching Problem, September 2022. doi:10.48550/arXiv.2209.09661.

10 David Eppstein. Representing all minimum spanning trees with applications to counting and generation. Technical report, UC Irvine: Donald Bren School of Information and Computer Sciences, 1995. URL: https://escholarship.org/uc/item/5w35f33b.

11 David Eppstein. K-best enumeration. *Bulletin of EATCS*, 1(115), 2015.

12 Anna Galluccio and Martin Loebl. On the Theory of Pfaffian Orientations. I. Perfect Matchings and Permanents. *The Electronic Journal of Combinatorics*, pages R6–R6, 1999. doi:10.37236/1438.

13 Jim Geelen and Rohan Kapadia. Computing Girth and Cogirth in Perturbed Graphic Matroids. *Combinatorica*, 38(1):167–191, February 2018. doi:10.1007/s00493-016-3445-3.

14 Rohit Gurjar, Arpita Korwar, Jochen Messner, Simon Straub, and Thomas Thierauf. Planarizing Gadgets for Perfect Matching Do Not Exist. In *Mathematical Foundations of Computer Science 2012*, pages 478–490, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-32589-2_43.

15 Meike Hatzel, Konrad Majewski, Michał Pilipczuk, and Marek Sokołowski. Simpler and faster algorithms for detours in planar digraphs. In *2023 Symposium on Simplicity in Algorithms (SOSA)*, Proceedings, pages 156–165. Society for Industrial and Applied Mathematics, January 2023. doi:10.1137/1.9781611977585.ch15.

16 Xinrui Jia, Ola Svensson, and Weiqiang Yuan. The Exact Bipartite Matching Polytope Has Exponential Extension Complexity. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 1635–1654. Society for Industrial and Applied Mathematics, January 2023. doi:10.1137/1.9781611977554.ch61.

17 Donald B. Johnson. Efficient Algorithms for Shortest Paths in Sparse Networks. *J. ACM*, 24(1):1–13, January 1977. doi:10.1145/321992.321993.

18 Kuo-Hua Kao, Jou-Ming Chang, Yue-Li Wang, and Justie Su-Tzu Juan. A Quadratic Algorithm for Finding Next-to-Shortest Paths in Graphs. *Algorithmica*, 61(2):402–418, October 2011. doi:10.1007/s00453-010-9402-4.

19 A. V. Karzanov. Maximum matching of given weight in complete and complete bipartite graphs. *Cybern Syst Anal*, 23(1):8–13, January 1987. doi:10.1007/BF01068796.

20 Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21 of *Algorithms and Combinatorics*. Springer, Berlin, Heidelberg, 2018. doi:10.1007/978-3-662-56039-6.

21 I. Krasikov and S. D. Noble. Finding next-to-shortest paths in a graph. *Information Processing Letters*, 92(3):117–119, November 2004. doi:10.1016/j.ipl.2004.06.020.

22 Kumar N. Lalgudi and Marios C. Papaefthymiou. Computing strictly-second shortest paths. *Information Processing Letters*, 63(4):177–181, August 1997. doi:10.1016/S0020-0190(97)00122-1.

23 Shisheng Li, Guangzhong Sun, and Guoliang Chen. Improved algorithm for finding next-to-shortest paths. *Information Processing Letters*, 99(5):192–194, September 2006. doi:10.1016/j.ipl.2006.04.013.

24 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, March 1987. doi:10.1007/BF02579206.

25 Hitoshi Murakami and Yutaro Yamaguchi. An FPT Algorithm for the Exact Matching Problem and NP-Hardness of Related Problems. *IEICE Transactions on Information and Systems*, E108.D(3):214–220, March 2025. doi:10.1587/transinf.2024FCP0009.

26 Katta G Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations research*, 16(3):682–687, 1968.

27 Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *J. ACM*, 29(2):285–309, April 1982. doi:10.1145/322307.322309.

28 Ildikó Schlotter and András Sebő. Odd Paths, Cycles, and T-Joins: Connections and Algorithms. *SIAM Journal on Discrete Mathematics*, 39(1):484–504, March 2025. doi:10.1137/23M158156X.

29 J. W. Suurballe. Disjoint paths in a network. *Networks*, 4(2):125–145, January 1974. doi:10.1002/net.3230040204.

30 Vijay V. Vazirani. NC algorithms for computing the number of perfect matchings in K3,3-free graphs and related problems. In *SWAT 88*, pages 233–242, Berlin, Heidelberg, 1988. Springer. doi:10.1007/3-540-19487-8_27.

31 Bang Ye Wu. A Simpler and More Efficient Algorithm for the Next-to-Shortest Path Problem. *Algorithmica*, 65(2):467–479, February 2013. doi:10.1007/s00453-011-9601-7.

32 Raphael Yuster. Almost Exact Matchings. *Algorithmica*, 63(1-2):39–50, June 2012. doi:10.1007/S00453-011-9519-0.

33 Cong Zhang and Hiroshi Nagamochi. The Next-to-Shortest Path in Undirected Graphs with Nonnegative Weights. *Theory of Computing*, 128, 2012. URL: http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV128Zhang.html.

Beeping Deterministic CONGEST Algorithms in Graphs

Pawel Garncarek
Institute of Computer Science, University of Wroclaw, Poland

Dariusz R. Kowalski
School of Computer & Cyber Sciences, Augusta University, GA, USA

Shay Kutten
Department of Data and Decision Sciences, Technion, Haifa, Israel
The Grand Technion Energy Program, Haifa, Israel

Miguel A. Mosteiro
Pace University, Computer Science Department, New York, NY, USA

Abstract

Beeping Network (BN) is a popular graph-based model of wireless computation, which applies the OR operation to one-bit messages sent simultaneously by neighbors. It admits fast (polylogarithmic in the number of nodes n) randomized solutions to many graph problems, but all known deterministic algorithms for non-trivial graph problems are at least polynomial in the maximum node degree Δ.

We improve known results for deterministic algorithms by showing that this polynomial can be as low as $\widetilde{O}(\Delta^2)$. More precisely, we show how to simulate a single round of any CONGEST algorithm in any network in $O(\Delta^2 \ polylog\ n)$ beeping rounds, each accommodating at most one beep per node, even if the nodes intend to send different messages to different neighbors. This upper bound reduces polynomially the time for a *deterministic* simulation of CONGEST in a Beeping Network, comparing to the best known algorithms, and nearly matches the time obtained recently using *randomization* (up to a poly-logarithmic factor) as well as the lower bound. Specifically, any algorithm designed for the CONGEST networks can be run in BNs with $O(\Delta^2 \ polylog\ n)$ multiplicative overhead, e.g., we can now deterministically compute an MIS in any BN in $O(\Delta^2 \ polylog\ n)$ beeping rounds, improving the previous best $\Theta(\Delta^3)$-round solution. For h-hop simulations, we prove a lower bound $\Omega(\Delta^{h+1})$, and we design a nearly matching algorithm that is able to "pipeline" the node-to-node information in a faster way than beeping layer-by-layer.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Beeping Networks, CONGEST Networks, deterministic simulations, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.20

Related Version *Full Version*: https://arxiv.org/abs/2502.13424 [19]

Funding *Pawel Garncarek*: Supported by the National Science Center, Poland (NCN), grant 2020/39/B/ST6/03288.
Shay Kutten: The work of this author was funded in part by grant 1346/22 from the Israeli Science Foundation. His research was carried out within the framework of the Grand Technion Energy Program (GTEP). A part of this work was done while with Fraunhofer SIT Darmstadt.
Miguel A. Mosteiro: Partially supported by Pace SRC grant and Kenan Fund.

1 Introduction

The study of the Beeping Networks (BN) model [10] is simultaneously interesting, challenging, and useful in several respects. Even some less-restrictive ad-hoc networks (e.g., wireless) are frequently the algorithmist delight: a seemingly simple computational problem that



© Pawel Garncarek, Dariusz R. Kowalski, Shay Kutten, and Miguel A. Mosteiro;
licensed under Creative Commons License CC-BY 4.0



becomes challenging under harsh yet realistic conditions. In terms of communication, BN may be the harshest model: network nodes can only *beep* (emit a signal) or *listen* (detect if a signal is emitted in its vicinity). A listening node may distinguish between *silence* (no beeps) and *noise* (at least one beep), but it cannot distinguish between single and multiple beeps. Theoretically, the BN model is important since it enables one to study whether distributed tasks can be performed efficiently under minimal conditions. It is a fundamental model to study communication complexity on channels where signals are superimposed (by applying the OR operator), which makes it more challenging than the basic model in which transmissions on links are independent. Moreover, in a one-hop network,[1] non-adaptive communication schedules in the BN model are equivalent to superimposed codes, which have been widely applied not only to communication problems, but also to text alignments, bio-computing, data pooling, dimensionality reduction, and other areas.

In addition to its theoretical importance, this model is recognized as useful for studying natural communication in biological networks [30, 2] (e.g., cells). A better understanding of the power of biological communication processes may lead to better nature-inspired algorithms [28]. It has been argued that the model is also a practical tool because algorithms designed for BNs can be implemented on non-expensive devices with low energy consumption. For example, nodes deployed (in ad-hoc topologies) for monitoring in Internet of Things (IoT) applications such as Sensor Networks. It should be noted that a mechanism similar in properties to beeping, called "busy tone," has been in wide use in wireless networks but for very limited purposes in channel access algorithms (as opposed to the Beeping model that is intended for general-purpose distributed algorithms). See e.g., [32, 22].

A wealth of successful research on computational problems in Beeping Networks has appeared in the literature (see, for instance [5, 15] and the references therein). Nevertheless, fundamental distributed computing questions remain open in the context of Beeping Networks, especially for deterministic algorithms. On the other hand, the CONGEST [31] model has been profusely studied. A natural question that follows is how to efficiently transform CONGEST Network algorithms into Beepping Network algorithms.

1.1 Our Contributions

The main results are detailed and compared to previous work in Table 1. Our main contributions are the near optimal[2] deterministic implementations of two simulators.

The main simulator, see Section 3 and Theorem 8, efficiently translates any CONGEST algorithms (deterministic or randomized) to the Beeping model. Each CONGEST round is simulated using $O(\Delta^2 \; polylog \; n)$ rounds, improving the previous result of $O(\Delta^4 \log n)$ of [5] and recent $O(\Delta^3 \log n)$ of [12] [3] (for $\Delta \in \omega(\; polylog \; n)$), and matches the lower bound $\Omega(\Delta^2 \log n)$ in [12] (that lower bound holds even for randomized simulations, hence combined with our algorithm they imply that the randomization can not help more than, possibly, a polylogarithmic factor).

Our approach differs substantially from previous ones based on superimposed codes. Our algorithm is adaptive, but using *a family of codes*, called avoiding selectors, that are parts of nodes' beeping schedules. These codes combined assure that, gradually, more and more

[1] For any given path of links connecting two nodes, the number of *hops* is the number of links in such path.
[2] Optimal up to a polylogarithmic factor.
[3] The latter result was not explicitly stated in [12], but one could use its randomized $O(\Delta^2 \log n)$-round solution designed for broadcasting $O(\Delta \log n)$ input bits stored in each node (this is the number of bits in our case, as messages to individual Δ neighbors could be different), and then derandomize it by using the suggested superimposed codes (with additional factor $O(\Delta)$, hence $O(\Delta^3 \log n)$ in total).

Table 1 Summary of our main results and related work for Beeping Networks with n nodes and maximum degree Δ. All protocols are distributed. Highlighted pairs of cells show direct comparison of our main results with previous work. B denotes the maximum number of input bits at a node.

problem	protocol type	beeping rounds	ref
simulation of one CONGEST round	randomized	$O(\Delta \min(n, \Delta^2) \log n)$ whp	[3]
		$O(\Delta^2 \log n)$ whp	[12]
	deterministic	$O(\Delta^4 \log n)$	[5]
		$O(\Delta^3 \log n)$	[12]
		$O(\Delta^2 \ \textit{polylog}\ n)$	Thm. 8
	rand./det.	$\Omega(\Delta^2 \log n)$	[12]
B-bit h-hop simulation	deterministic	$O(h \cdot B\Delta^{h+2} \ \textit{polylog}\ n)$	Thm. 11
	rand./det.	$\Omega(B\Delta^{h+1})$	Thm. 10
Learning Neighborhood	deterministic	$O(\Delta^2 \log^2 n)$	in full version of this work [19]
Cluster Gathering		$O(\Delta^2 \log^4 n)$	in full version of this work [19]
$(\log n, \log^2 n)$-Network Decomposition		$O(\Delta^2 \log^8 n)$	in full version of this work [19]
MIS		$O(\Delta^3 + \Delta^2 \log n)$	[5]
		$O(\Delta^2 \ \textit{polylog}\ n)$	Cor. 9

links will "successfully beep" the information between their two end nodes, by "avoiding" interfering beeps from other nodes. The main challenge is that each node may have some incident links already successful and other links that are not – hence, it has to beep anyways and may interfere with other neighbors but, thanks to our avoiding selectors, not all the time. In this process, once a node recognizes that a neighbor is a unique beeper (so called "announcer"), it decodes its ID and message, and then it competes with other nodes to respond in subsequent rounds, and after that – it gets confirmation from the announcer.

Simulations of some CONGEST algorithms, relying on sending *the same message to all neighbors in a round* such as Network Decomposition in [20], could be further improved by a polylogarithmic factor by applying a deterministic version of local broadcast algorithm (i.e., same message to all neighbors) proposed in [12] (see the full version of this work [19]).

Our second main result is an extension of the notion of neighborhood to $h \geq 1$ hops and pipelining of the one-hop simulation. We call this problem a *B-bit h-hop simulation*, and show bounds $O(h \cdot B\Delta^{h+2} \ \textit{polylog}\ n)$ and $\Omega(B\Delta^{h+1})$, where the lower bound holds also for randomized solutions. See Section 4, and Theorems 11 and 10, respectively. Our algorithm efficiently "pipelines" point-to-point messages, and achieves substantially better complexity (for $h > 3$) than a straightforward application of 1-hop simulator h times (which would need $O(\Delta^{2h} \ \textit{polylog}\ n)$ time). For the simpler problem when each node has to deliver the same message to all nodes in its h-hop neighborhood, which we call *B-bit h-hop Local Broadcast*, we show a lower bound of $\Omega(B\Delta^h)$ even with randomization (see the full version of this work [19]).

1.2 Related Work

The Beeping Network model was defined by Cornejo and Kuhn in [10] in 2010, inspired by continuous beeping studied by Degesys et al. [13] and Motskin et al. [29], and by the implementation of coordination by carrier sensing given by Flury and Wattenhofer in [17]. Since then, the literature has included studies on MIS and Coloring [2, 1, 27, 23, 5, 7], Naming [8], Leader Election [21, 18, 14], Broadcast [21, 25, 26, 11, 4], and Shortest Paths [15].

Techniques to implement CONGEST algorithms in BNs were studied. In [5], the approach is to schedule transmissions according to a 2-hop c-coloring to avoid collisions. The multiplicative overhead introduced by the simulation is $O(c^2 \log n)$. A constant c is enough for the simulation, but the only coloring algorithm provided in the same paper takes time $O(a^2\Delta^2 \log^2 n + a^3\Delta^3 \log n)$ for a (Δ^2+1)-coloring. Thus, the multiplicative overhead is $O(\Delta^4 \log n)$, which our simulation improves by a factor of $\Delta^2/$ *polylog* n.

On the side of randomized protocols, in a recent work by Davies [12], a protocol that simulates a CONGEST round in a Beeping Network with $O(\Delta^2 \log n)$ overhead is presented. The protocol works even in the presence of random noise in the communication channel. Still, it is correct only with high probability (whp),[4] and requires a polynomial number of CONGEST rounds in the simulated algorithm. More relevant for comparison with our work, in the same paper, a lower bound of $\Omega(\Delta^2 \log n)$ on the overhead to simulate a CONGEST round is shown. The lower bound applies even in a noiseless environment and regardless of randomization. Thus, our simulation is optimal modulo some poly-logarithmic factor. [12] also proposed using superimposed codes instead of randomness, resulting in $\Theta(\Delta)$ multiplicative overhead, i.e., $O(\Delta^3 \log n)$ rounds in total. Another randomized simulation was previously presented in [3] with overhead of $O(\Delta \min(n, \Delta^2) \log n)$ whp.

Regarding the MIS problem, the closest work is the deterministic protocol in [5], which runs in $O(\Delta^2 \log n + \Delta^3)$. Thus, our results improve by a factor of $\Delta/$ *polylog* n for any $\Delta \in \omega(\log n)$, and match the running time (modulo poly-logarithmic factor) for $\Delta \in O(\log n)$.

On the side of randomized MIS protocols, an upper bound of $O(\log^3 n)$ has been shown in [1] for the same beeping model, and faster with some additional assumptions.

Our network decomposition algorithm for BNs is based on the protocol for the CONGEST model [20] that shows a $(\log n, \log^2 n)$ network decomposition in $O(\log^5 n)$ CONGEST rounds. We get the same network decomposition in $O(\Delta^2 \log^7 n)$ beeping rounds. Other graph problems and (restricted) local broadcast of *same message* are discussed in the full version of this work [19].

Non-synchronized BNs pose even more challenges to graph algorithms, cf. [1, 18, 14, 24].

2 Model, Notation, and Problems

We specify first the communication network notation used throughout, and we define the communication models studied in the section that follows. We consider a communication network formed by n devices with communication and computation capabilities, called **nodes**. Each node has a unique **ID** from the range $[1, n^c]$ for some constant $c \geq 1$.[5] Nodes communicate by sending **messages** among them. A message is composed of a binary sequence containing the source node ID, the destination node ID (if applicable), and the specific information to be sent. If the destination node receives the message from the source node,

[4] An event E occurs *with high probability* if $Prob(E) \geq 1 - 1/n^c$ for some $c > 0$.

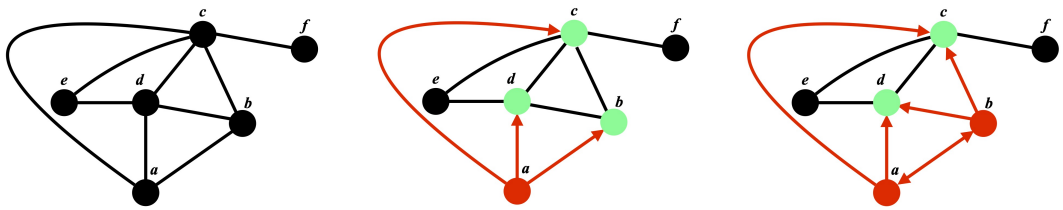
[5] The availability of identifiers is essential in order to break symmetry in deterministic protocols, as pointed out in previous works on deterministic protocols in the Beeping model [5, 14].

we say that the message was **delivered**. Each pair of nodes that are able to communicate directly (i.e., without relaying communication through other nodes) are said to be connected by a communication **link** and are called **neighbors**. We assume that links are **symmetric**, i.e., messages can be sent in both directions (delivery is restricted to the communication models specified below). The network topology defined by the communication links is modeled with an undirected graph $G = (V, E)$ where V is the set of nodes and E is the set of links. If E is such that for every pair of nodes $u, v \in V$ there is a path of links connecting u and v we say that the network is **connected**. For each node $v \in V$, the set of neighbors of v is called its **neighborhood**, denoted as $N(v)$. We assume that time is slotted in **rounds** of communication. All nodes start running protocols simultaneously, i.e., the network is **synchronous**. We assume that computations take negligible time with respect to communication. Thus, we measure algorithm performance in rounds.

2.1 Communication Models

Beeping Networks (BNs) [10]. In this model, in each round each node can either **beep** (send a signal) or **listen** (do not send any signal). By doing so, nodes obtain the following **communication channel feedback**. In any given round, a listening node **hears** either **silence** (no neighbor beeps) or **noise** (one or more neighbors beep). A listening node that hears noise cannot distinguish between a single beep and multiple beeps.

Network protocols may use the channel feedback (i.e. the temporal sequence of strings from {“silence”,“noise”}) to make decisions adaptively. However, delivering messages is not straightforward because it requires sending (and receiving) the whole binary sequence of the message (according to some beeping schedule, possibly changing adaptively during the communication), somehow encoded with beeps. In that sense, protocols for Beeping Networks can be seen as radio network coding to cope with the communication restrictions (as in [16] to cope with noise).



(a) All nodes listen. All nodes hear silence.

(b) Node a beeps, $\{b, c, d, e, f\}$ listen. $\{b, c, d\}$ hear noise. $\{e, f\}$ hear silence.

(c) Nodes $\{a, b\}$ beep, $\{c, d, e, f\}$ listen. $\{c, d\}$ hear noise. $\{e, f\}$ hear silence.

Figure 1 Beeping Network communication model example.

CONGEST Networks [31]. In this model, in each round each node can send a (possibly different) message of $O(\log n)$ bits to each neighbor independently. All nodes receive the messages sent by their neighbors. That is, there are no collisions. In the **CONGEST Broadcast** version of this model, each node can only broadcast the same message to all its neighbors in each round.

2.2 Problems Studied

Our main research question in this work is how to efficiently simulate a round of communication of a CONGEST Network protocol in a Beeping Network, and how to improve MIS in BNs.

CONGEST Simulation. Given a Beeping Network with topology graph $G = (V, E)$, where each node $u \in V$ may hold a message $m_{u,v}$ of $O(\log n)$ bits that must be delivered to node $v \in N(u)$, the CONGEST simulation problem is solved once for every $u \in V$ and $v \in N(u)$, $m_{u,v}$ and the ID of u has been delivered to v.

Maximal Independent Set (MIS). Given a Beeping Network with topology graph $G = (V, E)$, The Maximal Independent Set problem is solved when, for some such set, $S \subseteq V$, every node $v \in S$, v knows that it is in S.

Our second main problem – generalized **h-hop simulations** – are defined and studied in Section 4 and in the full version of this work [19]. We also study several distributed computing problems of independent interest (see their definition in the full version of this work [19]) in the context of Beeping Networks and applications of our simulator that improve efficiency with respect to known solutions for BNs.

3 Simulation of a CONGEST Round in Beeping Networks

We present a deterministic distributed algorithm that efficiently simulates a round of *any algorithm designed for the CONGEST model* in the Beeping Networks model, even if the algorithm sends different messages to neighbors. It is only somewhat (polylogarithmically) slower than the existing more restricted local broadcast algorithms, which require a node to send *the same message* to all its neighbors), e.g., the local broadcast solution mentioned in [12] (see also more details in the full version of this work [19]) and faster nearly by factor Δ than the full deterministic simulation also mentioned in [12]. Unlike the superimposed-codes-based deterministic algorithms in [12], our simulator is adaptive and uses heavier machinery. It is built hierarchically using the known family of codes called "avoiding selectors". The codes, intuitively, say "when to beep". However, it is still possible that when two neighbors of some node v send their messages (by beeping in subsequent rounds), node v will receive a "message" that is a logical OR of the two. Our simulator resolves such collision problems and assures fast progress by:

- coding messages (cf. extended-IDs) such that OR of more than one message is identified;
- beeping according to specific avoiding selectors, assuring that a fraction of links that have not yet "delivered" the beeped messages will now succeed in one direction (i.e., one end node of such link "announces" itself to the other one through successful beeping of its ID);
- allowing those who received such announcements, the responders, to compete in successful delivery of their IDs to the corresponding announcers; this competition sets up bi-directional matching between the announcers and the corresponding responders, which allows bi-directional message exchange via beeping (3-stage parallel handshaking); this competition also uses avoiding selectors, but differently parameterized than the ones used for announcing (as announcing and responding are different processes).

The announcing, followed by responders' competition, continues until all links end up in some of the successfully realized matchings (as described in the above bullet points).

Avoiding selectors for n nodes use two parameters, k, ℓ, corresponding to the number of competing neighbors/responders versus the other (potentially interrupting) neighbors:

Algorithm 1 C2B algorithm for each node u. $E(u)$ is a global variable.

$E(u) \leftarrow \{\{u,v\} | \{u,v\} \in E \text{ for some } v \in V\}$

Algorithm `C2B()`

```
1  for each epoch i = 1, 2, ..., log Δ do                    // iterating epochs
2      k_i ← Δ/2^i
3      for each phase j = 1, 2, ..., |F_{Δ,k_i}| do
4          if u ∈ F_{Δ,k_i}(j) then
5              announcer(u, k_i)
6          else
7              responder(u, k_i)
```

▶ **Definition 1** (Avoiding selectors). *A family $\mathcal{F}$ of subsets of $[n]$ is called an (n,k,ℓ)-*avoiding selector*, where $1 \leq \ell < k \leq n$, if for every non-empty subset $S \subseteq [n]$ such that $|S| \leq k$ and for any subset $R \subseteq S$ of size at most ℓ, there is an element $a \in S \setminus R$ for which there exists a set $F \in \mathcal{F}$ such that $|F \cap S| = \{a\}$.*

The following fact follows directly from Definition 1, see also [6, 9].

▶ **Fact 1.** *Suppose we are given an (n,k,ℓ)-avoiding selector $\mathcal{F}$ and a set S of size at most k. Then, the number of elements in S not "selected" by selector $\mathcal{F}$ (i.e., for which there is no set in the selector that intersects S on such singleton element) is smaller than $k-\ell$.*

▶ **Theorem 2** ([6, 9]). *There exists an (n,k,ℓ)-avoiding selector of size $O\left(\frac{k^2}{k-\ell}\log n\right)$, and moreover, an (n,k,ℓ)-avoiding selector of size $O\left(\frac{k^2}{k-\ell}\operatorname{polylog} n\right)$ can be efficiently deterministically constructed (in polynomial time of n) for some polylogarithmic function* polylog n*, locally by each node.*

3.1 The C2B Algorithm

The pseudocode of our main C2B algorithm is in Algorithm 1, and its subroutines in Algorithms 2 and 3. The algorithm proceeds in epochs $i = 1, \ldots \log\Delta$. In the beginning, each node has all its links not successfully realized – here by a link $\{v,w\}$ being realized we understand that up to the current round, an input message/ID sent by v has been successfully encoded by w (using a sequence of beeps) and vice versa (note that these are two different messages and were sent/encoded each in different rounds); the formal definition of link realization will be given later. The goal of the algorithm is to preserve the following invariant for epoch $i \geq 1$:

> At the end of epoch $i = 1, \ldots, \log\Delta$, each vertex has less than $\kappa_i = \Delta/2^i$ incident links not realized.

We set an auxiliary value $\kappa_0 = \Delta$, which corresponds to the maximum number of adjacent links per node at the beginning of the computation. For ease of presentation, we assume that node IDs come from the range $[1,n]$. Note that in all the formulas, the number of possible IDs appears only under logarithms, so the algorithm and analysis for range $[1,n^c]$ are the same.

Algorithm for epoch i: Preliminaries and main concepts

Epoch i proceeds in subsequent batches of $2\log n$ rounds, each batch is called a **super-round**. In a single super-round, a node can constantly listen or keep beeping according to some 0-1 sequence of length $2\log n$, where 1 corresponds to beeping in the related round and 0 means staying silent. The sequences that the nodes use during the algorithm are **extended-IDs**, defined as follows: the first $\log n$ positions contain an ID of some node in $\{1,\ldots,n\}$, while the next $\log n$ positions contain the same ID with the bits flipped, that is, with ones swapped to zeros and vice versa. Note that extended-IDs are pairwise different, and each of them contains exactly $\log n$ ones and $\log n$ zeros. We say that a node v *beeps an extended-ID of node w in a super-round s* if, within super-round s, node v beeps only in rounds corresponding to positions with 1's in the extended ID of w (w could be a different node id than v). We say that a node w *receives an extended-ID of a node v in a super-round s* if:

- w does not beep in super-round s,
- the sequence of noise/silence heard by w in super-round s form an extended-ID of v.

From the perspective of receiving information in a super-round, all other cases not falling under the above definition of receiving an extended-ID, i.e., when a node is not silent in the super-round or receives a sequence of beeps that does not form any extended-ID, are ignored by the algorithm, in the sense that it could be treated as meaningless information noise.

Analogously to extended-ID's, nodes create an **extended-message** by taking the binary representation of the message of logarithmic length and transforming it to a $2\log n$ binary sequence in the same way as an extended-ID is created from the binary ID of a node. An extended-message, as well as an extended-ID, is easily decodable after being received without interruptions from other neighbors.

A specification of the conditions to achieve one-to-one communication, which is given "for free" in the CONGEST model, is crucial. An illustration of the following handshake communication procedure is shown in the full version of this work [19]. We say that our algorithm **realizes link** $\{v,w\}$ if the following are satisfied:

(a) there are three consecutive super-rounds (called "responding") in which v beeps an extended-ID of itself followed by an extended-ID of w and then by extended-message of v addressed to w, and w receives them in these super-rounds; intuitively, it corresponds to the situation when v "tells" w that it dedicates these three super-rounds for communication from itself to w, and w receives this information;

(b) there are three consecutive super-rounds (called "confirming") in which w beeps an extended-ID of itself followed by an extended-ID of v and by its extended-message addressed to v, and v receives them in these super-rounds; intuitively, it corresponds to the situation when w "tells" v that it dedicates these three super-rounds for communication from itself to v, and v receives this information;

(c) there is a super-round, not earlier than the one specified in point (a), at the end of which node w locally marks link $\{v,w\}$ as realized, and analogously, there is a super-round, not earlier than the one specified in point (b), at the end of which node v locally marks link $\{v,w\}$ as realized.

It is straightforward to see that in super-rounds specified in points (a) and (b), a multi-directional communication between v and w takes place – by sending and receiving both "directed pairs" of extended-IDs of these two nodes, each of them commits that the super-rounds specified in points (a) and (b) are dedicated for sending a message dedicated to the other node, and vice versa. Additionally, in some super-round(s) both nodes commit that it has happened (c.f., point (c) above).

Algorithm 2 C2B algorithm for announcer node v.

```
Procedure announcer(v, k_i)
    // announcing super-round
1   for each round r = 1, 2, ..., 2 log n do
2       if ⟨v⟩(r) = 1 then beep else listen
3   for each sub-phase a = 1, 2, ..., log k_i do
4       for b = 1, 2, ..., |F_{k_i/2^{a-2}, k_i/2^{a-1}}| do
            // responding 3 super-rounds
5           for 6 log n rounds do listen // announcer only listens
6           if some ⟨w⟩⟨v⟩⟨m_{w,v}⟩ was heard and {w, v} ∈ E(v) then
                // confirming 3 super-rounds
7               for each round r = 1, 2, ..., 2 log n do
8                   if ⟨v⟩(r) = 1 then beep else listen
9               for each round r = 1, 2, ..., 2 log n do
10                  if ⟨w⟩(r) = 1 then beep else listen
11              for each round r = 1, 2, ..., 2 log n do
12                  if ⟨m_{v,w}⟩(r) = 1 then beep else listen
13              E(v) ← E(v) \ {w, v} // link realized
14              if E(v) = ∅ then v stops executing
15          else
16              for 6 log n rounds do listen // wait to synchronize
```

Algorithm for epoch i: Structure

An epoch i is split into $|\mathcal{F}_{\Delta,k_i}|$ *phases*, for a given $(n, \Delta, \Delta - k_i)$-avoiding selector $\mathcal{F}_{\Delta,k_i}$ and parameter $k_i = \Delta/2^i$, parameterized by a variable j. Each phase starts with one *announcing super-round*, in which nodes in set $\mathcal{F}_{\Delta,k_i}(j)$ beep in pursuit to be received by some of their neighbors. This super-round is followed by $\log k_i$ *sub-phases*, parameterized by $a = 1, \ldots, \log k_i$. A sub-phase a uses sets from an $(n, k_i/2^{a-2}, k_i/2^{a-1})$-avoiding selector $\mathcal{F}_{k_i/2^{a-2},k_i/2^{a-1}}$ to determine who beeps in which super-round (together with additional rules to decide what extended-ID and extended-message to beep and how to confirm receiving them), and consists of $|\mathcal{F}_{k_i/2^{a-2},k_i/2^{a-1}}|$ 6-tuples of super-rounds (3 responding super-rounds and 3 confirming super-rounds). The goal of a phase is to realize links that were successfully received ("announced") in the first (announcing) super-round of this phase. This is particularly challenging in a distributed setting since many neighbors could receive such an announcement, but the links between them and the announcing node must be confirmed so that one-to-one communication between the announcer and responders could take place in different super-rounds (in one super-round, a node can receive only logarithmic-size information).

Definitions and notation. $\mathcal{F}_{\Delta,k_i}$ is a locally computed $(n, \Delta, \Delta - k_i)$-avoiding selector, and for any $a = 1, \ldots, \log k_i$, $\mathcal{F}_{k_i/2^{a-2},k_i/2^{a-1}}$ is a (locally computed) $(n, k_i/2^{a-2}, k_i/2^{a-1})$-avoiding selector, as in Theorem 2. We denote the extended-ID of node x as $\langle x \rangle$, and the extended-message of node x for node y as $\langle m_{x,y} \rangle$, both given as a sequence of bits. For any sequence of bits s, $s(i)$ is the i^{th} bit of s.

Algorithm 3 C2B algorithm for responder node w.

```
Procedure responder(w, k_i)
1   status(u) ← nil
    // announcing super-round
2   for 2 log n rounds do listen // responder only listens
3   if some ⟨v⟩ was heard and {v, w} ∈ E(w) then
4       status(w) ← v-responsive
5   for each sub-phase a = 1, 2, ..., log k_i do
6       for b = 1, 2, ..., |F_{k_i/2^{a-2}, k_i/2^{a-1}}| do
7           if status(w) = v-responsive and w ∈ F_{k_i/2^{a-2}, k_i/2^{a-1}}(b) then
                // responding 3 super-rounds
8               for each round r = 1, 2, ..., 2 log n do
9                   if ⟨w⟩(r) = 1 then beep else listen
10              for each round r = 1, 2, ..., 2 log n do
11                  if ⟨v⟩(r) = 1 then beep else listen
12              for each round r = 1, 2, ..., 2 log n do
13                  if ⟨m_{w,v}⟩(r) = 1 then beep else listen
                // confirming 3 super-rounds
14              for 6 log n rounds do listen // responder only listens
15              if some ⟨v⟩⟨w⟩⟨m_{v,w}⟩ was heard then
16                  status(w) ← nil
17                  E(w) ← E(w) \ {v, w} // link realized
18                  if E(w) = ∅ then w stops executing
19          else
20              for 12 log n rounds do listen // wait to synchronize
```

3.2 Analysis of the C2B Algorithm

Recall that the algorithm proceeds in synchronized super-rounds, each containing a subsequent $2 \log n$ rounds. Therefore, our analysis assumes that the computation is partitioned into consecutive super-rounds and, unless stated otherwise, it focuses on correctness and progress in super-rounds. Recall also that each node either stays silent (no beeping at all) or beeps an extended ID of some node or an extended message of one node addressed to one of its neighbors in a super-round. The missing proofs are deferred to the full version of this work [19].

In the next two technical results, we state and prove the facts that receiving an extended-ID by a node w in a super-round can happen if and only if there is *exactly one neighbor* of w has been beeping *the same extended-ID* during the considered super-round.

▶ **Fact 2** (Single beeping). *If during a super-round, exactly one neighbor of a node w beeps an extended-ID of some z, then w receives this extended-ID in this super-round.*

Proof. Directly from the definition of receiving an extended-ID. ◀

▶ **Lemma 3** (Correct receiving). *During the algorithm, if a node w receives some extended-ID of z in a super-round, then some unique neighbor v of w has been beeping an extended-ID of z in this super-round while all other neighbors of w have been silent. The above holds except, possibly, some second responding super-rounds, in which a node can receive an extended-ID of z that has been beeped by more than one neighbor.*

We now prove that link realization implemented by our algorithm is consistent with the definition – it allocates in a distributed way super-rounds for bi-directional communication of distinct messages.

▶ **Lemma 4** (Correct realization). *If a node v (locally) marks some link $\{v, w\}$ as realized, which may happen only at the end of a second confirming super-round, the link has been realized by then.*

As mentioned earlier in the description of the phase, the goal of a phase j (of epoch i) is to assure that any node v that was received by some other nodes w in the announcing super-round, gets all such links $\{v, w\}$ realized by the end of the phase (and vice versa, because the condition on the realization by this algorithm is symmetric). The next step is conditional progress in a sub-phase a of a phase j.

▶ **Lemma 5** (Sub-phase progress). *Consider any node v and suppose that in the beginning of sub-phase a of phase j, there are at most $\Delta/2^{i+a-2}$ nodes w such that w is (j, v)-responsive and it does not mark link $\{v, w\}$ as realized. Then, by the end of the sub-phase, the number of such nodes is reduced to less than $\Delta/2^{i+a-1}$.*

▶ **Lemma 6** (Phase progress). *Consider a phase j of epoch i and assume that in the beginning, there are at most $2k_i$ non-realized incident links to any node. Every node w that becomes (j, v)-responsive in the first (announcing) super-round of the phase, for some v, mark locally the link $\{v, w\}$ as realized during this phase. And vice versa, also node v marks locally that link as realized.*

The next lemma proves the invariant for epoch i, assuming that it holds in the previous epochs.

▶ **Lemma 7** (Epoch invariant). *The invariant for epoch $i \geq 1$ holds.*

▶ **Theorem 8.** *The* C2B *algorithm deterministically and distributedly simulates any round of any algorithm designed for the CONGEST networks in $O(\Delta^2 \operatorname{polylog} n \log \Delta)$ beeping rounds, where the $\operatorname{polylog} n$ is the square of the (poly-)logarithm in the construction of avoiding-selectors in Theorem 2 multiplied by $\log n$.*

Proof. By Lemma 7, each epoch i reduces by at least half the number of non-realized incident links. We next count the number of rounds in each epoch by counting the number of super-rounds and multiplying the result by the $O(\log n)$ length of each super-round. Recall that link realization means that some triples of responding and confirming super rounds were not interrupted by other neighbors of both end nodes of that link; therefore, the attached extended messages (in the third super-rounds in a row) were correctly received. Thus, the local exchange of messages addressed to specific neighbors took place successfully.

Each sub-phase a has $O(\Delta^2 \operatorname{polylog} n)$ super-rounds, because for each set in of the $(n, k_i/2^{a-1}, k_i/2^a)$-avoiding selector $\mathcal{F}_{k_i/2^{a-1}, k_i/2^a}$, there are four super-rounds and the selector itself has $O((k_i/2^a) \operatorname{polylog} n)$ set, by Theorem 2.

Therefore, the total number of super-rounds in all sub-phases executed within the loops in Line 3 of Algorithm 2 and Line 5 of Algorithm 3 is

$$O(\sum_{a=1}^{\log k_i} (k_i/2^a)\ \textit{polylog}\ n) \leq O(k_i\ \textit{polylog}\ n)\ .$$

Within one phase, they are executed as many times as the number of announcing super-rounds. The number of such super-rounds in a phase is $|\mathcal{F}_{\Delta,k_i}|$, which is $O((\Delta^2/k_i)\ \textit{polylog}\ n)$ by Thm. 2. Thus, the total number of super-rounds in a phase is $O((\Delta^2/k_i)\ \textit{polylog}\ n \cdot k_i\ \textit{polylog}\ n) \leq O(\Delta^2\ \textit{polylog}\ n)$, where the final *polylog* n is a square of the (poly-)log in Thm. 2.

Since there are $\log \Delta$ epochs, the total number of super-rounds is $O(\Delta^2\ \textit{polylog}\ n \log \Delta)$, which is additionally multiplied by $O(\log n)$ – the length of each super-round – if we want to refer the total number of beeping rounds. ◀

Maximal Independent Set (MIS). To demonstrate that the above efficient simulator can yield efficient results for many graph problems, we apply it to the $O(\log^5 n)$ MIS algorithm in [20] to improve polynomially (with respect to Δ) the best-known solutions for MIS (c.f. [5]):

▶ **Corollary 9.** *MIS can be solved deterministically on any network of maximum node-degree Δ in $O(\Delta^2$ polylog $n)$ beeping rounds.*

4 Multi-hop Simulation

We generalize the simulation at distance 1 to the following *B-bit h-hop simulation* problem: each node has messages, potentially different, of size at most B bits addressed to any other node, and it needs to deliver them to all destination nodes within distance at most h hops. If each node has only a single message of size at most B bits to be delivered to all nodes within distance at most h hops, then we call this restricted version *B-bit h-hop Local Broadcast*. Note also that we do not require messages addressed to nodes of distance larger than h to be delivered. Below we generalize the lower bound for single-hop simulation in [12] to multi-hop simulation and multi-hop local broadcast.

▶ **Theorem 10.** *There is an adversarial network of size $\Theta(\Delta^h)$ such that any B-bit h-hop simulation algorithm requires $\Omega(B\Delta^{h+1})$ beeping rounds to succeed with probability more than $2^{-\frac{1}{2}\cdot B(\Delta-1)^{h-2}(\Delta/2)^3} = 2^{-\Theta\left(B\Delta^{h+1}\right)}$.*

Proof. Problem instance. We describe the construction of the adversarial network and input set of messages used to prove our lower bound as follows (refer to Figures 2 and 3). Consider a full bipartite graph $K_{\Delta/2,\Delta/2}$, with one part called T and the other R. We focus on transmissions going towards nodes in R, hence nodes in R will be called receivers, while nodes in $T = T_1$ will be called the first of h layers of transmitters.

Each node in T_1 will be a root of an $(h-1)$-depth tree of transmitters. We create a second layer of transmitters T_2 composed of $(\Delta/2)^2$ nodes. Each node in T_1 (already connected to each node in R) will also be connected to different $\Delta/2$ nodes in T_2. For subsequent layers of transmitters, that is each layer T_i, for $3 \leq i \leq h$, will be composed of $(\Delta/2)^2(\Delta-1)^{i-2}$ nodes. Each node in layer T_{i-1}, for $3 \leq i \leq h$, will be connected to different $\Delta - 1$ nodes in layer T_i. Note that each node in the defined network has at most Δ neighbors.

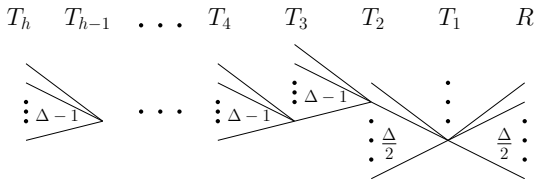


Figure 2 An illustration of the structure of the graph. The graph is partitioned into vertical layers $T_h, \ldots, T_1, R$. The graph is branching out heavily, so we show only a path from an arbitrary node in T_1 layer to an arbitrary node in T_h layer, with all the edges incident to the path. The numbers between the layers denote the number of edges between the layers that are incident to the path or on the path. Recall that layers T_1 and R have $\Delta/2$ nodes each, while the other layers have significantly more nodes, but we only show nodes that are adjacent to the considered path.

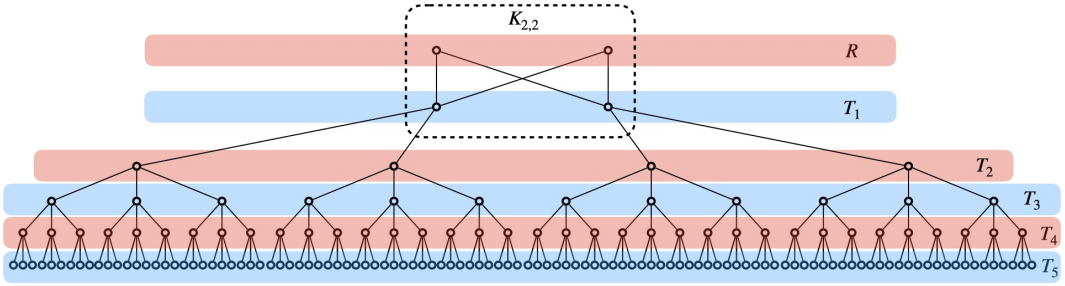


Figure 3 Illustration for Theorem 10. Example of adversarial graph for $\Delta = 4$ and $h = 5$.

We define now the input set of messages as follows. Let each node $v \in T_h$ have a B-bit message $m_{v \to u}$ to each node $u \in R$. We choose those messages uniformly at random. We will show that just these messages cannot be relayed efficiently and we do not need any other messages in our problem instance.[6]

Multihop Simulation. Here we analyze the multihop simulation algorithms.

There are $(\Delta/2)^2(\Delta-1)^{h-2}$ nodes in T_h, and each of them has $\Delta/2$ (possibly different) messages, one for each node in R. Therefore, there are $(\Delta/2)^3(\Delta-1)^{h-2} = \Theta(\Delta^{h+1})$ messages to nodes in R that are passing through nodes in T_1.

Let $\mathcal{R}$ be the concatenated string of local randomness in all the nodes in R. The output of any receiver $u \in R$ must depend only on $\mathcal{R}$, node IDs and the pattern of beeps and silences of nodes in T_1.

There are 2^t possible patterns of beeps and silences in t rounds. Therefore, the output of nodes in R must be one of the 2^t possible distributions, where a distribution is over the randomness of $\mathcal{R}$. The correct output of nodes in R is a string $\{0,1\}^{B(\Delta-1)^{h-2}(\Delta/2)^3} = \{0,1\}^{\Theta(B\Delta^{h+1})}$ chosen uniformly at random (since the input messages of nodes in T_h were chosen uniformly at random). Therefore, the probability of picking the correct result is at most $2^{t-B(\Delta-1)^{h-2}(\Delta/2)^3}$, and any algorithm that finishes within $t \leq \frac{1}{2} \cdot B(\Delta-1)^{h-2}(\Delta/2)^3$ rounds has at most $2^{-\frac{1}{2} \cdot B(\Delta-1)^{h-2}(\Delta/2)^3}$ probability of outputting the correct answer. ◀

[6] Alternatively we can make all other messages known to the optimal algorithm, e.g., by setting them to 0^B.

Algorithm

A simple algorithm would repeatedly use a 1-hop Local Broadcast routine to flood the network with the messages until nodes at a distance h received the messages. This, however, can take $\Omega(\Delta^{2h})$ rounds. Instead, we limit the flooding by only sending messages along the shortest paths to their destinations using a 1-hop simulation algorithm. The details of the algorithm as well as its analysis are presented next.

In the beginning, nodes use a standard protocol to disseminate their IDs up to distance h. They do it in h subsequent epochs, each epoch i of t_i rounds sufficient to run our Local Broadcast (see full version of this work [19]) for messages of size $\Delta^i \log n$. These messages contain different IDs learned by the node at the beginning of the current epoch. A direct inductive argument, also using the property that there are at most Δ^i nodes at a distance at most i, shows that at the end of epoch i, each node knows the IDs of all nodes at a distance at most i from it. Additionally, each node records in which epoch i it learned each known ID v for the first time and from which of its neighbors w – and stores this information as a triple (v, w, i). The invariant for $i = h$ proves that at the end of epoch h, each node knows IDs of all nodes of distance at most h from it. The round complexity is $\sum_{i=1}^{h} \Delta^i \log n \cdot \Delta^2 \log n = O(\Delta^{h+2} \log^2 n)$, and as will be seen later, it is subsumed by the round complexity of the second part of our algorithm (as the *polylog* n function in Theorem 8 is asymptotically bigger than $\log^2 n$).

Note that a sequence of triples $(v, w_1 = v, 1), \ldots, (v, w_\ell, \ell)$, stored at nodes $w_2, w_3, \ldots, w_\ell, w_{\ell+1} = u$ respectively, represents a shortest path to node v starting from the node u; the length of that path is ℓ.

In the second part, nodes also proceed in epochs, but this time each epoch i takes t_i^* rounds sufficient to execute 1-hop simulation algorithm from Section 3 (see Theorem 8) for point-to-point messages of size $(B + \log n)\Delta^h$. Here B denotes the known upper bound on the size of any input message. In epoch i, every node u transmits a (possibly different) message of size $(B + \log n)\Delta^h$ to each neighbor w. Such a message contains all the input messages of nodes within $i - 1$ distance and the recipients of these messages such that w is the next node on the saved shortest path to the recipient. The messages have already traveled $i - 1$ hops, so their destination is at most $h - (i - 1)$ hops away. More specifically, the message from node u addressed to a neighbor w in epoch i contains pairs $(v, m_{z \to v})$, where v is such that (v, w, i') is stored at the node for some $i' \leq h - (i - 1)$ and $m_{z \to v}$ is a message received by the node u in epoch $i - 1$ (in case of $i - 1 = 0$, it is the original message of the node addressed to v). A direct inductive argument shows that at the end of each epoch i a node knows at most Δ^i messages addressed to any node v of distance $\ell \leq h - i$ from the node. This invariant is based on the following arguments:

- Because there is a unique neighbor w of the node u such that a triple (v, w, ℓ) is stored at the node, the number of such nodes v of distance at most $h - i + 1$ from the node u is at most Δ^{h-i+1},
- by the end of epoch $i - 1$, node u could receive messages to be relayed to v from Δ^{i-1} different nodes at distance $i - 1$,
- each message contains up to B-bit long original message and an ID of length $\log n$,
- hence, messages of size at most $(B + \log n)\Delta^{i-1} \cdot \Delta^{h-i+1} = (B + \log n)\Delta^h$ are being sent to each neighbor in epoch i, and by definition – epoch i has sufficient number of rounds to deliver them.

The invariant for $i = h$ proves the desired property of B-bit h-hop simulation. The total number of rounds is $O(h \cdot (B + \log n)\Delta^h \cdot \Delta^2 \textit{ polylog } n) \subseteq O(h \cdot B\Delta^{h+2} \textit{ polylog } n)$, where factor h comes from the number of epochs, each sending at most Δ point-to-point messages of size at most $(B + \log n)\Delta^h$ to neighbors (by the invariant) using the 1-hop simulation protocol with overhead $O(\Delta^2 \textit{ polylog } n)$ (by Theorem 8). Hence we proved the following.

▶ **Theorem 11.** *There is a distributed deterministic algorithm solving the B-bit h-hop simulation problem in a beeping network in $O(h \cdot B\Delta^{h+2}$ polylog $n)$ rounds.*

5 Conclusions

We provided deterministic distributed algorithms to efficiently simulate a round of algorithms designed for the CONGEST model on the Beeping Networks. This allowed us to improve polynomially the time complexity of several (also graph) problems on Beeping Networks. The first simulation by the Local Broadcast algorithm is shorter by a polylogarithmic factor than the other, more general one – yet still powerful enough to implement some algorithms, including the prominent solution to Network Decomposition [20]. The more general one could be used for solving problems such as MIS. We also considered efficient pipelining of messages via several layers of BN. Two important lines of research arise from our work. First, whether some (graph) problems do not need local broadcast to be solved deterministically, and whether their time complexity could be asymptotically below Δ^2. Second, could a lower bound on any deterministic local broadcast algorithm, better than $\Omega(\Delta \log n)$, be proved?

References

1 Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. Beeping a maximal independent set. *Distributed Computing*, 26(4):195–208, 2013. doi:10.1007/S00446-012-0175-7.

2 Yehuda Afek, Noga Alon, Omer Barad, Eran Hornstein, Naama Barkai, and Ziv Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, 331(6014):183–185, 2011.

3 Yagel Ashkenazi, Ran Gelles, and Amir Leshem. Brief announcement: Noisy beeping networks. In *Proceedings of the 39th Symposium on Principles of Distributed Computing, PODC 2020*, pages 458–460, 2020. doi:10.1145/3382734.3405705.

4 Joffroy Beauquier, Janna Burman, Peter Davies, and Fabien Dufoulon. Optimal multi-broadcast with beeps using group testing. In *Structural Information and Communication Complexity: 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1–4, 2019, Proceedings 26*, pages 66–80. Springer, 2019. doi:10.1007/978-3-030-24922-9_5.

5 Joffroy Beauquier, Janna Burman, Fabien Dufoulon, and Shay Kutten. Fast beeping protocols for deterministic mis and (δ+ 1)-coloring in sparse graphs. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1754–1762. IEEE, 2018. doi:10.1109/INFOCOM.2018.8486015.

6 Annalisa De Bonis, Leszek Gasieniec, and Ugo Vaccaro. Optimal two-stage algorithms for group testing problems. *SIAM J. Comput.*, 34(5):1253–1270, 2005. doi:10.1137/S0097539703428002.

7 Arnaud Casteigts, Yves Métivier, John Michael Robson, and Akka Zemmari. Design patterns in beeping algorithms: Examples, emulation, and analysis. *Information and Computation*, 264:32–51, 2019. doi:10.1016/J.IC.2018.10.001.

8 Bogdan S Chlebus, Gianluca De Marco, and Muhammed Talo. Naming a channel with beeps. *Fundamenta Informaticae*, 153(3):199–219, 2017. doi:10.3233/FI-2017-1537.

9 Bogdan S. Chlebus and Dariusz R. Kowalski. Almost optimal explicit selectors. In Maciej Liskiewicz and Rüdiger Reischuk, editors, *Fundamentals of Computation Theory, 15th International Symposium, FCT 2005, Lübeck, Germany, August 17-20, 2005, Proceedings*, volume 3623, pages 270–280, 2005. doi:10.1007/11537311_24.

10 Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *Distributed Computing: 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings 24*, pages 148–162. Springer, 2010. doi:10.1007/978-3-642-15763-9_15.

11 Artur Czumaj and Peter Davies. Communicating with beeps. *Journal of Parallel and Distributed Computing*, 130:98–109, 2019. doi:10.1016/J.JPDC.2019.03.020.

12 Peter Davies. Optimal message-passing with noisy beeps. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023*, pages 300–309, 2023. doi:10.1145/3583668.3594594.

13 Julius Degesys, Ian Rose, Ankit Patel, and Radhika Nagpal. Desync: Self-organizing desynchronization and tdma on wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 11–20, 2007. doi:10.1145/1236360.1236363.

14 Fabien Dufoulon, Janna Burman, and Joffroy Beauquier. Beeping a deterministic time-optimal leader election. In *32nd International Symposium on Distributed Computing, DISC 2018*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

15 Fabien Dufoulon, Yuval Emek, and Ran Gelles. Beeping shortest paths via hypergraph bipartite decomposition. *arXiv preprint arXiv:2210.06882*, 2022. doi:10.48550/arXiv.2210.06882.

16 Klim Efremenko, Gillat Kol, and Raghuvansh Saxena. Interactive coding over the noisy broadcast channel. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 507–520, 2018. doi:10.1145/3188745.3188884.

17 Roland Flury and Roger Wattenhofer. Slotted programming for sensor networks. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 24–34, 2010. doi:10.1145/1791212.1791216.

18 Klaus-Tycho Förster, Jochen Seidel, and Roger Wattenhofer. Deterministic leader election in multi-hop beeping networks. In *Distributed Computing: 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings 28*, pages 212–226. Springer, 2014.

19 Pawel Garncarek, Dariusz R. Kowalski, Shay Kutten, and Miguel A. Mosteiro. Beeping deterministic congest algorithms in graphs, 2025. doi:10.48550/arXiv.2502.13424.

20 Mohsen Ghaffari, Christoph Grunau, and Vaclav Rozhoň. Improved deterministic network decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 2904–2923. SIAM, 2021. doi:10.1137/1.9781611976465.173.

21 Mohsen Ghaffari and Bernhard Haeupler. Near optimal leader election in multi-hop radio networks. In *Proceedings of the twenty-fourth annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 748–766. SIAM, 2013. doi:10.1137/1.9781611973105.54.

22 Zygmunt J Haas and Jing Deng. Dual busy tone multiple access (dbtma)-a multiple access control scheme for ad hoc networks. *IEEE Transactions on Communications*, 50(6):975–985, 2002. doi:10.1109/TCOMM.2002.1010617.

23 Stephan Holzer and Nancy Lynch. Brief announcement: beeping a maximal independent set fast. In *30th International Symposium on Distributed Computing, DISC 2016*, 2016.

24 Kokouvi Hounkanli, Avery Miller, and Andrzej Pelc. Global synchronization and consensus using beeps in a fault-prone multiple access channel. *Theoretical Computer Science*, 806:567–576, 2020. doi:10.1016/J.TCS.2019.09.020.

25 Kokouvi Hounkanli and Andrzej Pelc. Deterministic broadcasting and gossiping with beeps. *arXiv preprint arXiv:1508.06460*, 2015. arXiv:1508.06460.

26 Kokouvi Hounkanli and Andrzej Pelc. Asynchronous broadcasting with bivalent beeps. In *Structural Information and Communication Complexity: 23rd International Colloquium, SIROCCO 2016, Helsinki, Finland, July 19-21, 2016, Revised Selected Papers 23*, pages 291–306. Springer, 2016. doi:10.1007/978-3-319-48314-6_19.

27 Peter Jeavons, Alex Scott, and Lei Xu. Feedback from nature: simple randomised distributed algorithms for maximal independent set selection and greedy colouring. *Distributed Computing*, 29:377–393, 2016. doi:10.1007/S00446-016-0269-8.

28 Jason J. Moore, Alexander Genkin, Magnus Tournoy, Joshua L. Pughe-Sanford, Rob R. de Ruyter van Steveninck, and Dmitri B. Chklovskii. The neuron as a direct data-driven controller. *Proceedings of the National Academy of Sciences*, 121(27):e2311893121, 2024. doi:10.1073/pnas.2311893121.

29 Arik Motskin, Tim Roughgarden, Primoz Skraba, and Leonidas Guibas. Lightweight coloring and desynchronization for networks. In *IEEE INFOCOM 2009*, pages 2383–2391. IEEE, 2009. doi:10.1109/INFCOM.2009.5062165.

30 Saket Navlakha and Ziv Bar-Joseph. Distributed information processing in biological and computational systems. *Communications of the ACM*, 58(1):94–102, 2014. doi:10.1145/2678280.

31 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, Philadelphia, PA, USA, 2000.

32 Fouad Tobagi and Leonard Kleinrock. Packet switching in radio channels: Part ii-the hidden terminal problem in carrier sense multiple-access and the busy-tone solution. *IEEE Transactions on Communications*, 23(12):1417–1433, 1975. doi:10.1109/TCOM.1975.1092767.

A Unified FPT Framework for Crossing Number Problems

Éric Colin de Verdière
LIGM, CNRS, Univ Gustave Eiffel, F-77454 Marne-la-Vallée, France

Petr Hliněný
Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract

The basic (and traditional) *crossing number problem* is to determine the minimum number of crossings in a topological drawing of an input graph in the plane. We develop a unified framework that smoothly captures many generalized crossing number problems, and that yields fixed-parameter tractable (FPT) algorithms for them not only in the plane but also on surfaces.

Our framework takes the following form. We fix a surface $\mathcal{S}$, an integer r, and a map κ from the set of topological drawings of graphs in $\mathcal{S}$ to $\mathbb{Z}_+ \cup \{\infty\}$, satisfying some natural monotonicity conditions, but essentially describing the allowed drawings and how we want to count the crossings in them. Then deciding whether an input graph G has an allowed drawing D on $\mathcal{S}$ with $\kappa(D) \leq r$ can be done in time quadratic in the size of G (and exponential in other parameters). More generally, we may take as input an edge-colored graph, and distinguish crossings by the colors of the involved edges; and we may allow to perform a bounded number of edge removals and vertex splits to G before drawing it. The proof is a reduction to the embeddability of a graph on a two-dimensional simplicial complex.

This framework implies, in a unified way, quadratic FPT algorithms for many topological crossing number variants established in the graph drawing community. Some of these variants already had previously published FPT algorithms, mostly relying on Courcelle's metatheorem, but for many of those, we obtain an algorithm with a better runtime. Moreover, our framework extends, at no cost, to these crossing number variants in any fixed surface.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph theory

Keywords and phrases computational geometry, fixed-parameter tractability, graph drawing, graph embedding, crossing number, two-dimensional simplicial complex, surface

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.21

Related Version *Full Version*: https://arxiv.org/abs/2410.00206v3

Funding *Petr Hliněný*: Part of this work was done while this author was invited professor at LIGM, Marne-la-Vallée, supported by Paris-Est Sup.

1 Introduction

The (traditional) *crossing number* problem, minimizing the number of pairwise edge crossings in topological drawings of an input graph in the plane, is a long-standing and central task in graph drawing and visualization that comes in many established flavors; see the extensive dynamic survey by Schaefer [45]. In this paper, we give a framework that smoothly captures many of the variants of crossing numbers, and in a unified way provide efficient algorithms for them when parameterized by the solution value (i.e., the respective crossing number).

Many flavors of crossing numbers. There is currently a surge of crossing number variants. A motivation for such variations is that, in practical drawing applications, not every crossing or crossing pattern of edges may be "equal" to other ones. One may, e.g., want to avoid


© Éric Colin de Verdière and Petr Hliněný;
licensed under Creative Commons License CC-BY 4.0



mutual crossings of important edges. Or, to allow crossings only within specific parts of a graph, and not between unrelated parts. Or, to exclude crossings of edges of some type. Or, to allow only certain "comprehensible" crossing patterns in order to help visualize the graph. This is the main focus of the recent research direction called "beyond planarity" [15].

Paraphrasing Schaefer [45, Chapter 2], a crossing number variant minimizes, over all *allowed drawings* D of an input graph G in some specified *host surface*, an *objective function* related to the crossings in D. The *allowed drawings* may, e.g., restrict the crossings on one edge or in a local pattern of crossing edges, or prescribe or forbid certain local properties of the drawing (also depending on types or colors of edges). The *host surface* is usually the plane, but can be any fixed surface, orientable or not. The *objective function* is often the number of crossings between edges, but other possibilities include, e.g., the number of edges involved in crossings, or the number of pairs of edges that cross an odd number of times.

Existing algorithms for the traditional version. One usually considers the decision version of the (traditional) crossing number problem: Given an input graph G of size n, and an integer r, does G have a crossing number at most r? This problem is already NP-hard in the plane, as proved by Garey and Johnson in 1983 [18], and even in very specific cases [8, 24, 26]. Moreover, the problem is APX-hard [7], and the best known polynomial-time approximation algorithm [10] gives an approximation factor that is subpolynomial in n only for bounded-degree graphs.

Thus, the main current focus is on algorithms that are fixed-parameter tractable (FPT) in r – the runtime has the form $f(r) \cdot \mathsf{poly}(n)$, where f is a computable function of the *parameter* r. For fixed r, Grohe [20] has described an $O(n^2)$-time algorithm, and Kawarabayashi and Reed [32] have announced an $O(n)$-time algorithm. Both rely on Courcelle's metatheorem [14], which automatically entails a huge dependence in the parameter r.

Two recent approaches avoid resorting to Courcelle's theorem. First, Colin de Verdière, Magnard, and Mohar [13] have studied the problem of embedding a graph in a *two-dimensional simplicial complex* (2-complex for short), a topological space obtained from a surface by adding isolated edges and identifying vertices. Among other motivations for this problem, they observe [13, Introduction] that the crossing number problem reduces to the embeddability of the input graph in a certain 2-complex depending only on r. Then Colin de Verdière and Magnard [12] have shown that the embeddability of a graph of size n in a 2-complex of size C can be tested in $2^{\mathsf{poly}(C)} \cdot n^2$ time. Hence, this results in a $2^{\mathsf{poly}(r)} \cdot n^2$-time algorithm for the crossing number problem, which, moreover, extends to any fixed surface.

Second, even more recently, Lokshtanov, Panolan, Saurabh, Sharma, Xue, and Zehavi [35] have given a $2^{\mathcal{O}(r \log r)} \cdot n$-time algorithm for the traditional crossing number problem in the plane, by a reduction to bounded treewidth and dynamic programming, all in time linear in n.

Existing algorithms for other variants. For other flavors of crossing numbers, which are (typically) also NP-hard, the literature on FPT algorithms is scarce; besides Pelsmajer, Schaefer, and Štefankovič [41] giving FPT algorithms for the odd and pair crossing numbers, we are only aware of two very recent papers, which bring a general approach to multiple crossing number flavors and which we now detail. (The various flavors of crossing numbers are defined in Section 5.)

First, Münch and Rutter [37], extending Grohe's approach [20], provide a framework for quadratic FPT algorithms for the crossing number of several types of beyond-planar drawings of graphs in the plane, characterized by forbidden combinatorial crossing patterns. This includes the crossing number of k-planar, k-quasi-planar, min-k-planar, fan-crossing, and fan-crossing free drawings of a graph for any constant k.

Second, Hamm, Klute, and Parada [22], in a very recent preprint, have announced a generalization of the previous framework [37], also handling forbidden topological crossing patterns in drawing types in the plane and, more importantly, bringing the possibility of handling predrawn parts of the input graph (as already known for the traditional crossing number [21]). The dependence in the size n of the input graph is an unspecified polynomial.

The above results all rely on Courcelle's theorem. Apart from that, we are only aware of sporadic results for isolated variants. Kawarabayashi and Reed [32] and Jansen, Lokshtanov, and Saurabh [30] both claim, without details, a linear FPT algorithm to compute the skewness of a graph, and Nöllenburg et al. [39] give a non-uniform FPT algorithm for the splitting number in surfaces. All mentioned papers except Nöllenburg et al. are restricted to the plane.

Our contributions. Our general framework takes the following form. We fix a surface $\mathcal{S}$ and an objective function κ from the set $\mathcal{D}$ of (possibly edge-colored) topological drawings of graphs on $\mathcal{S}$ to the set $\mathbb{Z}_+ \cup \{\infty\}$; the allowed drawings D are those that satisfy $\kappa(D) < \infty$. The map κ has to satisfy some natural monotonicity conditions, defined later. Given an integer r, we prove that deciding whether an input graph G has a drawing D on $\mathcal{S}$ with $\kappa(D) \leq r$ can be done in quadratic time in the size n of G, and exponential in a polynomial of the other parameters, namely the genus of $\mathcal{S}$, the maximum size of a "fully crossing" drawing $D^\times$ satisfying $\kappa(D^\times) \leq r$, and the integer r. This remains true if one allows a bounded number of (possibly color-restricted) edge removals and vertex splits to G before drawing it; these numbers are also parameters. The proof is a reduction to the already mentioned problem of deciding the embeddability of a graph on a 2-complex [12]; while the general strategy is intuitive, many subtle technical details need to be overcome to make it all work.

We deduce FPT algorithms with runtime quadratic in the size n of the input graph, and exponential in a polynomial of the other parameters, for many established crossing number variants; we now survey some of them. First, we can restrict ourselves to many established drawing styles, such as k-planar, k-quasi-planar, min-k-planar, fan-crossing, weakly and strongly fan-planar, k-gap, and fixed-rotation drawings (k being an additional fixed parameter). For the k-gap and fixed rotation cases, no FPT algorithms were known. Second, we may assign colors to the edges of the input graph and count the crossings differently depending on the colors of the edges involved, leading to the first FPT algorithms on the joint crossing number on surfaces and its generalizations. Third, assuming suitable additional properties, we can handle problems that do not count the crossings, but rather the number of edges, or pairs of edges, involved in crossings, such as the edge, pair, and odd crossing numbers (no FPT algorithm was known for the edge crossing number). And fourth, since we allow for prior edge removals and vertex splits, our framework encompasses, e.g., the skewness, for which FPT algorithms were only sketched [30, 32] in the plane, and the splitting number, which was only known to admit a nonuniform FPT algorithm [39].

Last but not least, all these four aspects that define flavors of crossing numbers can be combined arbitrarily, and our main results are valid not only in the plane, but for arbitrary surfaces. For most flavors of crossing number, the arguments are direct, though they really differ according to the flavor; they essentially boil down to checking some technical conditions (see Definition 3.1), and to giving an algorithm (not necessarily polynomial) to check a given drawing of bounded size and to count its crossings (see Theorem 3.2). However, for a couple of crossing number flavors (in particular, when fixing the rotation scheme), additional ingredients are needed.

Comparison with the state of the art. Our main contribution is a convenient, versatile, and unified framework capturing a very general class of crossing number variants while giving FPT algorithms. In particular, as listed above, we get new FPT results for several established variants and their generalizations (to surfaces, or by combining the features of multiple variants). It is conceivable that some of the variants newly covered by our framework could be proved FPT using other techniques, perhaps the (very powerful) metatheorem of Courcelle, as already used in [20–22, 37]. However, such developments are extremely delicate even in isolated variants and lead to hard-to-describe algorithms. Indeed, first, checking a given drawing and counting its crossings must be formalized in the MSO_2 logic of graphs, which usually needs heavy tricks tailored to the specific case; second, there remains the specific and often highly nontrivial treewidth reduction step to be done. So far, such approaches have been successful only in the case of the plane.

Moreover, all approaches using Courcelle's theorem inherently come with a high computational cost. Indeed, they result in algorithms with a huge multi-level exponential dependence in the parameter r; for instance, it is an exponential tower of height at least four in the case of [20] and three in the case of [32], see the discussion in [35]. In contrast, our algorithms are singly exponential in a polynomial in r. Also, while Courcelle's theorem itself runs in linear time in the size n of the input graph, the necessary treewidth reduction step requires quadratic time, if not more, in the known approaches for the non-traditional crossing numbers. Our algorithms have a quadratic dependence in n, and the only bottleneck to a linear-time dependence is the quadratic runtime of the current best embeddability algorithm [12].

The frameworks used in the two very recent works mentioned above [22, 37], which both use treewidth reduction and Courcelle's theorem, can handle some of the drawing styles that we encompass, but different counting functions, edge removals and vertex splits, and nonplanar surfaces are out of their reach. However, it should not be difficult to extend their framework to handle edge-colored graphs, thanks to the MSO_2 logic being able to handle arbitrary sets of edges. On the other hand, the recent breakthrough of Lokshtanov et al. [35] for the traditional crossing number problem may fuel hope for more $2^{\mathcal{O}(r \log r)} \cdot n$-time algorithms, but tweaking all ingredients of that highly technical paper seems hard, even for isolated crossing number variants, and moreover this approach is inapplicable to surfaces other than the plane, due to the use of several inherently planar techniques (3-connectivity arguments and dependence on [30]).

Organization of the paper. After the preliminaries (Section 2), we describe our general framework and state our main result (Section 3), which is then proved (Section 4). We then list the numerous applications to crossing number variants (Section 5).

2 Preliminaries

Graphs and surfaces. In this paper, graphs are finite and undirected, but not necessarily simple unless specifically noted. The *size* of a graph G is the number of vertices plus the number of edges of G. A *vertex split* of a graph G at vertex v creates a new vertex v' and replaces some of the edges incident to v by making them incident to v' instead of v.

We follow [2, 36] for surface topology. A *surface* $\mathcal{S}$ is a topological space obtained from finitely many disjoint solid, two-dimensional triangles by identifying some of their edges in pairs. Surfaces are not necessarily connected. The *boundary* of $\mathcal{S}$ is the closure of the union of the unidentified edges. The *(Euler) genus* of a connected surface is twice its number of "handles", if it is orientable, and is its number of "crosscaps", otherwise. The *topological size*

of $\mathcal{S}$ equals $s := d+b+g$, where d is the number of connected components of $\mathcal{S}$, g is the sum of the (Euler) genera of its connected components, and b is the number of boundary components. The plane is not a surface according to our definition, but in this paper it can always be substituted with a disk, which is (homeomorphic to) a surface. A *self-homeomorphism* of $\mathcal{S}$ is just a homeomorphism from $\mathcal{S}$ to $\mathcal{S}$.

(Topological) drawings of graphs. A *curve* c in a topological space $\mathcal{X}$ is a continuous map from the closed interval $[0,1]$ into $\mathcal{X}$. The *relative interior* of c is the image, under c, of the open interval $(0,1)$. In a *drawing* D of a graph G in a topological space $\mathcal{X}$, vertices are represented by points of $\mathcal{X}$ and edges by curves in $\mathcal{X}$ such that the *endpoints* $c(0)$ and $c(1)$ of a curve c are the images of the end vertices of the corresponding edge. Moreover, we assume that distinct vertices are mapped to distinct points and that the relative interiors of the edges avoid the images of the vertices. Given a point $x \in \mathcal{X}$, its *multiplicity* in D is the number of pairs (c,t) such that $c(t) = x$, where c is a curve representing an edge of G in D, and $t \in (0,1)$. An *intersection point* (shortly an *intersection*) of D is a point with multiplicity at least two. We only consider drawings D with finitely many intersection points.

When considering a drawing D of a graph G on a *surface* $\mathcal{S}$, we *always* implicitly assume that G has no isolated vertex and that the image of D avoids the boundary of $\mathcal{S}$; these conditions are actually benign in a graph drawing context. In D, each edge e is subdivided into (finitely many) *pieces* by the intersection points along e. A point of multiplicity two corresponds to a *crossing* if no local perturbation of the curves can remove this intersection, or to a *tangency*, otherwise. D is *normal* [44] if every intersection point has multiplicity two and is a crossing (not a tangency). Moreover, D is *simple* if it is normal, no edge self-crosses, no two adjacent edges cross, and no two edges cross more than once.[1]

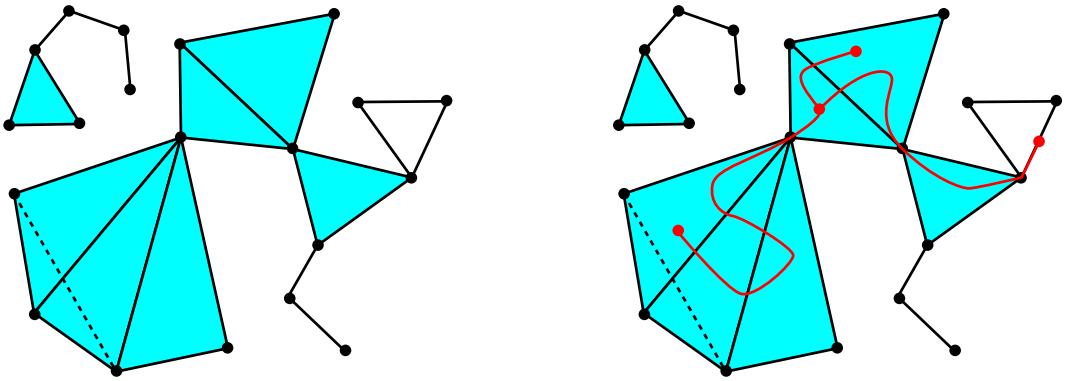
The (traditional) *crossing number* of a graph G in a surface $\mathcal{S}$ is the least number of crossings over all normal drawings of G in $\mathcal{S}$. Many variations exist [45], see Section 5.

We need to represent drawings of graphs on a surface $\mathcal{S}$ combinatorially, up to self-homeomorphisms of $\mathcal{S}$. Cellular embeddings can be represented by combinatorial maps [16], but we need to allow crossings, and moreover faces need not be disks. We can achieve this, ultimately relying on combinatorial maps. Details can be found in the full version.

2-complexes. In this paper, a 2-*complex* $\mathcal{C}$ (or two-dimensional simplicial complex) is a topological space obtained from a simple graph (without loops or multiple edges) by attaching solid, two-dimensional triangles to some of its cycles of length three; see Figure 1.[2] The *simplices* of $\mathcal{C}$ are its vertices, edges, and triangles. We can easily represent 2-complexes algorithmically, by storing the vertices, edges, and triangles, and the incidences between them. In general, many such representations correspond to the same topological space, and the choice of the representation only impacts the complexity analysis of our algorithms. The class of 2-complexes is quite general; it contains all graphs, all surfaces, and any space obtained from a surface by identifying finitely many finite subsets of points and by adding finitely many edges between any two points.

[1] Note that many authors do not consider a drawing D *simple* if D contains two parallel edges (even uncrossed), as two parallel edges necessarily intersect at their two endpoints. Likewise, uncrossed loops are usually not allowed in simple drawings. We do not make these additional restrictions here for technical reasons connected to Definition 3.1; see also Proposition 5.2.

[2] Our definition of 2-complex slightly departs from the standard one; it is the same as a geometric simplicial complex of dimension at most two, realized in some ambient space of dimension large enough.



(a) (b)

Figure 1 (a): An example of a two-dimensional simplicial complex, or 2-complex for short. (b): An embedding of some graph with four vertices and three edges on the same 2-complex.

An *isolated edge* of a 2-complex $\mathcal{C}$ is an edge incident to no triangle. The *surface part* of $\mathcal{C}$ is the union of all its triangles, together with their incident vertices and edges. A *singular point* of $\mathcal{C}$ is a point that has no open neighborhood homeomorphic to an open disk, a closed half-disk, or an open segment.

Our above definition of drawings in topological spaces applies to drawings in 2-complexes. In particular, vertices of G may lie anywhere on $\mathcal{C}$, and edges of G as curves may traverse several vertices, edges, and triangles of $\mathcal{C}$. An *embedding* of G into $\mathcal{C}$ is a drawing without any intersection point.

Embeddability of graphs on 2-complexes. The *embeddability problem* takes as input a graph G and a 2-complex $\mathcal{C}$, and the task is to decide whether G has an embedding in $\mathcal{C}$. This problem is fixed-parameter tractable in the size of the input 2-complex:

▶ **Theorem 2.1** (Colin de Verdière and Magnard [12, arXiv version, Theorem 1.1][3])**.** *One can solve the embeddability problem (of graphs in 2-complexes) in* $2^{\mathsf{poly}(C)} \cdot n^2$ *time, where* C *is the number of simplices of the input* 2*-complex and* n *is the size of the input graph.*

3 The framework: description and results

We consider *colored graphs*, in which each *edge* is labeled with a positive integer. A *colored drawing* in a surface $\mathcal{S}$ is a drawing of a colored graph G in $\mathcal{S}$ in which each edge of the drawing inherits the color of the corresponding edge of G. Throughout this paper, let $\mathcal{S}$ be a surface, and let $\mathcal{D}$ be the set of colored drawings of graphs on $\mathcal{S}$.

The general formulation of our result is based on the following definition, in which the function κ of a drawing should be thought of as a generalization of the (traditional) crossing number – "counting the crossings", or indicating (with value ∞) that "a drawing is invalid". More precisely, κ is a quality function that does not increase upon changes that add no new crossings, and λ bounds the number of pieces into which edges are split in valid drawings:

[3] The conference proceedings version of [12] gives a slightly worse bound on the running time, cubic in n. The theorems that we state take into account the improvement in the latest arXiv version.

▶ **Definition 3.1** (Crossing-counting pair). *Let $\kappa : \mathcal{D} \to \mathbb{Z}_+ \cup \{\infty\}$ be a function, and let $\lambda : \mathbb{Z}_+ \to \mathbb{Z}_+$ be a non-decreasing function. We say that (κ, λ) is a* crossing-counting pair *if:*

1. *for any $D, D' \in \mathcal{D}$, we have $\kappa(D') \leq \kappa(D)$ whenever D' results from D by one of the following operations:* (a) *the removal of a vertex or an edge;* (b) *the addition of a vertex whose image is disjoint from the image of D (and from the boundary of $\mathcal{S}$);* (c) *the addition of an edge, of an arbitrary color, connecting two vertices in D, and otherwise disjoint from the image of D (and from the boundary of $\mathcal{S}$);* (d) *a self-homeomorphism of $\mathcal{S}$;*
2. *for any drawing $D^\times \in \mathcal{D}$ such that each edge carries at least one intersection point in $D^\times$ (either a self-intersection or an intersection with another edge, but common endpoints do not count) and $\kappa(D^\times) < \infty$, the number of pieces of $D^\times$ is at most $\lambda(\kappa(D^\times))$;*
3. *one can compute $\lambda(i)$ in $2^{\mathcal{O}(i)}$ time.*

Note that Definition 3.1, Item 1, implies that κ is not affected by any of the operations (b), (c), and (d), since such operations can be reversed by an operation of type (a) or (d).

As an important special case, if $\kappa(D)$ is the number of crossings in D, or is ∞ if D is not a normal drawing, and $\lambda(i) = 4i$, then (κ, λ) satisfies the conditions of Definition 3.1 (indeed, note that each piece of an edge of $D^\times$ is incident to at least one crossing point, and each crossing point involves at most four pieces). This motivates the term "crossing-counting pair".

We now introduce the following problem (the reader may wish to focus on the simplified case of $p = q_1 = \ldots = q_c = 0$, which implies $G = G'$):

(κ, λ)-Crossing Number (for the class $\mathcal{D}$ of c-colored drawings, in surface $\mathcal{S}$)
Input: A colored graph G with colors in $\{1, \ldots, c\}$ for some integer c, with at most n vertices and edges, a non-negative integer r, and non-negative integers p and $q_1, q_2, \ldots, q_c$.
Question: Is there a colored drawing $D' \in \mathcal{D}$ in $\mathcal{S}$, with $\kappa(D') \leq r$, of a colored graph G', such that G' is obtained from G by performing at most p successive vertex splits and by removing, for $i = 1, \ldots, c$, at most q_i edges of color i?

Our main result is that, assuming κ is computable, the (κ, λ)-Crossing Number problem is fixed-parameter tractable in the topology of the surface, the number c of colors, the number $p + \sum_i q_i$ of allowed vertex splits and edge removals, and the value of r:

▶ **Theorem 3.2.** *Let $\mathcal{S}$ be a surface of topological size s, let $\mathcal{D}$ be the set of colored drawings of graphs in $\mathcal{S}$, and let (κ, λ) be a crossing-counting pair.*

Let δ be a non-decreasing function such that, given a representation of size i of a (colored) drawing $D \in \mathcal{D}$, one can compute $\kappa(D)$ in time $\delta(i)$.

Then the (κ, λ)-Crossing Number problem, on input instance $(G, r, p, q_1, \ldots, q_c)$, can be solved in time $2^{\mathsf{poly}(s+c+p+q+\lambda(r))} \cdot \delta(\mathcal{O}(s + \lambda(r))) \cdot n^2$, where n is the size of G and $q = \sum_i q_i$.

In all our applications, the factor $\delta(\cdot)$ in the runtime will be at most exponential in the parameters, and can thus be ignored. Moreover, for positive instances, we can compute an actual representation of the corresponding drawing; see the full version.

4 Sketch of proof of Theorem 3.2

Due to space restrictions, we only informally describe the reduction here; details are in the full version. Recall that $\mathcal{D}$ denotes the set of drawings of graphs on the surface $\mathcal{S}$. Let (κ, λ) be a crossing-counting pair for drawings in $\mathcal{D}$.

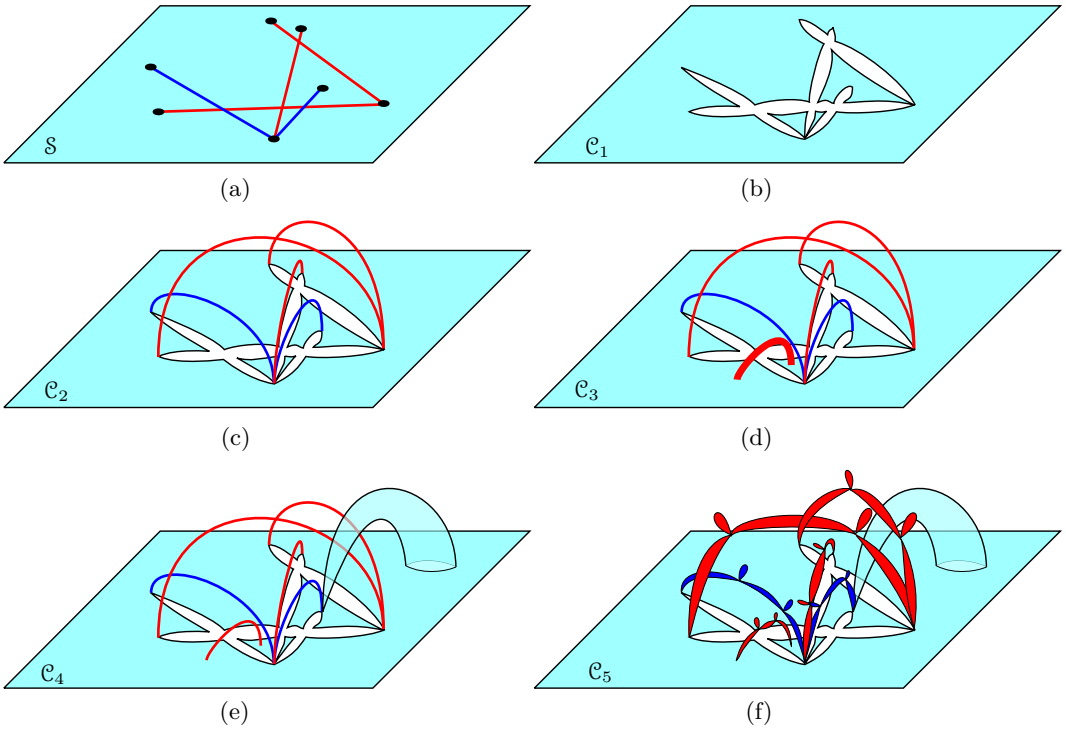


Figure 2 The construction of a 2-complex $\mathcal{C}_5$ in our reduction. (a): The surface $\mathcal{S}$ and the colored drawing $D^\times$. (b): $\mathcal{C}_1$ is obtained by cutting $\mathcal{S}$ along $D^\times$. Note that each vertex of G (in particular, the one in the bottom of the image) on $\mathcal{S}$ corresponds to a single point in $\mathcal{C}_1$, while, in this particular example, each intersection point in D of multiplicity i corresponds to $4i$ points in $\mathcal{C}_1$. (c) $\mathcal{C}_2$ is obtained by adding isolated edges corresponding to the edges of $D^\times$; we define the color of each isolated edge to be that of the corresponding edge in $D^\times$. (d) $\mathcal{C}_3$ is obtained by adding q_i isolated edges of color i for each color i. Here color 1 is represented in red, color 2 is represented in blue, and $q_1 = 1$, $q_2 = 0$. (e) $\mathcal{C}_4$ is obtained by iteratively identifying p pairs of points of $\mathcal{C}_3$ (here $p = 1$). (f) $\mathcal{C}_5$ is obtained by replacing each isolated edge of color i with a necklace (represented figuratively here) of thickness $p + 2q + 2\lambda(r) + i$ and beads of size $c - i + 2$.

The proof of Theorem 3.2 is a parameterized Turing reduction to the embeddability problem on 2-complexes. Consider an instance $(G, r, p, q_1, \ldots, q_c)$ of the (κ, λ)-Crossing Number problem. We define an uncolored graph $G_2 = G_2(G, r, p, q_1, \ldots, q_c)$ and a set of 2-complexes $\Gamma = \Gamma(r, p, q_1, \ldots, q_c)$ such that our (κ, λ)-Crossing Number instance is positive if and only if G_2 embeds in at least one of the 2-complexes in Γ (see a sketch in Figure 2). The set Γ is built by branching over a (small enough) set of properties for the hypothetical drawing of a graph G' (obtained from G by edge removals and vertex splits as in the problem definition) or, equivalently, by *guessing* some properties of that drawing.

Assume that G' has a colored drawing D' in $\mathcal{S}$ with $\kappa(D') \leq r$. Let $D^\times$ be the subdrawing of the subgraph of G' made of the edges that are involved in at least one intersection in D'; the definition of a crossing-counting pair implies that $\kappa(D^\times) \leq r$, so $D^\times$ is made of at most $\lambda(\kappa(D^\times)) \leq \lambda(r)$ pieces. Viewing $D^\times$ as an abstract drawing, without its correspondence with the vertices and edges of G', this implies that we can enumerate all such colored drawings up to self-homeomorphism of $\mathcal{S}$ in time $(s + c + \lambda(r))^{\mathcal{O}(s+\lambda(r))}$. In other words, we can *guess* the appropriate colored drawing $D^\times$.

Subsequently, we cut $\mathcal{S}$ along the relative interior of the edges of $D^\times$ (Figure 2(b); details are in the full version) and add isolated edges connecting the endpoints of the edges in $D^\times$ (Figure 2(c)), obtaining a 2-complex $\mathcal{C}_2$ in which G' embeds. If G' is allowed to result from G by edge removals and vertex splits, we modify the 2-complex appropriately, by guessing where the endpoints of the additional isolated edges must be inserted (Figure 2(d)) and which points of the 2-complex must be identified back (doing an inverse of the vertex split, Figure 2(e)). Note that these guessed points may be picked from the surface part of $\mathcal{C}_2$, as well as among the vertices of the edges of $D^\times$ in $\mathcal{C}_2$. We obtain a 2-complex $\mathcal{C}_4$ in which G embeds. Each isolated edge of $\mathcal{C}_4$ naturally bears the color of the edge of G it carries.

It remains to ensure that, conversely, the embeddability of G into one of the resulting 2-complexes implies a positive (κ, λ)-Crossing Number instance. For this purpose, we do the following. First (Figure 3(b)), as a minor technical detail, we apply a preprocessing step – attaching a 4-clique to each vertex of G, to ensure that all vertices of G have degree at least three. Let G_1 be this new graph. Second, and more importantly (Figure 2(f) and Figure 3), we need to encode the colors of the edges. We turn each edge of G_1 and each isolated edge of the 2-complex $\mathcal{C}_4$ (in which G_1 embeds, see above) into a *necklace* encoding the color, such that a necklace of a certain encoding type in the resulting graph can only use a necklace of the same type in the resulting 2-complex. This results in an (uncolored) graph G_2 embedded in a 2-complex $\mathcal{C}_5$. Finally, the set Γ of 2-complexes in which we try to embed G_2 is made of all 2-complexes $\mathcal{C}_5$, over all possible choices (guesses) described above.

In more detail, we define a *necklace of thickness h and beads of size k* to be the graph obtained from a path of length three by (1) replacing each edge with h parallel edges, and (2) attaching k loops to each of the two internal vertices of the path (Figure 3(c)). The intuition is that, by the minimum-degree condition on G_1, the internal vertices of necklaces of our 2-complex can essentially only be used by internal vertices of the necklaces of G_2. So, if edges of color i are encoded with necklaces of, say, thickness i and beads of size $c - i$, we effectively prevent graph necklaces from using 2-complex necklaces of different types. (We will actually use slightly refined formulas for the thickness and the size of the beads.)

5 Applications: FPT algorithms for diverse crossing number variants

In this section, we demonstrate the wide applicability of our framework to crossing number problems. First, as a toy example, we reprove that we can compute the traditional crossing number in quadratic FPT time on surfaces [12]:

▶ **Proposition 5.1.** *Let $r \geq 0$ be an integer, and $\mathcal{S}$ a surface with topological size s. Deciding whether an input graph G of size n has crossing number at most r in $\mathcal{S}$ can be solved in time $2^{\mathsf{poly}(s+r)} \cdot n^2$.*

Proof. We apply our framework with a single color ($c = 1$), no vertex splits or edge removals ($p = q_1 = 0$), and in the case where $\mathcal{D}$ is the set of drawings of graphs on $\mathcal{S}$. For any drawing D, let $\kappa(D)$ be the number of crossings in D, or ∞ if D is not a normal drawing. Moreover, let $\lambda(i) = 4i$. Since $\kappa(D) < \infty$ only if D is normal, (κ, λ) is a crossing-counting pair. Theorem 3.2 implies the result. ◀

We now turn to the promised applications. We survey, in order, specific drawing styles, colored crossing number problems, various counting methods, and problems where prior edge removals and vertex splits are allowed. Throughout this section, again, let $\mathcal{S}$ be a surface and $\mathcal{D}$ the set of colored drawings of graphs on $\mathcal{S}$. In our reductions to the (κ, λ)-Crossing Number problem, we always set $p = q_1 = \ldots = q_c = 0$ except in Section 5.4. In nearly all

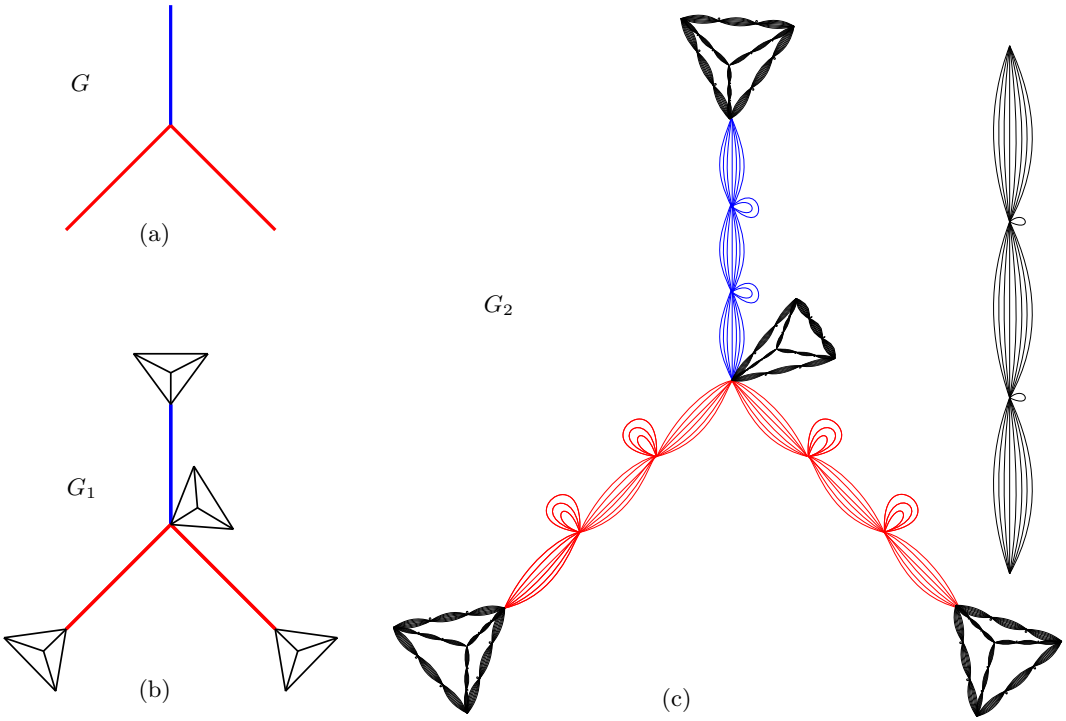


Figure 3 The construction of the graph G_2. (a): The graph G; the red edges have color 1 and the blue edge has color 2. (b): The graph G_1. The new edges, in black, have color 3. (c): The graph G_2 in the case $c = 2$, $\lambda(r) = p = q = 1$. Each edge of color i is replaced with a necklace of thickness $p + 2q + 2\lambda(r) + i = 5 + i$ and beads of size $c - i + 2 = 4 - i$. On the right, a close-up on the replacement for a black edge in G_1, which is a necklace of thickness eight and beads of size one.

cases, only normal drawings of graphs are considered. Note that in all cases, for positive instances, we can compute a corresponding drawing; see the full version. Statements marked with * are proved in the full version.

5.1 Variations on the traditional crossing number

k-planar drawings. Introduced by Pach and Tóth [40], *k-planar drawings* (for $k \geq 0$) are normal drawings in the plane such that every edge is involved in at most k crossings. For any drawing D, let $\kappa(D)$ be the number of crossings in D, if D is a normal k-planar drawing, and $\kappa(D) = \infty$ otherwise. The *k-planar crossing number* is then the smallest value of $\kappa(D)$ over all normal drawings D of G (which may equal ∞). Already for $k = 1$, deciding the existence of a 1-planar drawing is NP-hard even for very restricted inputs [8, 19]. This definition naturally extends to an arbitrary surface $\mathbb{S}$, which we call the *k-surface crossing number* in $\mathbb{S}$.

k-quasi-planar and min-k-planar drawings. There are similar concepts of *k-quasi-planar* [1] (no k edges pairwise cross) and of *min-k-planar* [6] (for every two edges with a common crossing, one carries at most k crossings) normal drawings in the plane, which give rise to the corresponding crossing number flavors in the plane, and we may likewise generalize them to the *k-quasi-surface* and *min-k-surface crossing numbers* in a surface $\mathbb{S}$.

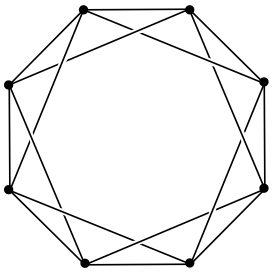


Figure 4 A drawing of some graph that is a 1-gap drawing, as certified by the casing.

Many authors restrict the problems to simple graphs and require the admissible drawings to be simple; this restriction defines the *simple-drawing k-surface* (*k-quasi-surface*, *min-k-surface*) crossing number(s). These simple-drawing variants can be very different from the non-simple ones, see, e.g., [44, Chapter 7] and [27], but we will show that they also smoothly fit into our framework.

k-gap drawings. Another known concept is that of *k-gap crossing number* [3,46], minimizing the number of crossings over *k-gap drawings* in the plane (Figure 4), namely, normal drawings D of a graph G admitting a mapping from each crossing in D to one of the two involved edges such that at most k crossings are mapped to each edge of G. This problem is NP-hard for $k = 1$ [3]. Again, the k-gap crossing number is naturally generalized to any surface $\mathcal{S}$.

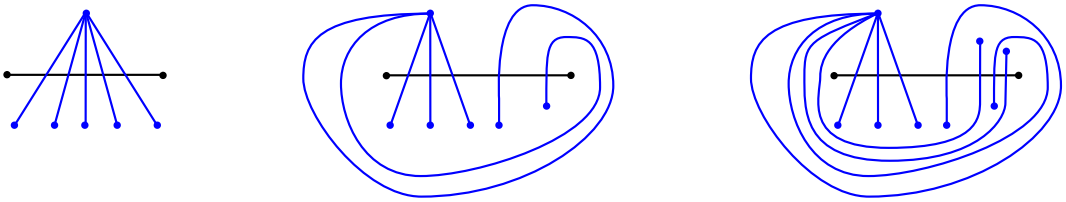
Applying our framework. Given $r \geq 0$, deciding whether a graph G has k-planar (k-quasi-planar, or min-k-planar) crossing number at most r can be done in quadratic FPT-time [37] using Courcelle's theorem. The parameterized complexity of these three problems in other surfaces, and of the k-gap crossing problem altogether, has not been studied prior to us. Even in the known planar cases, we largely improve the dependence in r in the runtime:

▶ **Proposition 5.2.** *Let $k \geq 1$ and $r \geq 0$ be integers, $\mathcal{S}$ a surface with topological size s, and let $\Pi \in \{$"k-surface", "k-quasi-surface", "min-k-surface", "k-gap"$\}$. Deciding whether an input graph G of size n has Π-crossing number at most r in $\mathcal{S}$ can be solved in time $2^{\mathsf{poly}(s+r)} \cdot n^2$. The same conclusion holds for the simple-drawing Π-crossing number of a simple input graph G.*

Proof. The proof relies on Theorem 3.2 with a single color ($c = 1$) and no splits or removals ($p = q_1 = 0$). For each Π, we must give an appropriate crossing-counting pair (κ, λ) such that κ is efficiently computable. Let $\kappa = \kappa_\Pi$ map a drawing D to ∞ if it is not a normal drawing or if it violates the property Π, and to the number of crossings of D otherwise. In particular, if k is greater than the number of crossings in D, then D always satisfies Π.

Given a drawing D, one may in time polynomial in the size of a representation of D verify that D is a normal drawing satisfying Π, and so compute $\kappa_\Pi(D)$. This is an easy routine in all cases of Π except when $\Pi =$ 'k-gap'; in the latter case we additionally employ a standard maximum flow algorithm to decide the existence of a mapping from the crossings to their incident edges respecting "capacities" of edges to accept at most k crossings each.

Since we only consider normal drawings, $\lambda(i) = 4i$ satisfies the assumption of Definition 3.1. Likewise, it is easy to see that κ_Π is non-increasing under the operations listed in Definition 3.1. Hence, our conclusion follows by an application of Theorem 3.2 to (κ_Π, λ) and the graph G.



(a) (b) (c)

Figure 5 (a) A strongly fan-planar drawing. (b) A weakly fan-planar drawing that is not strongly fan-planar. (c) A fan-crossing drawing that is not weakly fan-planar.

Next, we consider the simple-drawing Π-crossing number, assuming G is simple. We now, additionally, set $\kappa_\Pi(D) = \infty$ if D is a non-simple drawing. For this case, it is important that our definition of a simple drawing does not immediately exclude drawings of non-simple graphs; while this is irrelevant for our simple graph G, fulfillment of Definition 3.1, Item 1(c), depends on this technical twist. The rest of the argument is identical. ◀

Fan-crossing and fan-planar drawings. A normal drawing is *fan-crossing* [31] if all edges crossing the same edge are incident to a common vertex. One can restrict this notion to *strongly* or *weakly fan-planar* drawings, which not necessarily lead to the same crossing number problems, see Figure 5 and [9]. The *fan-crossing*, *strongly fan-planar*, and *weakly fan-planar crossing numbers* are the minimum number of crossings over all strongly fan-planar, weakly fan-planar, and fan-crossing drawings in the plane, respectively.

FPT algorithms, all using Courcelle's theorem, exist for the fan-crossing crossing number, by Münch and Rutter [37], and for the weakly fan-planar crossing number, by Hamm, Klute, and Parada [22], and it seems that with further adjustments they ([22]) could also handle the strongly fan-planar version. We solve all three variants, again with a better runtime. For the strongly fan-planar version, we use the fact that our framework allows for surfaces with boundary, and thus can easily capture the "infinite face" of the drawing; indeed, strong fan-planarity is invariant under self-homeomorphisms of the plane, but not of the sphere (e.g., the drawings in Figure 5(a) and (b) are homeomorphic when seen on the sphere).

▶ **Proposition 5.3.*** *Let $r \geq 0$ be an integer. The problem to decide whether an input graph G of size n has strongly fan-planar, weakly fan-planar, or fan-crossing crossing number at most r in the plane can be solved in time $2^{\mathsf{poly}(r)} \cdot n^2$.*

The proof is very similar to that of Proposition 5.2 and extends to arbitrary surfaces.

Drawings that may not be normal. To illustrate the applicability of our framework to non-normal drawings, we introduce the following example. For an integer $k \geq 2$, we define the *k-intersecting crossing number* of a graph G to be the minimum value r such that there is a drawing of G with at most r intersection points in which the multiplicity of every intersection point is at most k. For every graph G, the traditional crossing number of G is at most $\binom{k}{2}$ times its k-intersecting crossing number, and this bound can be tight. We prove:

▶ **Proposition 5.4.*** *Let $k \geq 2$ and $r \geq 0$ be integers, $\mathbb{S}$ a surface with topological size s. The problem to decide whether an input graph G of size n has k-intersecting crossing number at most r in $\mathbb{S}$ can be solved in time $2^{\mathsf{poly}(s+k+r)} \cdot n^2$.*

We refer to Section 5.3 for more examples of different methods for counting crossings.

5.2 Crossing problems with colored edges

We now turn our attention to problems in which the colors of intersecting edges play a role.

Joint crossing number and generalizations. In the *joint crossing number* problem [38], the goal is to embed two input graphs simultaneously on the same surface while minimizing the number of crossings between them. It is NP-hard for any fixed Euler genus $g \geq 6$ [25,28], and the complexity parameterized by the solution value has not been studied prior to our work. We actually solve the following new problem generalizing the joint crossing number:

Color-Constrained Crossing Problem (in surface $\mathcal{S}$)
Input: A colored graph G with colors in $\{1,\dots,c\}$, and a symmetric matrix M of size $c \times c$ with nonnegative integer values.
Question: Is there a normal colored drawing D of G in $\mathcal{S}$ such that, for each $i,j \in \{1,\dots,c\}$, the number of crossings involving two edges, one colored i and the other j, is at most $M_{i,j}$?

▶ **Proposition 5.5.*** *Let $\mathcal{S}$ be a surface of with topological size s. The Color-Constrained Crossing Problem in $\mathcal{S}$ of the input graph G of size n and matrix M can be solved in time $2^{\mathsf{poly}(s+r)} \cdot n^2$, where $r = \sum_{1 \leq i \leq j \leq c} M_{i,j}$.*

Proposition 5.5 solves the joint crossing number in $\mathcal{S}$ simply by giving the edges of one graph color red and the edges of the other color blue, and allowing only red–blue crossings.

Our approach can also be applied (see the full version) to the *homeomorphic joint crossing number* [28] of two graphs G_1 and G_2, in which G_1 and G_2 are given as cellularly embedded in $\mathcal{S}$, and the task is to minimize the number of crossings of $G_1 \cup G_2$ under an additional condition that the subdrawings of G_1 and G_2 are homeomorphic to the given embeddings.

Fixing the rotation system. More generally, a natural restriction on crossing number problems is to fix the clockwise cyclic order of edges around each vertex (the *rotation system*). The traditional crossing number with a fixed rotation system remains NP-hard [43], and its parameterized complexity has not been considered so far. We solve the variant of the Color-Constrained Crossing Problem where the input also prescribes the cyclic permutation of the edges around each vertex of G, and give a solution in the plane. (We remark that the problem naturally generalizes to orientable surfaces, but the proof is more complicated then, so in this version we only formulate it in the plane.)

▶ **Theorem 5.6.*** *The Color-Constrained Crossing Problem with Rotation System (in the plane) of the input connected graph G of size n, matrix M, and rotation system $\big(\pi_v : v \in V(G)\big)$ can be solved in time $2^{\mathsf{poly}(r)} \cdot n^2$, where $r = \sum_{1 \leq i \leq j \leq c} M_{i,j}$.*

The proof reduces to the Color-Constrained Crossing Problem. The main delicate task in it is to "transfer the sense of orientation" from one vertex to another, which is achieved with suitably crafted tri-colored rigid "chains" as sketched in Figure 6.

5.3 Non-traditional methods of counting crossings

We now look at different problems which do not count the crossings simply "one by one".

Edge crossing number. The *edge crossing number* of a graph G is the smallest r such that G has a normal drawing having at most r edges with at least one crossing. Computing the edge crossing number is NP-hard [4], and according to Schaefer [45, Section 3.2: Edge crossing number], no FPT algorithm parameterized by the solution size r is known even in the plane.

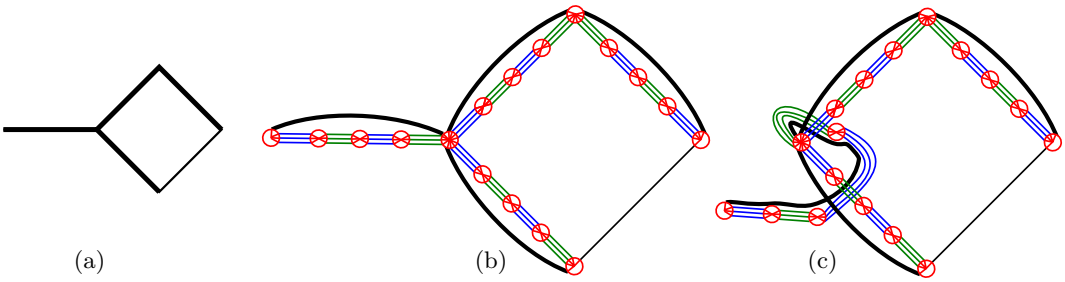


Figure 6 The construction of the graph G_1 for the proof of Theorem 5.6.(a) The graph G, with the spanning tree T in thick lines, shown with the desired rotation system. (b) The graph G_1; its original edges are black, and the new colors red, blue, and green are as pictured. Red edges cannot carry any crossing, and blue-blue and green-green crossings are forbidden. (c) A valid drawing of the graph G_1 with crossings. In general, we must allow crossings between the chains, and these are made possible with the alternating green-blue paths; forbidding blue-blue and green-green crossings forces the consistency of the rotation systems of the wheels.

▶ **Proposition 5.7.** *Let $r \geq 0$ be an integer and $\mathbb{S}$ be a surface with topological size s. The problem to decide whether an input graph G of size n has edge crossing number at most r in $\mathbb{S}$ can be solved in time $2^{\mathsf{poly}(s+r)} \cdot n^2$.*

Proof. For a drawing D in $\mathbb{S}$, let $\kappa(D)$ be equal to the number ℓ of edges of D carrying an intersection point if the total number of intersection points in D is at most $\binom{\ell}{2}$, and let $\kappa(D) = \infty$ otherwise. We also set $\lambda(i) = 4\binom{i}{2}$. It is again routine to check that (κ, λ) is a crossing-counting pair and that $\kappa(D)$ is computable in polynomial time.

It is well known, see, e.g., Schaefer [44], that any drawing D with ℓ edges carrying an intersection point can be turned into a normal drawing D in which no two edges cross twice, without introducing new crossed edges, and thus having at most $\binom{\ell}{2}$ crossings. Thus, the edge crossing number of G indeed equals the minimum value of $\kappa(D)$ over all drawings D of G. Hence, Theorem 3.2 implies an algorithm running in time $2^{\mathsf{poly}(s+r)} \cdot n^2$. ◀

A natural edge-colored generalization can be solved by a straightforward combination of the ideas in the proof of Proposition 5.5 and of the previous proof.

Odd and pair crossing numbers. The *pair (resp. odd) crossing number* of a graph G is the minimum r such that there exists a normal drawing of G in which at most r pairs of edges mutually cross (resp. cross an odd number of times). While the question whether the pair crossing number coincides with the traditional crossing number is one of the biggest open problems in crossing numbers, Pelsmajer, Schaefer, and Štefankovič [42] proved that the odd crossing number can be lower than the traditional crossing number. FPT algorithms for the pair and odd crossing numbers were given by Pelsmajer, Schaefer, and Štefankovič [41], with an unspecified dependence on r; however, because these algorithms are based on an adaptation of Grohe's algorithm [20], they are quadratic in the size n of the input and the dependence on r is at least an exponential tower of height four. We improve over their algorithms by providing algorithms also quadratic in n but with a better dependence in the parameter r:

▶ **Proposition 5.8.*** *Let $r \geq 0$ be an integer, and let $\Pi \in \{$'pair', 'odd'$\}$. The problems to decide whether an input graph G of size n has Π-crossing number at most r in the* plane *can be solved in time $2^{2^{\mathcal{O}(r)}} \cdot n^2$.*

5.4 Allowing edge removals and vertex splits before drawing

Finally, we consider crossing number variants that allow graph simplifications before drawing.

Skewness. The *skewness* of a graph G is the smallest number of edges whose removal from G leaves a planar graph. Deciding whether the skewness of G is at most q is NP-complete [34], and linear-time FPT algorithms are claimed, without details, by Kawarabayashi and Reed [32] and Jansen, Lokshtanov, and Saurabh [30]. We generalize the problem as follows.

Color-Constrained $\mathcal{S}$-Skewness with Crossings
Input: A colored graph G with colors in $\{1,\dots,c\}$, non-negative integers $q_1,\dots,q_c$ and r.
Question: Can we remove, for each $i=1,\dots,c$, at most q_i edges of color i from G, such that the resulting graph has crossing number at most r in $\mathcal{S}$?

▶ **Proposition 5.9.*** *Let $\mathcal{S}$ be a surface with topological size s. The* Color-Constrained $\mathcal{S}$-Skewness with Crossings *problem for an input graph G of size n and parameter r can be solved in time $2^{\mathsf{poly}(s+q+r)}\cdot n^2$, where $q=\sum_i q_i$.*

Splitting number. The smallest integer p such that a graph obtained from the given graph G by p successive vertex splits is embeddable in $\mathcal{S}$ is called the *splitting number* of G in $\mathcal{S}$ [23,29]. This problem is NP-hard [17], already in the plane. Nöllenburg et al. [39] proved that the $\mathcal{S}$-splitting number has a nonuniform FPT algorithm parameterized by p using the theory of graph minors. We generalize and improve the latter result to a uniform FPT algorithm:

▶ **Proposition 5.10.*** *Let $p,r\geq 0$ be integers and $\mathcal{S}$ be a surface with topological size s. The problem to decide whether an input graph G of size n has, after at most p vertex splits, a crossing number at most r in $\mathcal{S}$ can be solved in time $2^{\mathsf{poly}(s+p+r)}\cdot n^2$.*

6 Conclusions

More applications. Theorem 3.2 allows for almost endless combinations for crossing number variants, in terms of drawing styles, ways to count crossings, allowed vertex splits and edge removals, all possibly taking edge colors into account, all on an arbitrary surface.

Limitations. Nonetheless, in our definition of crossing-counting pair, $\kappa(D)$ must functionally bound the total number of intersection points in the drawing D, and the value of κ cannot grow upon deletion of vertices and edges. The first requirement immediately excludes, e.g., the local crossing number (the minimum r such that a given graph has an r-planar drawing, which is NP-complete to compute already for $r=1$ [8,19]), and the second requirement excludes some recently introduced drawing styles such as, for instance, 1^+ and 2^+-real face drawings [5]. It is conceivable that Courcelle-based approaches would be able to handle 1^+ and 2^+-real face drawings, albeit not easily.

Another weakness, in particular compared to the very recent preprint by Hamm, Klute, and Parada [22], is our quite restricted ability to handle predrawn parts of the input graph, essentially limited to fixing *uncrossable* parts of the graph via rigid subembeddings, and to special restrictions like the fixed rotation system in Theorem 5.6. In contrast, [21] and [22] are very general in this respect and, in particular, allow crossings of fixed parts with the unfixed rest of the graph.

Remarks and possible extensions. Beyond their use for strongly fan-planar drawings, surfaces with boundaries can be useful for other problems, because they allow to pinpoint specific regions of the surface. One could actually consider a version with colored boundaries, restricting Item 1(d) of the definition of a crossing-counting pair to color-preserving self-homeomorphisms; all our arguments carry through. Finally, a possible extension of the (κ, λ)-Crossing Number problem would be to replace the surface $\mathcal{S}$ with an arbitrary 2-complex. We leave open whether such an extension is possible, and also potential applications.

References

1 Pankaj K. Agarwal, Boris Aronov, János Pach, Richard Pollack, and Micha Sharir. Quasi-planar graphs have a linear number of edges. *Comb.*, 17(1):1–9, 1997. doi:10.1007/BF01196127.

2 Mark Anthony Armstrong. *Basic topology.* Undergraduate Texts in Mathematics. Springer-Verlag, 1983.

3 Sang Won Bae, Jean-François Baffier, Jinhee Chun, Peter Eades, Kord Eickmeyer, Luca Grilli, Seok-Hee Hong, Matias Korman, Fabrizio Montecchiani, Ignaz Rutter, and Csaba D. Tóth. Gap-planar graphs. *Theor. Comput. Sci.*, 745:36–52, 2018. doi:10.1016/j.tcs.2018.05.029.

4 Martin Balko, Petr Hliněný, Tomás Masařík, Joachim Orthaber, Birgit Vogtenhuber, and Mirko H. Wagner. On the uncrossed number of graphs. In Stefan Felsner and Karsten Klein, editors, *32nd International Symposium on Graph Drawing and Network Visualization, GD 2024, September 18-20, 2024, Vienna, Austria*, volume 320 of *LIPIcs*, pages 18:1–18:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.GD.2024.18.

5 Carla Binucci, Giuseppe Di Battista, Walter Didimo, Vida Dujmovic, Seok-Hee Hong, Michael Kaufmann, Giuseppe Liotta, Pat Morin, and Alessandra Tappini. Graphs drawn with some vertices per face: Density and relationships. *IEEE Access*, 12:68828–68846, 2024. doi:10.1109/ACCESS.2024.3401078.

6 Carla Binucci, Aaron Büngener, Giuseppe Di Battista, Walter Didimo, Vida Dujmovic, Seok-Hee Hong, Michael Kaufmann, Giuseppe Liotta, Pat Morin, and Alessandra Tappini. Min-k-planar drawings of graphs. In *GD (1)*, volume 14465 of *Lecture Notes in Computer Science*, pages 39–52. Springer, 2023. doi:10.1007/978-3-031-49272-3_3.

7 Sergio Cabello. Hardness of approximation for crossing number. *Discrete Comput. Geom.*, 49(2):348–358, 2013. doi:10.1007/s00454-012-9440-6.

8 Sergio Cabello and Bojan Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM J. Comput.*, 42(5):1803–1829, 2013. doi:10.1137/120872310.

9 Otfried Cheong, Henry Förster, Julia Katheder, Maximilian Pfister, and Lena Schlipf. Weakly and strongly fan-planar graphs. In Michael A. Bekos and Markus Chimani, editors, *Graph Drawing and Network Visualization - 31st International Symposium, GD 2023, Isola delle Femmine, Palermo, Italy, September 20-22, 2023, Revised Selected Papers, Part I*, volume 14465 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2023. doi:10.1007/978-3-031-49272-3_4.

10 Julia Chuzhoy and Zihan Tan. A subpolynomial approximation algorithm for graph crossing number in low-degree graphs. In *STOC*, pages 303–316. ACM, 2022. doi:10.1145/3519935.3519984.

11 Éric Colin de Verdière and Arnaud de Mesmay. Testing graph isotopy on surfaces. *Discrete & Computational Geometry*, 51(1):171–206, 2014. doi:10.1007/s00454-013-9555-4.

12 Éric Colin de Verdière and Thomas Magnard. An FPT algorithm for the embeddability of graphs into two-dimensional simplicial complexes. In *Proceedings of the 29th European Symposium on Algorithms (ESA)*, pages 32:1–32:17, 2021. Full improved version in arXiv:2107.06236v2. doi:10.4230/LIPIcs.ESA.2021.32.

13 Éric Colin de Verdière, Thomas Magnard, and Bojan Mohar. Embedding graphs into two-dimensional simplicial complexes. *Computing in Geometry and Topology*, 1(1):Article 6, 2022.

14 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

15 Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. A survey on graph drawing beyond planarity. *ACM Comput. Surv.*, 52(1):4:1–4:37, 2019. doi:10.1145/3301281.

16 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 599–608, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644208.

17 Luérbio Faria, Celina M. H. de Figueiredo, and Candido Ferreira Xavier de Mendonça Neto. Splitting Number is NP-complete. *Discret. Appl. Math.*, 108(1-2):65–83, 2001.

18 Michael R. Garey and David S. Johnson. Crossing number is NP-complete. *SIAM J. Algebr. Discrete Methods*, 4(3):312–316, September 1983.

19 Alexander Grigoriev and Hans L. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007. doi:10.1007/S00453-007-0010-X.

20 Martin Grohe. Computing crossing numbers in quadratic time. *J. Comput. Syst. Sci.*, 68(2):285–302, 2004. doi:10.1016/j.jcss.2003.07.008.

21 Thekla Hamm and Petr Hliněný. Parameterised partially-predrawn crossing number. In Xavier Goaoc and Michael Kerber, editors, *38th International Symposium on Computational Geometry, SoCG 2022, June 7-10, 2022, Berlin, Germany*, volume 224 of *LIPIcs*, pages 46:1–46:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SOCG.2022.46.

22 Thekla Hamm, Fabian Klute, and Irene Parada. Computing crossing numbers with topological and geometric restrictions. *CoRR*, abs/2412.13092, 2024. doi:10.48550/arXiv.2412.13092.

23 Nora Hartsfield, Brad Jackson, and Gerhard Ringel. The splitting number of the complete graph. *Graphs and Combinatorics*, 1:311–329, 1985. doi:10.1007/BF02582960.

24 Petr Hliněný. Crossing number is hard for cubic graphs. *Journal of Comb. Theory, Ser. B*, 96(4):455–471, 2006. doi:10.1016/J.JCTB.2005.09.009.

25 Petr Hliněný. Complexity of anchored crossing number and crossing number of almost planar graphs. In Pawel Gawrychowski, Filip Mazowiecki, and Michal Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science, MFCS 2025, August 25-29, 2025, Warsaw, Poland*, volume 345 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.MFCS.2025.59.

26 Petr Hliněný and Liana Khazaliya. Crossing number is NP-hard for constant path-width (and tree-width). In Julián Mestre and Anthony Wirth, editors, *35th International Symposium on Algorithms and Computation, ISAAC 2024, December 8-11, 2024, Sydney, Australia*, volume 322 of *LIPIcs*, pages 40:1–40:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ISAAC.2024.40.

27 Petr Hliněný and Csenge Lili Ködmön. Note on min-k-planar drawings of graphs. In Stefan Felsner and Karsten Klein, editors, *32nd International Symposium on Graph Drawing and Network Visualization, GD 2024, September 18-20, 2024, Vienna, Austria*, volume 320 of *LIPIcs*, pages 8:1–8:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.GD.2024.8.

28 Petr Hliněný and Gelasio Salazar. On hardness of the joint crossing number. In *ISAAC*, volume 9472 of *Lecture Notes in Computer Science*, pages 603–613. Springer, 2015.

29 Brad Jackson and Gerhard Ringel. Splittings of graphs on surfaces. *Proceedings of the 1st Colorado Symposium on Graph Theory*, pages 203–219, 1982.

30 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1802–1811, 2014. doi:10.1137/1.9781611973402.130.

31 Michael Kaufmann and Torsten Ueckerdt. The density of fan-planar graphs. *Electron. J. Comb.*, 29(1), 2022. doi:10.37236/10521.

32 Ken-ichi Kawarabayashi and Bruce A. Reed. Computing crossing number in linear time. In *STOC*, pages 382–390. ACM, 2007. doi:10.1145/1250790.1250848.

33 Sóstenes Lins. Graph-encoded maps. *Journal of Combinatorial Theory, Series B*, 32:171–181, 1982. doi:10.1016/0095-8956(82)90033-8.

34 P.C. Liu and R.C. Geldmacher. On the deletion of nonplanar edges of a graph. In *Proceedings of the Tenth Southeastern Conference on Combinatorics, Graph Theory and Computing (Florida Atlantic Univ.,Boca Raton, Fla., 1979)*, pages 727–738, 1979.

35 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Roohani Sharma, Jie Xue, and Meirav Zehavi. Crossing number in slightly superexponential time. In *Proceedings of the 36th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1412–1424, 2025. doi:10.1137/1.9781611978322.44.

36 Bojan Mohar and Carsten Thomassen. *Graphs on surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2001.

37 Miriam Münch and Ignaz Rutter. Parameterized algorithms for beyond-planar crossing numbers. In Stefan Felsner and Karsten Klein, editors, *32nd International Symposium on Graph Drawing and Network Visualization, GD 2024, September 18-20, 2024, Vienna, Austria*, volume 320 of *LIPIcs*, pages 25:1–25:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.GD.2024.25.

38 Seiya Negami. Crossing numbers of graph embedding pairs on closed surfaces. *J. Graph Theory*, 36(1):8–23, 2001. doi:10.1002/1097-0118(200101)36:1\%3C8::AID-JGT2\%3E3.0.CO;2-O.

39 Martin Nöllenburg, Manuel Sorge, Soeren Terziadis, Anaïs Villedieu, Hsiang-Yun Wu, and Jules Wulms. Planarizing graphs and their drawings by vertex splitting. In *GD*, volume 13764 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2022. doi:10.1007/978-3-031-22203-0_17.

40 János Pach and Géza Tóth. Graphs drawn with few crossings per edge. *Comb.*, 17(3):427–439, 1997. doi:10.1007/BF01215922.

41 Michael J. Pelsmajer, Marcus Schaefer, and Daniel Štefankovič. Crossing numbers and parameterized complexity. In *Graph Drawing, 15th International Symposium, GD 2007, Sydney, Australia, September 24-26, 2007. Revised Papers*, volume 4875 of *Lecture Notes in Computer Science*, pages 31–36. Springer, 2007. doi:10.1007/978-3-540-77537-9_6.

42 Michael J. Pelsmajer, Marcus Schaefer, and Daniel Štefankovič. Odd crossing number and crossing number are not the same. *Discret. Comput. Geom.*, 39(1-3):442–454, 2008. doi:10.1007/S00454-008-9058-X.

43 Michael J. Pelsmajer, Marcus Schaefer, and Daniel Štefankovič. Crossing numbers of graphs with rotation systems. *Algorithmica*, 60(3):679–702, 2011. doi:10.1007/S00453-009-9343-Y.

44 Marcus Schaefer. *Crossing Numbers of Graphs*. Discrete mathematics and its applications. CRC Press, Taylor & Francis Group, 2017.

45 Marcus Schaefer. The graph crossing number and its variants: A survey. *Electron. J. Comb.*, Dynamic Surveys, #DS21, 2024. Eighth Edition: May 17, 2024. doi:10.37236/2713.

46 Nathan van Beusekom, Irene Parada, and Bettina Speckmann. Crossing numbers of beyond-planar graphs revisited. *J. Graph Algorithms Appl.*, 26(1):149–170, 2022. doi:10.7155/jgaa.00586.

Constructing Long Paths in Graph Streams

Christian Konrad
School of Computer Science, University of Bristol, UK

Chhaya Trehan
Unaffiliated Researcher, Durham, UK

Abstract

In the *graph stream model of computation*, an algorithm processes the edges of an n-vertex input graph in one or more sequential passes while using a memory that is sublinear in the input size. The streaming model poses significant challenges for algorithmically constructing long paths. Many known algorithms that are tasked with extending an existing path as a subroutine require an entire pass over the input to add a single additional edge. This raises a fundamental question: Are multiple passes inherently necessary to construct paths of non-trivial lengths, or can a single pass suffice? To address this question, we systematically study the Longest Path problem in the one-pass streaming model. In this problem, given a desired approximation factor α, the objective is to compute a path of length at least $\mathrm{lp}(G)/\alpha$, where $\mathrm{lp}(G)$ is the length of a longest path in the input graph G.

We study the problem in the insertion-only and the insertion-deletion streaming models, and we give algorithms as well as space lower bounds for both undirected and directed graphs. Our results are:

1. We show that for undirected graphs, in both the insertion-only and the insertion-deletion models, there are *semi-streaming algorithms*, i.e., algorithms that use space $O(n \operatorname{poly} \log n)$, that compute a path of length at least $d/3$ with high probability, where d is the average degree of the input graph. These algorithms can also yield an α-approximation to Longest Path using space $\tilde{O}(n^2/\alpha)$.
2. Next, we show that such a result cannot be achieved for directed graphs, even in the insertion-only model. We show that computing a $(n^{1-o(1)})$-approximation to Longest Path in directed graphs in the insertion-only model requires space $\Omega(n^2)$. This result is in line with recent results that demonstrate that processing directed graphs is often significantly harder than undirected graphs in the streaming model.
3. We further complement our results with two additional lower bounds. First, we show that semi-streaming space is insufficient for small constant factor approximations to Longest Path for undirected graphs in the insertion-only model. Last, in undirected graphs in the insertion-deletion model, we show that computing an α-approximation requires space $\Omega(n^2/\alpha^3)$.

2012 ACM Subject Classification Theory of computation → Streaming models; Mathematics of computing → Paths and connectivity problems

Keywords and phrases Longest Path Problem, Streaming Algorithms, One-way Two-party Communication Complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.22

Related Version *Full Version*: https://arxiv.org/abs/2508.16022

Funding *Christian Konrad*: Supported by EPSRC New Investigator Award EP/V010611/1.
Chhaya Trehan: Most of the work was done while C.T. was at the University of Bristol where she was supported by EPSRC New Investigator Award EP/V010611/1.

1 Introduction

In the *graph stream model of computation*, an algorithm processes a stream of edge insertions and deletions that make up an n-vertex input graph $G = (V, E)$ via one or multiple sequential passes. The primary objective is to design algorithms that use as little space as possible. The most studied and best-understood setting is the one-pass insertion-only setting, where


© Christian Konrad and Chhaya Trehan;
licensed under Creative Commons License CC-BY 4.0

only a single pass is allowed and the input stream does not contain any edge deletions. It is known that many fundamental problems can be solved well in this setting, e.g., there are *semi-streaming algorithms*, i.e., algorithms that use space $\tilde{O}(n) = O(n \operatorname{poly} \log n)$[1], for computing a spanning tree and a maximal matching [16], while no $o(n^2)$ space algorithms exist for other problems such as computing a maximal independent set or a BFS/DFS tree [14, 6]. For such problems, algorithms that make multiple passes over the input data are typically considered [24, 13, 1, 9, 4].

Our work is motivated by the observation that streaming algorithms that construct long paths, often as a subroutine, require a large number of passes. For example, many streaming algorithms for computing large matchings construct augmenting paths as a subroutine (e.g., [29, 28, 30]), and these algorithms typically only add a single edge or a few edges per pass to a not-yet-completed augmenting path. Another example is streaming algorithms for computing BFS/DFS trees that extend partial trees/paths by adding only a single edge per pass [24, 13]. This raises a fundamental question: Is the one/few-edges-per-pass strategy best possible and multiple passes are inherently necessary to construct paths of non-trivial lengths, or can a single pass suffice?

The Longest Path Problem. We address this question by systematically studying the Longest Path (LP) problem in the one-pass streaming setting. In this problem, the objective is to compute a path of length $\operatorname{lp}(G)/\alpha$, where $\operatorname{lp}(G)$ denotes the length of a longest path in the input graph G, and $\alpha \geq 1$ is the approximation factor.

In the offline setting, in undirected graphs, it is known that it is NP-hard to compute a constant factor approximation to Longest Path [23] (see the same paper for stronger impossibility results that are based on other hardness conjectures). In directed graphs, a much stronger hardness result is known. Björklund, Husfeldt and Khanna [11] showed that it is NP-hard to approximate LP in directed graphs within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$. Regarding upper bounds, it is known that a path of length $d_{\min}$ can be constructed greedily in polynomial time, where $d_{\min}$ is the min-degree of the input graph [23]. There are also FPT algorithms with runtime polynomial in n but exponential in the length of the path constructed (see, for example, [10] and the references therein). We note that streaming algorithms for the LP problem have also previously received attention, however, only from a practical perspective. Kliemann et al. [25] studied practical multi-pass algorithms without providing any theoretical guarantees.

1.1 Our Results

In this work, we give one-pass streaming algorithms and space lower bounds for undirected graphs as well as a strong space lower bound for directed graphs. We consider both the insertion-only model (no deletions) and the insertion-deletion model (deletions allowed).

As our first result, we show that, in undirected graphs, if we sample $O(n \log n)$ random edges from the input graph, then this sample contains a path of length $\Omega(d)$ with high probability, where d is the average degree of the input graph. Since this sampling task can be implemented in both the insertion-only and the insertion-deletion streaming models using standard techniques, we obtain the following theorem:

[1] We write $\tilde{O}(.)$ to mean the usual Big-O notation with poly-logarithmic dependencies suppressed. $\tilde{\Theta}(.)$ and $\tilde{\Omega}(.)$ are defined similarly.

▶ **Theorem 1.** *In both the insertion-only and the insertion-deletion models, for undirected graphs, there are $O(n \operatorname{poly} \log n)$ space algorithms that compute a path of length at least $d/3$ with high probability, where d is the average degree of the input graph.*

We remark that our algorithm can be used to obtain an α-approximation to LP using $\tilde{O}(n^2/\alpha)$ space, for any $\alpha \geq 1$. This is achieved by running our algorithm in parallel with the trivial algorithm that stores $\tilde{\Theta}(n^2/\alpha)$ edges. Then, if the space constraint of $\tilde{\Theta}(n^2/\alpha)$ is large enough to store the entire graph then we can find an exact solution in exponential time. If not, then we are guaranteed that the average degree of the input graph is $\Omega(n/\alpha)$, which implies that the algorithm of Theorem 1 yields the desired result. We thus obtain the following corollary:

▶ **Corollary 2.** *In both insertion-only and insertion-deletion models, for undirected graphs, there are $\tilde{O}(n^2/\alpha)$-space streaming algorithms for an α-approximation to Longest Path.*

Next, we ask whether a similar algorithmic result is possible in directed graphs. As our main and most technical result, we show that this is not the case in a strong sense:

▶ **Theorem 3.** *Every one-pass streaming algorithm for Longest Path in the insertion-only model in directed graphs with approximation factor $n^{1-o(1)}$ requires space $\Omega(n^2)$.*

Theorem 3 together with Corollary 2 establishes a separation in the space complexity between algorithms for undirected and directed graphs for LP in the insertion-only model. This lower bound is also in line with recent results for directed graphs that demonstrate that problems on general directed graphs are often hard to solve in the streaming model [12].

Finally, we complement our results with two additional lower bounds. First, we give a lower bound for insertion-only streams and undirected graphs, ruling out the existence of semi-streaming algorithms with constant approximation factor close to 1.

▶ **Theorem 4.** *Every one-pass insertion-only streaming algorithm for Longest Path on undirected graphs with approximation factor $1 + \frac{1}{25} - \gamma$, for any $\gamma > 0$, requires space $n^{1+\Omega(\frac{1}{\log \log n})}$.*

We note that this lower bound result is significantly weaker than our lower bound for directed graphs, both in terms of approximation factor and space. This, however, is in line with the status of the problem in the offline setting, where a very strong impossibility result for directed graphs is known, but only significantly weaker impossibility results for undirected graphs exist. While the offline setting is orthogonal to the streaming setting, one cannot help but wonder whether similar mechanisms are at work that prevent us from obtaining stronger NP-hardness results for LP approximation in undirected graphs and from obtaining stronger space lower bounds in the streaming setting.

Last, we show that, in insertion-deletion streams, space $\Omega(n^2/\alpha^3)$ is required for an α-approximation of LP.

▶ **Theorem 5.** *Every one-pass insertion-deletion streaming algorithm for Longest Path on undirected graphs with approximation factor $\alpha \geq 1$ requires space $\Omega(n^2/\alpha^3)$.*

This lower bound together with our algorithm show that the optimal dependency of the space complexity on α in insertion-deletion streams is between $1/\alpha$ and $1/\alpha^3$.

1.2 Our Techniques

Algorithm. We will first explain the key ideas behind our algorithm. As observed by Karger et al. [23], a path of length $d_{\min}$, where $d_{\min}$ is the minimum degree of the input graph G, can be computed as follows: Start at any vertex v_0 and visit an arbitrary neighbor that we denote by v_1. In a general step i, we have already constructed the path $v_0, \ldots, v_i$. We then visit a neighbor v_{i+1} of v_i that has not previously been visited. Then, as long as $i < d_{\min}$, we can always find a yet unvisited neighbor, and, hence, we obtain a path of length $d_{\min}$. We first see that this argument can also be applied to the average degree d of the graph G. It is well known that, by repeatedly removing vertices of degree at most $d/2$ from G until no such vertex remains, we are left with a non-empty graph G' with min-degree at least $d/2$. We can now apply the same argument as above to G' and thus find a path of length at least $d/2$.

The approach outlined above, however, does not yield a small space streaming algorithm since a) we do not know which vertices are contained in G', and b) we cannot afford to store $\Theta(d)$ incident edges on each vertex. To overcome these obstacles, we resort to randomization. Our algorithm solely samples $O(n \log n)$ random edges F from the input graph and outputs a longest path among the edges F. To see that a long path in F exists, we argue that the subset of edges $F' \subseteq F$ that are also contained in G' contains a path of length at least $d/3$ with high probability. Such a path can be constructed greedily. Suppose we have already constructed a partial path of length $\ell < d/3$ solely using the edges F', and let $v_{\ell+1} \in V(G')$ denote its current endpoint. Then, since $v_{\ell+1}$ has a degree of at least $d/2$ in G', there are at least $d/2 - d/3 = d/6$ neighbors of $v_{\ell+1}$ in G' that have not yet been visited in the path. Since we sample $\Theta(n \log n)$ edges overall, the probability that any one of these edges incident to $v_{\ell+1}$ that connect to these $d/6$ vertices is sampled is $\Omega(\frac{n \cdot \log n}{n \cdot d}) = \Omega(\frac{\log n}{d})$, which implies that at least one of these edges is sampled with high probability and we can extend the path.

Space Lower Bounds. All our space lower bounds are proved in the one-way two-party communication setting. In this setting, two parties that we denote by Alice and Bob each hold a subset of the edges of the input graph $G = (V, E = E_A \cup E_B)$, with E_A being Alice's edges, and E_B being Bob's edges. Alice sends a single messages Π to Bob, and Bob computes the output of the protocol. Then, it is well-known that a lower bound on the size of the message Π also constitutes a lower bound on the space of any one-pass streaming algorithm.

We work with *induced matchings* in all our lower bound constructions. In a graph $G = (V, E)$, a matching $M \subseteq E$ is *induced* if the edges of the vertex-induced subgraph $G[V(M)]$ are precisely the edges of M. Suppose now that G is bipartite, and we denote it by $G = (A, B, E = E_A \cup E_B)$. Furthermore, we suppose that Alice's subgraph, i.e., the graph spanned by the edges E_A, contains a matching M that is induced in the final graph. We say that M is the *special* matching. We complete our lower bound constructions so that:

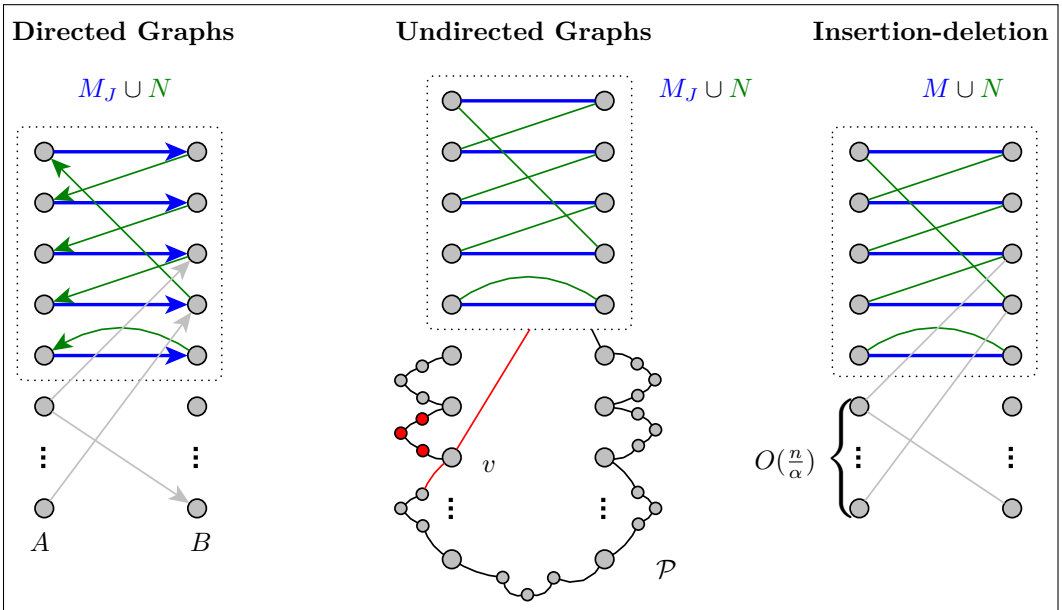
1. Every long path in the graph must contain many edges of the special matching M;
2. The special matching M is *hidden* among the edge set E_A so that Alice cannot identify M, and, given a limited communication budget, Alice therefore cannot forward many of M's edges to Bob.

The two properties then imply a lower bound since, if Bob knows only few edges of M, but every long path contains many such edges, then Bob cannot output a long path.

To achieve property 2, in our insertion-deletion lower bound, we make use of edge deletions as part of Bob's input to turn a large matching in E_A into an induced matching, and in our insertion-only lower bounds, we work with *Rusza-Szemerédi graphs* (RS-graphs in short), which have been extensively used for proving lower bounds for matching problems in the streaming setting (e.g., [18, 8, 26, 27]). An (r, t)-RS-graph is a balanced bipartite graph on $2n$

vertices such that its edge set can be partitioned into t induced matchings $M_1, \ldots, M_t$, each of size r. Our insertion-only constructions are such that each of these t matchings can take the role of the special matching M, which will allow us to argue that Alice essentially has to send many edges of each induced matching to Bob if Bob is to report a long path. We note that the framework of working with RS-graphs and a special hidden matching among Alice's edges is well-established and has previously been used, for example, in [18, 7, 3, 27, 5, 4].

Regarding establishing property 1, we pursue different strategies that depend on the specific streaming setting, and on whether we work with directed or undirected graphs. These strategies are described below and illustrated in Figure 1.



Directed Graphs: Since N constitutes the only edges directed from right to left, and M_J is an induced matching, up to potentially the first and last edge, every long path alternates between edges of M_J and N and thus contains many edges of M_J.

Undirected Graphs: (Insertion-Only) The path $\mathcal{P}$ is connected to a longest path in $V(M_J)$. Our argument is based on the fact that it is detrimental to the length of the output path to include many edges between $V \setminus V(M_J)$ and $V(M_J)$. Indeed, suppose that the output path contains the two red edges incident on vertex v. Then, the vertices highlighted in red cannot be visited by the output path. This allows us to argue that, for each edge going across the cut $(V \setminus V(M_J), V(M_J))$, multiple vertices on the path $\mathcal{P}$ are not visited. This in turn implies that every long path must contain many edges of M_J in order to visit the vertices in $V(M_J)$.

Insertion-Deletion Streams: (Undirected Graphs) Our construction is such that there are only $O(\frac{n}{\alpha})$ vertices outside $V(M)$. Hence, if Bob does not know many edges of M then there are at most $O(\frac{n}{\alpha})$ vertices of $V \setminus V(M)$ available to connect to vertices within $V(M)$. Since this number is small, Bob is required to learn many edges of M in order to output a long path.

Figure 1 Illustrations of our three lower bound constructions.

Directed Graphs in the Insertion-only Model. The cleanest case is our lower bound for directed graphs. In this setting, Alice holds the edges of an RS-graph $G' = (A, B, E)$, and we assume that all edges are directed from A to B. We then pick a matching M_J, for a uniform random $J \in [t]$, and Bob inserts a random matching N that matches $B(M_J)$ to $A(M_J)$ so that all edges in N are directed from B to A. We then leverage the well-known result that the expected length of a longest cycle in a random permutation of the set $[r]$ is of length $\Theta(r)$ [20, 19] in order to argue that $M_J \cup N$ contains a path of expected length $\Theta(|M_J|)$. It then remains to argue that every path of length ℓ, for any ℓ, in the input graph $G = G' \cup N$ must contain $\Omega(\ell)$ edges of M_J, which uses the fact that the matching M_J is induced. Since, however, Alice did not know that M_J is the special matching, Alice is required to send a large number of edges of each induced matching to Bob so that Bob can output a long path.

For technical reasons, our graph construction is slightly more involved than described above since we need to turn Alice's RS-graph into an input distribution, see Section 4 for details. While it is easy to argue that if Alice sends k edges of M_J to Bob then Bob can compute a path of length at most $O(k)$ in $M_J \cup N$, we need to argue a much stronger bound. We show that, even if Alice sends as many as $r/100$ M_J edges to Bob then Bob can still only find a path of size $O(\log r)$ in $M_J \cup N$. We implement our approach using the information complexity paradigm, including direct sum and message compression arguments.

Undirected Graphs in the Insertion-only Model. In our construction for undirected graphs, Alice holds an RS-graph $G' = (A, B, E)$, and, given a random index $J \in [t]$, Bob holds a random matching N matching $A(M_J)$ to $B(M_J)$. Similar as above, the matchings $M_J \cup N$ contain a path of length $\Omega(|M_J|)$. However, it is no longer true that any long path in $G' \cup N$ must contain many edges of M_J as, for example, the vertices in $V \setminus V(M_J)$ can now be used to visit the vertices within $V(M_J)$ as their incident edges are undirected.

Our aim is to ensure that it is detrimental for constructing long paths to include many edges across the cut $(V \setminus V(M_J), V(M_J))$ in the path. Once this property is established, we will argue that, if Bob only knows few of the M_J edges then many vertices of $V(M_J)$ will remain unvisited in the output path, which implies that the path is bounded in length as not all vertices are visited. To argue that only few edges across the cut $(V \setminus V(M_J), V(M_J))$ are included in the output path, we ensure that all the vertices $v \in V \setminus V(M_J)$ serve as *gateways* to other parts of the graph that are introduced by Bob. The construction is so that these other parts cannot be visited if v connects to a vertex in $V(M_J)$ in the output path, which will then be detrimental for the construction of a long path. To achieve this, Bob introduces additional edges as follows. First, let $\mathcal{P}'$ be a path consisting of novel edges that visits every vertex in $V \setminus V(M_J)$, and let $\mathcal{P}$ be the path obtained from $\mathcal{P}'$ by subdividing every edge in $\mathcal{P}'$, ℓ times, for some integer ℓ. Furthermore, this path is connected to the longest path within $M_J \cup N$. Then, the path $\mathcal{P}$ significantly contributes to the longest path in the input graph. Observe, however, if a vertex $v \in V \setminus V(M_J)$ connects to a vertex in $V(M_J)$ then the vertices on the subdivision on one of the edges of $\mathcal{P}$ incident on v cannot be visited anymore. By setting ℓ large enough, we show that it is detrimental for the algorithm to use such vertices to connect to $V(M_J)$, which renders the edges of M_J indispensable for a long path. As above, the actual lower bound construction is more involved since we need to turn Alice's RS-graph into an input distribution. On a technical level, we give a reduction to the two-party communication problem Index using ideas similar to those introduced by Dark and Konrad [15].

Undirected Graphs in the Insertion-deletion Model. The key advantage that allows us to prove a much stronger lower bound for insertion-deletion streams than for insertion-only streams is that Bob can insert edge deletions that turn a large matching contained in Alice's edges E_A into an induced matching. We see that it is possible to achieve this so that:

1. The special (induced) matching M is of size $n - O(n/\alpha)$; and
2. The special matching M is hidden among Alice's edges.

Furthermore, we make sure that there exists a second matching N such that $N \cup M$ forms a path of length $\Omega(n)$. Property 1 significantly helps in proving our lower bound since, as opposed to our lower bound for undirected graphs in insertion-only streams, there are only $O(n/\alpha)$ vertices outside the set $V(M)$, and, hence, these vertices only allow us to visit $O(n/\alpha)$ vertices of $V(M)$. We can then argue that, for the remaining $n - O(n/\alpha)$ vertices of $V(M)$, Bob is required to know the edges of M for those to be visited. This however requires Alice to send these edges to Bob. Then, Property 2 ensures that Alice cannot identify these edges and would therefore have to send most edges of the input graph to Bob, which exceeds the allowed communication budget. On a technical level, we give a reduction to the Augmented-Index two-party communication problem, again, reusing many of the ideas given in the lower bound by Dark and Konrad [15].

1.3 Outline

We provide notation, state RS-graph constructions, introduce the information complexity framework, state important inequalities involving mutual information, and give a message compression theorem in our preliminaries section, Section 2. Then, we present our algorithm in Section 3, and our lower bound for directed graphs in insertion-only streams is given in Section 4. Due to space restrictions, our lower bound for undirected graphs in insertion-only streams and our lower bound for undirected graphs in insertion-deletion streams are deferred to the full version of this paper. We conclude in Section 5 with open problems.

2 Preliminaries

Given a graph $G = (V, E)$, for vertices $u, v \in V$, we denote an undirected edge between u and v by $\{u, v\}$, and a directed edge with tail u and head v by (u, v).

2.1 Rusza-Szemerédi Graphs

In our lower bounds, we make use of Rusza-Szemerédi graphs.

▶ **Definition 6** (Rusza-Szemerédi Graph). *A bipartite graph $G = (A, B, E)$ is an (r, t)-*Ruzsa-Szemerédi graph*, (r, t)-RS graph for short, if its edge set can be partitioned into t induced matchings, each of size r.*

We will use the RS-graph constructions of Alon et al. [2] and Goel et al. [18], the latter is based on the construction by Fischer et al. [17].

▶ **Theorem 7** (Ruzsa-Szemerédi Graph Constructions). *There are bipartite RS-graphs $G = (A, B, E)$ with $|A| = |B| = n$ with the following parameters:*

1. *$r = n^{1-o(1)}$, and $t \cdot r = \Omega(n^2)$ [2]; and*
2. *$r = (\frac{1}{2} - \epsilon)n$, for any constant $\epsilon > 0$, and $t = n^{\Omega(\frac{1}{\log \log n})}$ [18].*

2.2 Streaming Models

Given an input graph $G = (V, E)$, an *insertion-only* stream describing G is an arbitrarily ordered sequence of the edge set E. An *insertion-deletion* stream is a sequence of edge insertions and deletions so that, at the end of the stream, the surviving edges constitute the edge set E. Furthermore, the stream is such that only edges that have previously been

inserted are deleted, i.e., the multiplicity of an edge is never negative. We also assume that at any moment the multiplicity of any edge is polynomially bounded. This standard assumption ensures that sampling methods such as ℓ_0-sampling require only $\operatorname{poly}\log(n)$ space.

2.3 Communication Complexity

We consider the one-way two-party model of communication for proving our space lower bounds. In this setting, two parties, denoted Alice and Bob, share the input data $X = (X_A, X_B)$ so that Alice holds X_A and Bob holds X_B. They operate as specified in a protocol Π in order to solve a problem $\mathcal{P}$. Alice and Bob can make use of both private and public randomness. Randomness is provided via infinite sequences of uniform random bits. Private randomness can only be accessed by one party. The sequence of public randomness can be accessed by both Alice and Bob, and we denote this sequence by R.

The protocol Π instructs the parties to operate as follows. First, Alice computes a message that we (ambiguously) also denote by Π as a function of X_A, R, and her private randomness, sends the message Π to Bob, who then computes the output of the protocol as a function of X_B, Π, R and his private randomness. The *communication cost* of a protocol Π is the maximum length of the message sent by Alice in any execution of Π. The *communication complexity* of a problem $\mathcal{P}$ is the minimum communication cost of any protocol that solves $\mathcal{P}$.

2.4 Information Complexity and Message Compression

We prove lower bounds using the *information complexity paradigm*, which is a framework that is based on information theory. Let $(A, B, C) \sim \mathcal{D}$ be jointly distributed random variables according to distribution $\mathcal{D}$. We denote the *Shannon entropy* of A by $H_{\mathcal{D}}(A)$, the entropy of A conditioned on B by $H_{\mathcal{D}}(A \mid B)$, the *mutual information* of A and B by $I_{\mathcal{D}}(A \ : \ B)$, and the conditional mutual information between A and B conditioned on C by $I_{\mathcal{D}}(A \ : \ B \mid C)$. We may drop the subscript $\mathcal{D}$ in $H_{\mathcal{D}}(.)$ and $I_{\mathcal{D}}(.)$ if it is clear from the context.

We will use the following standard facts about entropy and mutual information: (let $(A, B, C, D) \sim \mathcal{D}$ be jointly distributed random variables.)

P1: If A and C are independent conditioned on D then: $I(A \ : \ B \mid D) \leq I(A \ : \ B \mid C, D)$
P2: $I(A \ : \ B \mid C, D) = \mathbb{E}_{d \leftarrow D} I(A \ : \ B \mid C, D = d)$
P3: Let E be an event independent of A, B, C. Then: $I(A \ : B \mid C, E) = I(A \ : B \mid C)$
P4: $I(A, B \ : C \mid D) = I(A \ : C \mid D) + I(B \ : C \mid D, A)$

Given a one-way two-party communication protocol Π and an input distribution $(X_A, X_B) \sim \mathcal{D}$, we will measure the amount of information that the message Π reveals about Alice's input under distribution $\mathcal{D}$. The following quantity is denoted the *external information cost* of Π:

▶ **Definition 8.** *The (external) information cost* $\mathrm{ICost}_{\mathcal{D}}(\Pi)$ *of the one-way two-party communication protocol* Π *under input distribution* $\mathcal{D}$ *is defined as:*

$$\mathrm{ICost}_{\mathcal{D}}(\Pi) = I_{\mathcal{D}}(X_A \ : \ \Pi \mid R) \ .$$

Then, for a given problem $\mathcal{P}$, we denote the *information complexity* $\mathrm{IC}_{\mathcal{D}}(\mathcal{P})$ of $\mathcal{P}$ under distribution $\mathcal{D}$ as the minimum information cost of a protocol that solves $\mathcal{P}$.

It is well-known that information cost of a protocol lower bounds its communication cost.

We will also use a *message compression* result, which is due to Harsha et al. [21]. We follow the presentation of this result given in [31].

▶ **Theorem 9** (Message Compression). *Let Π be a protocol in the one-way two-party communication setting. Then, the protocol can be simulated with a different protocol that sends a message of expected size at most*

$$I_{\mathcal{D}}(X_A \ : \ \Pi) + 2 \cdot \log\left(1 + I_{\mathcal{D}}(X_A \ : \ \Pi)\right) + O(1) \ .$$

3 Algorithm

We will first describe and analyze our sampling-based algorithm in Subsection 3.1, and then discuss implementations of this algorithm in streaming models in Subsection 3.2.

3.1 Sampling-based Algorithm

Let $G = (V, E)$ be the input graph with $n = |V|$, $m = |E|$, and average degree $d = 2\frac{m}{n}$. Consider the following algorithm:

Algorithm 1 Sampling-based algorithm for constructing a path of length $\Omega(d)$.

Require: $G = (V, E)$
Sample $10 \cdot n \cdot \ln n$ random edges from E and denote this set by F
return Longest path in $G[F]$

We show that Algorithm 1 constructs a path of length at least $d/3$ with high probability.

▶ **Theorem 10.** *Algorithm 1 constructs a path of length $d/3$ with probability at least $1 - \frac{1}{n^2}$. It can also be regarded as an $O(\frac{n}{d})$-approximation algorithm for Longest Path.*

Proof. Given the input graph $G = (V, E)$, let $U \subseteq V$ denote a subset of vertices of V such that $G[U]$ has minimum degree at least $d/2$. It is well-known that such a subset of vertices exists (see Lemma 19 for completeness).

Let $u_0 \in U$ be any vertex, and let $P_0 = \{u_0\}$ be the path of length 0 with start and end point u_0. We will extend P_0 using the edges in F that are also contained in $G[U]$, as follows:

In step $i = 1, 2, \ldots, d/3$, we add the edge (u_{i-1}, u_i) to P_{i-1} and obtain the path P_i. The edge (u_{i-1}, u_i) is an arbitrary edge in F incident on u_{i-1} that connects to a vertex $u_i \in U$ that has not yet been visited on the path P_{i-1}. We will now prove that such a vertex exists.

First, recall that, by definition of U, the vertex u_{i-1} has at least $d/2$ neighbors in $G[U]$. Since $i \leq d/3$, at least $d/2 - d/3 = d/6$ of these neighbors are not visited by the path P_{i-1}. Denote by $E(u_{i-1})$ this set of at least $d/6$ edges connecting u_{i-1} to not yet visited neighbors in $G[U]$. We now claim that at least one of the edges of $E(u_{i-1})$ is contained in F with high probability. Observe that at stage $i-1$, we have only learnt so far that the sample F contains the $i-1$ edges of the path P_{i-1}. Hence,

$$\begin{aligned}
&\Pr\left[F \cap E(u_{i-1}) = \varnothing \mid P_{i-1} \subseteq F\right] = \frac{\binom{m-(i-1)-|E(u_{i-1})|}{|F|-(i-1)}}{\binom{m-(i-1)}{|F|-(i-1)}} \\
&\leq \exp\left(-\frac{|E(u_{i-1})|(|F| - (i-1))}{m-(i-1)}\right) \leq \exp\left(-\frac{\frac{d}{6}(10n\ln(n) - n\ln(n))}{m}\right) \\
&= \exp(-\frac{\frac{m}{3n}(9n\log n)}{m}) = \frac{1}{n^3} \ ,
\end{aligned}$$

where we used the inequality $\frac{\binom{a-c}{b}}{\binom{a}{b}} \leq \exp\left(-\frac{bc}{a}\right)$ (see Lemma 18) to obtain the first inequality and the bound $(i-1) \leq n\ln(n)$ to obtain the second.

We can therefore extend the path with probability $1 - \frac{1}{n^3}$ at any step i. Since we run $d/3 \leq n$ steps overall to create the final path $P_{d/3}$ of length $d/3$, by the union bound, we succeed with probability at least $1 - \frac{1}{n^2}$. Last, since a longest path in G is of length at most n, the algorithm also constitutes an $O(n/d)$-approximation algorithm to Longest Path. ◀

3.2 Implementation in Streaming Models

Algorithm 1 can easily be implemented in both the insertion-only and the insertion-deletion streaming models. In the insertion-only model, a uniform sample of the edges in the stream can be obtained using *reservoir sampling* [32], and, in the insertion-deletion model, this can be achieved using ℓ_0-samplers [22] and rejection sampling. We obtain our main theorem:

▶ **Theorem 1.** *In both the insertion-only and the insertion-deletion models, for undirected graphs, there are semi-streaming algorithms that compute a path of length at least $d/3$ with high probability, where d is the average degree of the input graph.*

4 Insertion-only Lower Bound for Directed Graphs

In this section, we prove an $\Omega(n^2)$ space lower bound for one-pass streaming algorithms for LP on directed graphs that compute an $(n^{1-o(1)})$-approximation. To this end, we work with two input distributions $\mathcal{D}_{\text{SLP}}$ (**S**imple **L**ongest **P**ath) and $\mathcal{D}_{\text{LP}}$ (**L**ongest **P**ath). In Subsection 4.1, we give a lower bound on the information cost of protocols that solve $\mathcal{D}_{\text{SLP}}$ well. We achieve this by, first, proving a lower bound on the communication cost of any protocol that solves $\mathcal{D}_{\text{SLP}}$ directly via combinatorial arguments, and then employ a message compression argument that allows us to conclude that the information cost of such protocols must also be large. Then, in Subsection 4.2, we present the distribution $\mathcal{D}_{\text{LP}}$, which makes use of an (r, t)-RS-graph, and we establish a direct sum argument, showing that the information cost of protocols that solve $\mathcal{D}_{\text{LP}}$ is at least t times the information cost of protocols that solve $\mathcal{D}_{\text{SLP}}$, which bounds the information cost of protocols that solve $\mathcal{D}_{\text{LP}}$ from below. Then, since information cost is a lower bound on communication cost, we obtain our result.

4.1 A Simple Distribution

We will first work with the distribution denoted $\mathcal{D}_{\text{SLP}}(r)$, see Figure 2.

We first argue that the expected length of a longest path in $H \sim \mathcal{D}_{\text{SLP}}(r)$ is $\Omega(r)$.

▶ **Lemma 11.** *The expected length of a longest path in $H \sim \mathcal{D}_{SLP}(r)$ is bounded as follows:*

$$\mathop{\mathbb{E}}_{H \leftarrow \mathcal{D}_{SLP}(r)} \text{lp}(H) \geq 2\lambda r \geq 1.24r \ ,$$

where $\lambda = 0.62432\ldots$ is the Golomb-Dickman constant.

Proof. For an input graph $H = (A, B_1 \cup B_2, E) \sim \mathcal{D}_{\text{SLP}}(r)$ with $B_1 = \{b_1^1, \ldots, b_r^1\}$ and $B_2 = \{b_1^2, \ldots, b_r^2\}$, let H' be the graph obtained from H by *contracting* the vertex pairs b_i^1 and b_i^2, for all i, and by treating parallel edges in the resulting graph as single edges, see Figure 3 for an illustration. It is easy to see that $\text{lp}(H') = \text{lp}(H)$. Furthermore, denote by $\mathcal{D}'_{\text{SLP}}(r)$ the distribution of H'. Observe that $\mathcal{D}_{\text{SLP}}(r)$ and $\mathcal{D}'_{\text{SLP}}(r)$ are uniform distributions over their respective supports.

Next, consider the set Σ_r of permutations of the set $\{1, 2, \ldots, r\}$. Consider now the bijection $f : \Sigma_r \to \text{range}(\mathcal{D}'_{\text{SLP}}(r))$, where a permutation $\sigma \in \Sigma_r$ is mapped to the graph that contains the edges $(b_i, a_{\sigma(i)})$, for all i. Then, we observe that, for a permutation $\sigma \in \Sigma_r$, the length of the longest cycle $\text{lc}(\sigma)$ is related to $\text{lp}(f(\sigma))$ as follows:

$$2 \cdot \text{lc}(\sigma) - 1 = \text{lp}(f(\sigma)) \ .$$

Input Distribution $\mathcal{D}_{\text{SLP}}(r)$:

The directed graph $G = (A, B = B_1 \cup B_2, E)$ with $|A| = |B_1| = |B_2| = r$, $A = \{a_1, \ldots, a_r\}$, $B_1 = \{b_1^1, \ldots, b_r^1\}$, $B_2 = \{b_1^2, \ldots, b_r^2\}$, and $E = E_A \cup E_B$, where E_A are Alice's edges and E_B are Bob's edges, is obtained as follows:

Alice's Input: Edge set E_A

For each $i \in [r]$, flip an unbiased coin $X_i \in \{0, 1\}$ and if it comes out heads then insert the directed edge (a_i, b_i^1) into the graph. If it comes out tail then insert the edge (a_i, b_i^2). Observe that this constitutes a directed matching M that matches all A-vertices and exactly one of b_i^1, b_i^2, for all i.

Alice holds the edges $E_A = M$.

Bob's Input: Edge set E_B

Let N^1 be a uniform random matching between A and B_1, directed from B_1 towards A. Let N^2 be a copy of N^1, but every B_1-vertex is replaced by the corresponding B_2-vertex.

Bob holds the edges $E_B = N^1 \cup N^2$.

Figure 2 Input Distribution $\mathcal{D}_{\text{SLP}}(r)$.

It is known that the expected length of a longest cycle in a random permutation $\sigma \in \Sigma_r$ is at least $\lambda \cdot r$, where $\lambda = 0.624\ldots$ is the Golomb-Dickman constant [20, 19]. Hence, we obtain

$$\mathop{\mathbb{E}}_{\mathcal{H}' \sim \mathcal{D}'_{\text{SLP}}(r)} \text{lp}(H') \geq 2 \cdot \lambda \cdot r - 1 \geq 1.24 \cdot r \ ,$$

using the assumption that r is large enough. Since the longest paths in H and H' are identical, the result follows. ◀

Next, we show that any deterministic protocol on distribution $\mathcal{D}_{\text{SLP}}(r)$ that communicates at most $\frac{1}{100}r$ bits outputs a path of length $O(\log r)$ with high probability over $\mathcal{D}_{\text{SLP}}(r)$.

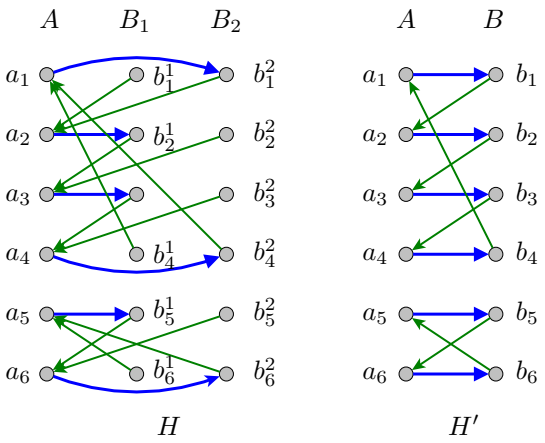
▶ **Lemma 12.** *Let Π_{SLP} be a deterministic one-way communication protocol for Longest Path on distribution $\mathcal{D}_{SLP}(r)$ that communicates at most $\frac{r}{100}$ bits. Then, the probability over the input distribution $\mathcal{D}_{SLP}(r)$ that Π_{SLP} outputs a path of length $O(\log r)$ is at least $1 - \frac{1}{500}$.*

Proof. Denote by $\mathcal{M}$ the set of possible directed input matchings M for Alice in $\mathcal{D}_{\text{SLP}}(r)$. Then, $|\mathcal{M}| = 2^r$. Next, since every message sent by protocol Π_{SLP} is of length at most $s := r/100$, there are at most 2^s different messages. On average, a message therefore corresponds to 2^{r-s} inputs, and, at most $\frac{1}{1000} \cdot 2^r$ inputs yield messages that in turn only correspond to at most $\frac{1}{1000}2^{r-s}$ inputs. Hence, at least $\frac{999}{1000}2^r$ inputs yield messages that each correspond to at least $\frac{1}{1000}2^{r-s}$ inputs. Consider now one of these messages π, and let $M_1, M_2, \ldots$ be the matchings M that correspond to π. Given π, the protocol can only output an edge if it is contained in all matchings M_i. Suppose that there are k such edges. Then, there are at most 2^{r-k} input graphs that contain these edges. We then have:

$$2^{r-k} \geq \frac{1}{1000}2^{r-s} \ ,$$

which implies $k \leq s + 10$.

Denote by K this set of at most $k \leq s + 10$ edges. We will now prove that the longest path in $K \cup E_B$ is of length $O(\log r)$ with high probability.



$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 3 & 4 & 1 & 6 & 5 \end{pmatrix}$$

Figure 3 Illustration of the proof of Lemma 11. The vertices b_i are obtained by contracting b_i^1 and b_i^2. We observe that the permutation σ has a longest cycle of length 4 $(1-2-3-4)$, while H' (and H) have longest paths of lengths 7 $(a_1, b_1, a_2, b_2, \ldots, a_4, b_4)$.

Denote by A' the A-endpoints of the edges K, and let $a'_0 \in A'$ be any vertex. We bound the length of the path with starting point a'_0. This path first uses the edge in K incident on a'_0, and denote by b'_0 the other endpoint of this edge. Then, the path uses the edge of N_1 or N_2 incident on b'_0 to return to an A-vertex that we denote by a'_1. Observe that we can only continue on this path if $a'_1 \in A'$. If this is the case then we can continue in the same fashion, visit a B-vertex b'_1, and return to another A-vertex a'_2. Again, we can only continue if $a'_2 \in A'$, and so on. For any $i \geq 1$, the following holds:

$$\Pr[a'_i \in A' \mid a'_0, \ldots, a'_{i-1} \in A'] \leq \frac{s+10-i}{r-i} \leq \frac{s+10}{r} = \frac{1}{100} + \frac{10}{r} \leq \frac{1}{50} ,$$

assuming that r is large enough. Hence, the probability that the path contains $p+2$ A'-vertices and is thus of length $2(p+1)$ is at most $\frac{1}{50^p}$. This further implies that, with probability at least $1 - \frac{1}{r^2}$, the path is of length $O(\log r)$.

Observe that the argument above applies when considering any start vertex in A'. Hence, by a union bound over all possible start vertices in A', all paths starting at an A' vertex are of length $O(\log r)$ with probability at least $1 - \frac{1}{r}$. Last, observe that a longest path that may be able to form could also start at a B-vertex. Such a path however is only by one edge longer than a path starting at an A-vertex. The $O(\log r)$ length bound therefore still holds.

So far, we proved that for a message that corresponds to at least $\frac{1}{1000}2^{r-s}$ inputs, Bob can only output a path of length at most $O(\log r)$ with high probability, where the probability is over Bob's input. Hence, overall, for a uniform input from $\mathcal{D}_{\mathrm{SLP}}(r)$, the probability that Bob will output a path of length $O(\log r)$ is at least $\frac{999}{1000} \cdot (1 - \frac{1}{r}) \geq \frac{998}{1000}$ for r large enough. ◀

We use the notation $\mathsf{LP}_{\epsilon}^{\alpha}$ to denote LP with approximation factor α and error probability ϵ.

▶ **Theorem 13.** *The randomized one-way communication complexity of* $\mathsf{LP}_{1/4}^{O(r/\log r)}$ *on inputs from* $\mathcal{D}_{SLP}(r)$ *is* $\Omega(r)$.

Proof. Towards a contradiction, let Π_{r} be a randomized protocol for Longest Path on inputs from $\mathcal{D}_{\mathrm{SLP}}(r)$ with approximation factor $o(r/\log r)$ that errs with probability at most $1/4$. Given Π_{r}, by Yao's Lemma, there exists a deterministic protocol Π_{d} on distribution $\mathcal{D}_{\mathrm{SLP}}(r)$ with distributional error $1/4$ and approximation factor $o(r/\log r)$, i.e., on at least $3/4$ of the inputs, the protocol achieves a $o(r/\log r)$-approximation.

Lemma 11 states that $\mathbb{E}_{H \leftarrow \mathcal{D}_{\mathrm{SLP}}(r)} \mathrm{lp}(H) \geq 1.24r$. This allows us to bound the quantity $\Pr_{H \leftarrow \mathcal{D}_{\mathrm{SLP}}(r)}[\mathrm{lp}(H) \geq \frac{1}{2}r]$, as follows:

$$1.24r \leq \mathop{\mathbb{E}}_{H \leftarrow \mathcal{D}_{\mathrm{SLP}}(r)} \mathrm{lp}(H) \leq \Pr_{H \leftarrow \mathcal{D}_{\mathrm{SLP}}(r)}[\mathrm{lp}(H) \geq \frac{1}{2}r] \cdot 2r + (1 - \Pr_{H \leftarrow \mathcal{D}_{\mathrm{SLP}}(r)}[\mathrm{lp}(H) \geq \frac{1}{2}r])\frac{1}{2}r$$
$$= r \cdot \left(1.5 \cdot \Pr_{H \leftarrow \mathcal{D}_{\mathrm{SLP}}(r)}[\mathrm{lp}(H) \geq \frac{1}{2}r] + \frac{1}{2}\right) ,$$

where we used the fact that the longest path in H is at most the number of vertices in H, i.e., $2r$. The previous inequality then implies that $\Pr_{H \leftarrow \mathcal{D}_{\mathrm{SLP}}(r)}[\mathrm{lp}(H) \geq \frac{1}{2}r] \geq \frac{1.24-0.5}{1.5} \geq \frac{1}{3}$.

Next, Lemma 12 states that, with probability at least $1 - \frac{1}{500}$, Π_{d} outputs a path of length at most $O(\log r)$. Hence, with probability at least $1/3 - 1/500 > 1/4$, there simultaneously exists a longest path of length at least $\frac{1}{2}r$ and Π_{d} outputs one of length $(O \log r)$. The approximation factor of Π_{d} is therefore $\Omega(r/\log r)$, contradicting the fact that Π_d achieves an approximation factor of $o(r/\log r)$ on at least $3/4$ of the instances. Protocols Π_d and Π_r therefore do not exist, which completes the proof. ◀

▶ **Lemma 14.** *Let Π be a randomized protocol for $\mathsf{LP}^{O(r/\log r)}_{1/4-\frac{1}{100}}$ on inputs from $\mathcal{D}_{SLP}(r)$. Then:*

$$\mathrm{ICost}_{\mathcal{D}_{SLP}(r)}(\Pi) = \Omega(r) .$$

Proof. Denote by Π a randomized one-way two-party communication protocol for $\mathsf{LP}^{O(r/\log r)}_{1/4-\frac{1}{100}}$ on inputs from $\mathcal{D}_{\mathrm{SLP}}(r)$. Given Π, using the message compression technique stated in Theorem 9, we obtain a protocol Π' for $\mathsf{LP}^{O(r/\log r)}_{1/4-\frac{1}{100}}$ that sends a message of expected size (recall that M is Alice's input matching)

$$s := I_{\mathcal{D}_{\mathrm{SLP}}(r)}(M \ : \ \Pi) + 2 \cdot \log\left(1 + I_{\mathcal{D}_{\mathrm{SLP}}(r)}(M \ : \ \Pi)\right) + O(1) , \tag{1}$$

where the expectation is taken over the randomness used by the protocol. Then, by the Markov inequality, the probability that the message is of size at least $100 \cdot s$ is at most $\frac{1}{100}$.

Given Π', we now construct a protocol Π'' with maximum message size $100 \cdot s$ that solves LP with slightly increased error: Whenever Π' sends a message of size at most $100 \cdot s$, Π'' also sends this message and the protocol behaves exactly the same as Π'. However, when Π' sends a message of size at least $100 \cdot s$, Π'' aborts and thus fails. Since the probability of sending a message of size larger than $100 \cdot s$ is $1/100$, the error probability of Π'' is $(\frac{1}{4} - \frac{1}{100}) + \frac{1}{100} = \frac{1}{4}$. From Theorem 13, we obtain that the communication cost of Π'' is $\Omega(r)$, which implies that $100 \cdot s = \Omega(r)$. Hence, combined with Inequality 1, we conclude that

$$\Omega(r) = I_{\mathcal{D}_{\mathrm{SLP}}(r)}(M \ : \ \Pi) \leq I_{\mathcal{D}_{\mathrm{SLP}}(r)}(M \ : \ \Pi \mid R) = \mathrm{ICost}_{\mathcal{D}_{\mathrm{SLP}}(r)}(\Pi) ,$$

where the inequality follows from property **P1** stated in the preliminaries. ◀

4.2 Hard Input Distribution and Direct Sum Argument

The input distribution $\mathcal{D}_{\mathrm{LP}}(n)$ used to obtain our main lower bound is stated in Figure 4.

We now argue that a protocol $\Pi_{\mathrm{LP}}(n)$ that solves LP under distribution $\mathcal{D}_{\mathrm{LP}}(n)$ can be used to obtain a protocol $\Pi_{\mathrm{SLP}}(r)$ that solves LP under $\mathcal{D}_{\mathrm{SLP}}(r)$, where r is the size of the induced matchings of the RS-graph used in $\mathcal{D}_{\mathrm{LP}}(n)$. This establishes a connection between the information costs of $\Pi_{\mathrm{LP}}(n)$ and $\Pi_{\mathrm{SLP}}(r)$. We use the reduction stated in Algorithm 2.

This reduction has the following properties:

Input Distribution $\mathcal{D}_{\text{LP}}(n)$:

Let $H = (A^H, B^H, E^H)$ be an (r,t)-RS graphs with $|A^H| = |B^H| = n$ and induced matchings $M_1^H, M_2^H, \ldots, M_t^H$.

Alice's Input: Edge set E_A

Given $H = (A^H, B^H, E^H)$, we construct the directed bipartite graph $G = (A, B = B_1 \dot\cup B_2, E)$, where:

- A is a copy of A^H, and B_1 and B_2 are copies of B^H.
- For every edge $e = \{a, b\} \in E^H$, we flip an unbiased coin $X_{a,b}$. If it comes out heads then the directed edge (a, b) between the sets A and B_1 is introduced, and if it comes out tail then the directed edge (a, b) between the sets A and B_2 is introduced.

We denote the edges in E that originated from the matching M_i^H, for any i, by M_i. Alice holds the edges $E_A = \cup_i M_i$.

Bob's Input: Edge set E_B

- Let $J \in [t]$ be a uniform random index.
- Consider the matching $M_J^H \in E_H$ and let $A'_H = A(M_J^H)$ and $B'_H = B(M_J^H)$. Let N_J be a uniform random matching between A' and B'.
- Bob introduces two copies of N_J into G: To this end, let $A' \subseteq A$ be the copy of A'_H in G, let $B'_1 \subseteq B_1$ be the copy of B' in B_1, and let $B'_2 \subseteq B_2$ be the copy of B' in B_2. The first copy N_J^1 is introduced between A' and B'_1, and the second copy N_J^2 is introduced between A' and B'_2. The edges in N_J^1 and N_J^2 are directed such that the A-vertex constitutes the head and the B-vertex the tail of the directed edge.

Bob holds the edges $E_B = N_J^1 \cup N_J^2$.

Figure 4 Input Distribution $\mathcal{D}_{\text{LP}}(n)$.

Algorithm 2 Construction of protocol $\Pi_{\text{SLP}}(r)$.

Require:

1. Protocol $\Pi_{\text{LP}}(n)$ that solves $\mathsf{LP}^{\alpha}_{1/4}$ under distribution $\mathcal{D}_{\text{LP}}(n)$
2. Input $(M, N^1, N^2) \sim \mathcal{D}_{\text{SLP}}(r)$, Alice holds matching M, Bob holds matchings N^1, N^2

1: Alice and Bob use public randomness to generate a uniform random index $J \in [t]$
2: Alice samples graph $G \sim \mathcal{D}_{\text{LP}}(n)$ and updates matching M_J in G such that $M_J = M$
3: Bob set $N_J^1 = N^1$ and $N_J^2 = N^2$ and adds these to G
4: Alice and Bob run the protocol $\Pi_{\text{LP}}(n)$ on G, denote by $\mathcal{P}$ the output of $\Pi_{\text{LP}}(n)$
5: **return** Longest sub-path of $\mathcal{P}$ that solely uses edges in $M_J \cup N_J^1 \cup N_J^2$

▶ **Lemma 15.** *Consider the reduction in Algorithm 2. Given any path $\mathcal{P}$ in G of length at least 2, the path $\mathcal{P}'$ with the first and last edge removed only uses edges from $M_J \cup N_J^1 \cup N_J^2$.*

Proof. Let $\mathcal{P}$ be a path of length at least 2 in G. Since G is bipartite, every other edge in $\mathcal{P}$ must be an edge from $N_J^1 \cup N_J^2$ since these are the only edges with head in A. Observe that the edges M_J are the only edges with both endpoints in $A(N_J^1 \cup N_J^2)$ and $B(N_J^1 \cup N_J^2)$. Hence, for any three consecutive edges e, f, g in path $\mathcal{P}$, if e and g are edges from $N_J^1 \cup N_J^2$, then f must be an edge from M_J. It follows that if an edge that is not contained in $M_J \cup N_J^1 \cup N_J^2$ is included in $\mathcal{P}$ then this edge must be the first or the last edge of $\mathcal{P}$. ◀

Given our reduction, we now relate the information costs of $\Pi_{\text{SLP}}(r)$ and $\Pi_{\text{LP}}(n)$:

▶ **Lemma 16.** *Let $\Pi_{LP}(n)$ be a protocol for LP^{α}_{ϵ}, for some parameters α and ϵ, and let $\Pi_{SLP}(r)$ be the protocol obtained from $\Pi_{LP}(n)$ via the reduction given in Algorithm 2. Then, $\Pi_{SLP}(r)$ has approximation factor $O(\alpha)$, errs with the same probability ϵ, and:*

$$\mathrm{ICost}_{\mathcal{D}_{LP}(n)}(\Pi_{LP}(n)) = t \cdot \mathrm{ICost}_{\mathcal{D}_{SLP}(r)}(\Pi_{SLP}(r)) \ .$$

Proof. We denote by R_{SLP} and R_{LP} the public randomness used in $\Pi_{\mathrm{SLP}}(r)$ and in $\Pi_{\mathrm{LP}}(n)$, respectively. Then (see the rules **P1**, ..., **P4** in the preliminaries):

$$\begin{aligned}
\mathrm{ICost}_{\mathcal{D}_{\mathrm{SLP}}(r)}(\Pi_{\mathrm{SLP}}(r)) &= I_{\mathcal{D}_{\mathrm{SLP}}(r)}(M \ : \ \Pi_{\mathrm{SLP}}(r) \mid R_{\mathrm{SLP}}) \\
&= I_{\mathcal{D}_{\mathrm{SLP}}(r)}(M_J \ : \ \Pi_{\mathrm{LP}(n)} \mid R_{\mathrm{LP}}, J) \\
&= \mathbb{E}_{j \leftarrow J} \, I_{\mathcal{D}_{\mathrm{LP}}(n)}(M_J \ : \ \Pi_{\mathrm{LP}(n)} \mid R_{\mathrm{LP}}, J = j) && \textbf{P2} \\
&= \frac{1}{t} \cdot \sum_{j \in [t]} I_{\mathcal{D}_{\mathrm{LP}}(n)}(M_j \ : \ \Pi_{\mathrm{LP}(n)} \mid R_{\mathrm{LP}}, J = j) \\
&= \frac{1}{t} \cdot \sum_{j \in [t]} I_{\mathcal{D}_{\mathrm{LP}}(n)}(M_j \ : \ \Pi_{\mathrm{LP}(n)} \mid R_{\mathrm{LP}}) && \textbf{P3} \\
&\leq \frac{1}{t} \cdot \sum_{j \in [t]} I_{\mathcal{D}_{\mathrm{LP}}(n)}(M_j \ : \ \Pi_{\mathrm{LP}(n)} \mid M_1, \ldots, M_{j-1}, R_{\mathrm{LP}}) && \textbf{P1} \\
&= \frac{1}{t} \cdot I_{\mathcal{D}_{\mathrm{LP}}(n)}(M_1, \ldots, M_t \ : \ \Pi_{\mathrm{LP}(n)} \mid R_{\mathrm{LP}}) && \textbf{P4} \\
&= \mathrm{ICost}_{\mathcal{D}_{\mathrm{LP}}(n)}(\Pi_{\mathrm{LP}}(n)) \ .
\end{aligned}$$

Regarding the approximation factor, observe that $\mathrm{lp}(H) \leq \mathrm{lp}(G)$. Furthermore, by Lemma 15, the path found by $\Pi_{\mathrm{SLP}}(r)$ is at most by 2 shorter than the path found by $\Pi_{\mathrm{LP}}(n)$, which in turn is of length at least $\mathrm{lp}(G)/\alpha$. Hence, the approximation factor of $\Pi_{\mathrm{SLP}}(r)$ is bounded by $\frac{\mathrm{lp}(H)}{\mathrm{lp}(G)/\alpha - 2} \leq \frac{\mathrm{lp}(G)}{\mathrm{lp}(G)/\alpha - 2} = O(\alpha)$. ◀

We are now ready to state our main lower bound result for directed graphs.

▶ **Theorem 17.** *The randomized one-way communication complexity of $LP^{O(r/\log r)}_{1/4-\frac{1}{500}}(n)$ is $\Omega(r \cdot t)$.*

Proof. Let $\Pi_{\mathrm{LP}}(n)$ be a protocol for $\mathsf{LP}^{O(r/\log r)}_{1/4-\frac{1}{500}}(n)$. Then, from Lemmas 16 and 14, we obtain

$$\mathrm{ICost}_{\mathcal{D}_{\mathrm{LP}}(n)}(\Pi_{\mathrm{LP}}(n)) = \Omega(r \cdot t) \ .$$

Since the choice of $\Pi_{\mathrm{LP}}(n)$ was arbitrary, and information complexity is a lower bound on communication complexity, the result follows. ◀

Finally using the well-known connection between streaming algorithms and one-way two-party communication protocols as well as the RS-graph construction by Alon et al. as stated in Theorem 7, we obtain the following theorem:

▶ **Theorem 3.** *Any one-pass algorithm for $LP^{O(n^{1-o(1)}/\log(n^{1-o(1)})}_{1/4-\frac{1}{500}}(n) = LP^{O(n^{1-o(1)})}_{1/4-\frac{1}{500}}(n)$ requires space $\Omega(n^2)$.*

5 Conclusion

We studied one-pass streaming algorithms for LP and showed that, in both insertion-only and insertion-deletion streams, for undirected graphs, there are semi-streaming algorithms that find paths of lengths at least $\frac{d}{3}$ with high probability, where d is the average degree of the input graph. The algorithm can also give an α-approximation algorithm that uses space $\tilde{O}(n^2/\alpha)$. We then showed that no such result can be obtained for directed graphs in that a $n^{1-o(1)}$-approximation requires space $\Omega(n^2)$, even in insertion-only streams. We also showed that semi-streaming algorithms in the insertion-only model for undirected graphs cannot yield an arbitrarily small constant factor approximation, and, in insertion-deletion streams, space $\Omega(n^2/\alpha^3)$ is necessary to obtain an α-approximation in undirected graphs.

We conclude with two open questions: First, while we resolved the space complexity of one-pass algorithms for directed graphs in both the insertion-only and the insertion-deletion models, the space complexity for undirected graphs, in particular, in the insertion-only model, remains wide open. Can we close this gap? Second, for undirected graphs, are there multi-pass semi-streaming algorithms that compute paths longer than $\Theta(d)$, where d is the average degree of the input graph?

References

1 Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. *Algorithmica*, 83(7):1980–2017, 2021. doi:10.1007/S00453-021-00816-9.

2 Noga Alon, Ankur Moitra, and Benny Sudakov. Nearly complete graphs decomposable into large induced matchings and their applications. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19–22, 2012*, pages 1079–1090. ACM, 2012. doi:10.1145/2213977.2214074.

3 Sepehr Assadi. A two-pass (conditional) lower bound for semi-streaming maximum matching. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9–12, 2022*, pages 708–742. SIAM, 2022. doi:10.1137/1.9781611977073.32.

4 Sepehr Assadi, Soheil Behnezhad, Christian Konrad, Kheeran K. Naidu, and Janani Sundaresan. Settling the pass complexity of approximate matchings in dynamic graph streams. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 864–904. SIAM, 2025. doi:10.1137/1.9781611978322.25.

5 Sepehr Assadi, Aaron Bernstein, Zachary Langley, Lap Chi Lau, and Robert Wang. Streaming and communication complexity of load-balancing via matching contractors. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 3423–3449. SIAM, 2025. doi:10.1137/1.9781611978322.113.

6 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786. SIAM, 2019. doi:10.1137/1.9781611975482.48.

7 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1723–1742. SIAM, 2017. doi:10.1137/1.9781611974782.113.

8 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364. SIAM, 2016. doi:10.1137/1.9781611974331.CH93.

9 Sepehr Assadi, Christian Konrad, Kheeran K. Naidu, and Janani Sundaresan. O(log log n) passes is optimal for semi-streaming maximal independent set. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 847–858. ACM, 2024. doi:10.1145/3618260.3649763.

10 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.

11 Andreas Björklund, Thore Husfeldt, and Sanjeev Khanna. Approximating longest directed paths and cycles. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming*, pages 222–233, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-27836-8_21.

12 Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova. Vertex ordering problems in directed graph streams. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1786–1802. SIAM, 2020. doi:10.1137/1.9781611975994.109.

13 Yi-Jun Chang, Martin Farach-Colton, Tsan-sheng Hsu, and Meng-Tsung Tsai. Streaming complexity of spanning tree computation. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 34:1–34:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.STACS.2020.34.

14 Graham Cormode, Jacques Dark, and Christian Konrad. Independent sets in vertex-arrival streams. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 45:1–45:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.45.

15 Jacques Dark and Christian Konrad. Optimal lower bounds for matching and vertex cover in dynamic graph streams. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CCC.2020.30.

16 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. doi:10.1016/J.TCS.2005.09.013.

17 Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. Monotonicity testing over general poset domains. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 474–483. ACM, 2002. doi:10.1145/509907.509977.

18 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485. SIAM, 2012. doi:10.1137/1.9781611973099.41.

19 S. W. Golomb. Random permutations. *Bull. Amer. Math. Soc. 70*, 1964.

20 S. W. Golomb, L. R. Welch, and R. M. Goldstein. Cycles from nonlinear shift registers. Technical report, Progress Rep. No. 20-389, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, 1959.

21 Prahladh Harsha, Rahul Jain, David A. McAllester, and Jaikumar Radhakrishnan. The communication complexity of correlation. In *22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA*, pages 10–23. IEEE Computer Society, 2007. doi:10.1109/CCC.2007.32.

22 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58. ACM, 2011. doi:10.1145/1989284.1989289.

23 Ramkumar G. D. S. Karger D, Motwani R. On approximating the longest path in a graph. *Algorithmica*, pages 82–98, 1997. doi:10.1007/BF02523689.

24 Shahbaz Khan and Shashank K. Mehta. Depth First Search in the Semi-streaming Model. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2019.42.

25 Lasse Kliemann, Christian Schielke, and Anand Srivastav. A streaming algorithm for the undirected longest path problem. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 56:1–56:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ESA.2016.56.

26 Christian Konrad and Kheeran K. Naidu. On two-pass streaming algorithms for maximum bipartite matching. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 19:1–19:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.APPROX/RANDOM.2021.19.

27 Christian Konrad and Kheeran K. Naidu. An unconditional lower bound for two-pass streaming algorithms for maximum matching approximation. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 2881–2899. SIAM, 2024. doi:10.1137/1.9781611977912.102.

28 Christian Konrad, Kheeran K. Naidu, and Arun Steward. Maximum matching via maximal matching queries. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 41:1–41:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.STACS.2023.41.

29 Andrew McGregor. Finding graph matchings in data streams. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2005. doi:10.1007/11538462_15.

30 Slobodan Mitrovi?, Anish Mukherjee, Piotr Sankowski, and Wen-Horng Sheu. Faster semi-streaming matchings via alternating trees, 2025. doi:10.48550/arXiv.2412.19057.

31 Anup Rao and Amir Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020.

32 Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985. doi:10.1145/3147.3165.

A Technical Lemmas

▶ **Lemma 18.** *Let a, b, c be positive with $a - c \geq b$. Then:* $\frac{\binom{a-c}{b}}{\binom{a}{b}} \leq \exp(-\frac{b \cdot c}{a})$.

Proof. We compute as follows:

$$\frac{\binom{a-c}{b}}{\binom{a}{b}} = \frac{\frac{(a-c)!}{(a-b-c)!b!}}{\frac{a!}{(a-b)!b!}} = \frac{(a-c)\cdot(a-c-1)\cdot\ldots\cdot(a-b-c+1)}{a\cdot(a-1)\cdot\ldots\cdot(a-b+1)} \leq (1-\frac{c}{a})^b \leq \exp(-\frac{bc}{a}) ,$$

where we used the inequality $1 + x \leq \exp(x)$, which holds for all x. ◀

▶ **Lemma 19.** *Let $G = (V, E)$ be a graph with $|V| = n$, $|E| = m$, and average degree $d = 2\frac{m}{n}$. Then, there exists a subset of vertices $U \subseteq V$ such that the vertex-induced subgraph $G[U]$ has minimum degree greater than $\frac{d}{2}$.*

Proof. We iteratively remove vertices of degree at most $d/2$ from G until no such vertex is left. Let G_i be the graph with the first i vertices removed, and let $G_0 = G$. We denote m_i the number of edges in G_i, and $n_i = n - i$ the number of vertices in G_i. It can then be seen by induction that the average degree d_i of every graph G_i is still at least $d_0 = d$. Indeed, observe that removing a vertex of degree at most $d/2$ removes at most $d/2$ edges from the graph. Then, the average degree of graph G_{i+1} is:

$$2 \cdot \frac{m_{i+1}}{n_{i+1}} \geq 2 \cdot \frac{m_i - \frac{d}{2}}{n-(i+1)} = 2 \cdot \frac{\frac{d}{2}(n-i) - \frac{d}{2}}{n-(i+1)} = d .$$

Then, since the average degree remains as high as d throughout, and we only ever remove vertices of degree at most $d/2$, the process must leave a non-empty graph with minimum degree at least $\lfloor d/2 + 1 \rfloor$ behind. The set U then is the set of vertices that are not removed by this process. ◀

Computing Largest Subsets of Points Whose Convex Hulls Have Bounded Area and Diameter

Gianmarco Picarella
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Marc van Kreveld
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Frank Staals
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Sjoerd de Vries
Department of Information and Computing Sciences, Utrecht University, The Netherlands
Department of Digital Health, University Medical Center Utrecht, The Netherlands

Abstract

We study the problem of computing a convex region with bounded area and diameter that contains the maximum number of points from a given point set P. We show that this problem can be solved in $O(n^6k)$ time and $O(n^3k)$ space, where n is the size of P and k is the maximum number of points in the found region. We experimentally compare this new algorithm with an existing algorithm that does the same but without the diameter constraint, which runs in $O(n^3k)$ time. For the new algorithm, we use different diameters. We use both synthetic data and data from an application in cancer detection, which motivated our research.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases convex polygon, dynamic programming, implementation

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.23

Related Version *Full Version*: `https://arxiv.org/abs/2507.04933` [8]

Supplementary Material *Software (Source Code)*: `https://github.com/gianmarcopicarella/area-diameter-algorithm`, archived at `swh:1:dir:d5b399b88843edd53f14c5ed19d8459d7018cd36`

1 Introduction

Medical image data is examined by specialists to detect anomalies. For example, human tissue may contain mitotic cells, where dense areas are indicative of the presence of breast cancer. Therefore, pathologists used to count mitotic cells in small areas to establish a risk and treatment regime. To automate this process, mitotic cells are nowadays identified from images and converted to (planar) point sets. Dense areas are regions of a certain size and shape that contain many points, which need to be found and inspected.

According to medical specialists, a region of interest is convex, has an area of 2–4 mm^2, a bounded aspect ratio, and has the maximum number of points [9]. More generally, the identification of well-shaped, small, dense areas in point sets is a common data analysis task.

In this paper we address the problem of computing – for a point set P – a convex region with the largest number of points from P, given an upper bound on the area and on the diameter. We present both algorithmic and experimental results. Earlier research showed that a convex region with at least k points and the minimum area can be computed using dynamic programming in $O(n^3k)$ time and $O(n^2k)$ space. This algorithm can be used to compute the maximum number of points in a convex region of at most a given area, but the diameter constraint is not taken into account. Alternatively, computing a convex region with bounded diameter and the maximum number of points can be done using furthest-site

© Gianmarco Picarella, Marc van Kreveld, Frank Staals, and Sjoerd de Vries;
licensed under Creative Commons License CC-BY 4.0

Voronoi diagrams in $O(nk^{2.5} \log k + n \log n)$ time and $O(nk)$ space [1]. We show that with a dynamic programming method, we can ensure both a maximum area *and* a maximum diameter in $O(n^6 k)$ time and $O(n^3 k)$ space. A previous solution to this problem [9] takes exponential time, which made it applicable only to very small point sets. Hence, that paper also presented a heuristic to find good but not necessarily optimal solutions more efficiently.

Experimentally, we investigate three types of point set: (i) uniformly distributed in a fixed square, where we consider different numbers of points (and hence densities); (ii) normally distributed in a fixed square, with different standard deviations; (iii) real-world medical data that was also used in [9].

We investigate different maximal diameters and compare our new algorithm to the existing algorithm by Eppstein, Overmars, Rote, and Woeginger [5] that does not use the diameter constraint. In theory, their algorithm has a much better worst-case running time, and it is possible that the found area satisfies the diameter condition without explicitly enforcing it. This would be the case if densest areas by our definition typically yield roundish regions rather than elongated ones. The experiments show that this is not the case, and we do not get the diameter constraint "for free". The solutions are elongated; they have a larger diameter than expected and than is allowed in the application.

The new algorithm, which enforces a bounded diameter, has a high worst-case running time. Fortunately, the bound on the diameter allows us to perform significant pruning, making the algorithm often faster in practice for not too dense point sets when compared to the algorithm that bounds the area only, despite being a factor n^3 slower in theory. Our experiments analyze this for different data sets and different settings of the diameter with respect to the area.

Related work. The identification of dense areas – clusters – in point sets is a common task in many data analysis applications, as witnessed by the immensely popular DBSCAN method [6]. It is impossible to review the complete literature; we concentrate on algorithmic results related to this paper. These are 2-dimensional, use a clear definition of a dense area, compute exact results, and admit a running time analysis in the standard algorithmic sense.

Previous research has considered smallest area, perimeter, or diameter convex polygons that contain at least k points. The mentioned paper [5] addresses area or perimeter, and can find smallest polygons with k points on the boundary, or k points on the boundary while having no points inside, or k points in total. In all cases, an $O(n^3 k)$ time and $O(n^2 k)$ space algorithm solves the problem. More generally, the method works whenever the optimization concerns a "monotone decomposable weight function": any function where the value for a convex polygon can be obtained from the values of two parts (and possibly the chord used to split in those parts). Eppstein [4] later presented an alternative method for the problem of computing a minimum area convex polygon with k points inside that is more efficient when k is sufficiently small. It runs in $O(n^2 \log n + n^2 k^3)$ time and $O(n \log n + k^3)$ space.

Around the same time, Arkin, Khuller and Mitchell considered geometric knapsack problems [2], which include computing a minimum area convex polygon containing k points. They obtain essentially the same $O(n^3 k)$ time dynamic programming algorithm, but include some other generalizations than the ones in [5].

The problem of computing a convex polygon that contains k points and has minimum diameter uses a different approach. Note that the diameter is not a monotone decomposable weight function, and hence the approach in [5] does not work directly. Aggarwal, Imai, Katoh and Suri [1] propose an $O(k^{2.5} n \log k + n \log n)$ time and $O(kn)$ space algorithm for this problem using higher-order Voronoi diagrams.

For smallest k-point enclosing rectangle or circle algorithms, see [3, 7].

Our research was motivated by [9], where the problem of computing a convex region with bounded area and bounded aspect ratio that contains most points was mentioned. The aspect ratio is the diameter-to-width ratio. The paper describes an automated mitosis detection pipeline that includes an algorithm for the problem mentioned. The algorithm first cover the region with points into square patches that partially overlap, and then finds solutions in these patches by one of two methods, a brute-force method and a heuristic.

Note that bounding the aspect ratio is not the same as bounding the diameter, but in combination with a bounded area they realize the same goal of finding not too elongated areas. A constant upper bound on the aspect ratio guarantees a constant upper bound on the diameter, but the reverse is not true. If all points are (nearly) collinear, an upper-bounded aspect ratio cannot be attained by a convex hull of any subset (or perhaps only by a singleton point), whereas an upper-bounded diameter solution always exists, and might be desired.

Organization. This paper is organized as follows. In Section 2 we review the algorithm from [5] that computes a smallest area convex set with k points inside. We also present our new algorithm that computes a smallest area convex set with bounded diameter and k points inside, and prove its correctness and running time. In Section 3 we describe the experimental set-up, including the data used and implementation aspects. In Section 4 we present the results of the experiments and discuss them. These results concern running time, memory used, number of points inside, and realized area and diameter of the solution.

2 Algorithms

In this section we present two dynamic programming algorithms to compute minimum area polygons containing k points. The former algorithm appeared in [5]; we repeat it since our new algorithm can be seen as an extension of it. The latter is the new algorithm, which incorporates a given diameter bound. Both algorithms are used in our experiments.

We start with some notation that is used in both algorithms. Let P be a set of n points in the plane, and assume we are searching for a solution containing $2 \leq k \leq n$ points. Assume, for ease of description, that no three points in P are collinear. Observe that a minimum area solution with at least k points will always be a minimum area solution with exactly k points, because if the solution had more than k points from P, we can always exclude a corner of the convex solution and reduce its area, while reducing the number of points by one only.

For three points a, b, c, let $A(a, b, c)$ denote the area of the triangle $\triangle abc$ whose corners are a, b, c, and let $K(a, b, c)$ denote the number of points from P in this triangle. Points of P on the boundary of $\triangle abc$ are not counted.

For a convex polygon C, the diameter is the distance between the two points of C that realize the largest distance. Two points that realize the diameter are called a diametral pair. In our context, points will always be points that are in the set P, because C is always the convex hull of some subset of the points in P. A pair of points a, b of C is antipodal if two parallel lines exist, one through a and one through b, such that the interior of polygon C lies between these lines (inside the slab). When the vertices of C are in general position (no two sides are parallel), a convex polygon with m vertices has m antipodal pairs. If there are parallel sides, the number of antipodal pairs can be larger but is still $O(m)$. It is well-known that for a convex polygon C, its diameter is realized by an antipodal pair of vertices of C.

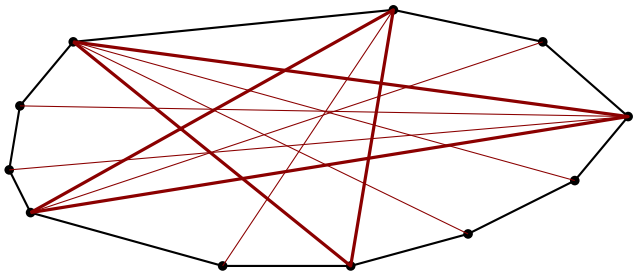


Figure 1 The antipodal structure of a convex polygon. Bold chords form a five-pointed star in brown, the six thin brown edges connect to leafs and split star angles.

If we assume that no two sides of C are parallel, the antipodal pairs have a clear structure. If we draw them as chords in C, then we get one odd-sided star plus zero or more edges that end in leaves, which always split an angle of a star vertex opposite the leaf, see Figure 1. This is true because any two chords have an endpoint in common or they intersect properly. Note that an antipodal pair may be the endpoints of an edge that bounds C.

The standard method to find all antipodal pairs of a convex polygon in linear time is referred to as rotating calipers [10]. It comes down to a traversal of the boundary of C on opposite sides in parallel, using the slopes of the edges to decide on which side we make the next step and find a new antipodal pair. This general idea will be integrated into the dynamic programming algorithm for the bounded-diameter case.

2.1 Computing a minimum area convex polygon containing k points

We describe the algorithm of Eppstein, Overmars, Rote and Woeginger [5].

Take any point $p \in P$ and choose it to be the bottommost vertex of a set of candidate solutions. In other words, we search for the optimal solution among all convex polygons with k points inside that have p as the bottommost vertex. By iterating over all points $p \in P$, we will find the global optimum with a linear factor extra in the time bound.

Given the bottommost point p in the solution, we can use a recursive structure when setting up the dynamic program by considering a bottom-vertex triangulation. This is a triangulation where all vertices on the boundary connect to p. A convex polygon C is now partitioned into a fan of triangles around p and above it.

Assume the vertices are listed in clockwise order around C as $p, p_1, \ldots, p_m$. Then the last (most clockwise) triangle in the fan around p is $\triangle p p_{m-1} p_m$.

If C is an optimal solution with k points from P, then the convex polygon $p, p_1, \ldots, p_{m-1}$ is an optimal solution with $k - K(p, p_{m-1}, p_m) - 1$ points among the solutions that have p_{m-1} as the counterclockwise neighbor of p and where p_{m-2}, p_{m-1}, p_m makes a right turn (is convex at p_{m-1} in C). This suggests a 3-dimensional table $T[q, r, k]$, defined as: the minimum area of a convex polygon with p as the bottommost vertex, q as its ccw (counterclockwise) neighbor, and r as the ccw neighbor of q. See Figure 2. This leads to a standard dynamic program with the recurrence:

$$T[q,\, r,\, k] = A(p, q, r) + \min_s\, T[r,\, s,\, k - K(p, q, r) - 1]\,,$$

where s is chosen over the points above the horizontal line through p, left of the directed line from p to r, and that leads to convexity at r. Since $K(p, q, r)$ counts only points strictly inside $\triangle pqr$, we subtract one more for point q. As the base case we let $T[q,\, r,\, k] = A(p, q, r)$ if triangle $\triangle pqr$ has exactly k points, p, q, and r included.

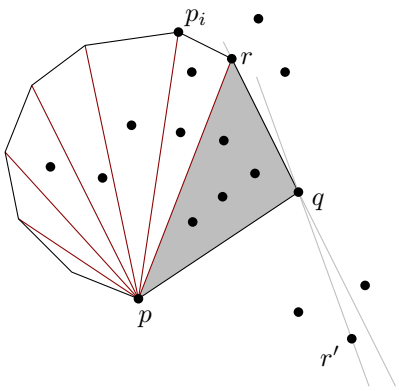


Figure 2 An optimal solution can be decomposed in $\triangle pqr$ and an optimal convex polygon with area $T[r, p_i, k - K(p, q, r) - 1]$. Table entries can be computed in $O(1)$ time by rotating around q.

It is easy to see that filling an entry takes $O(n)$ time, and the whole dynamic program requires $O(n^3k)$ time. Together with the fact that we try all n points as the bottommost point p, the whole algorithm takes $O(n^4k)$ time.

In [5] a technique is described that reduces the running time by a linear factor. A similar table is used, assuming a given point p as bottommost point, but with a subtly different definition: $T[q, r, k]$ is the smallest area convex polygon with k points inside where p is the bottommost vertex, q is its ccw neighbor, and the convex polygon lies on the same side of the line through q and r as p does. Note that r might not be part of the convex polygon itself. With a rotational sweep around q, entries are filled in only $O(1)$ time per point r. We consider all points r above a horizontal line through p, not just the ones left of the directed line through p and q. We consider the full lines through r and q and order them by angle around q, starting clockwise from the line through p and q.

The rotational sweep maintains a value T_q^* that is the minimum area encountered so far with k points. The sweep has two cases: (i) Point r lies left of the directed line through p and q (in that order), in which case we may use $T[q, r, k - K(p, q, r) - 1]$ to potentially find a smaller T_q^*, in which case we update T_q^* and fill $T[q, r, k]$; (ii) Point r lies to the right of the directed line, in which case we fill $T[q, r, k]$ with T_q^*. This leads to an $O(n^3k)$ time algorithm. Details can be found in [5].

2.2 Computing a minimum area convex polygon with bounded diameter containing k points

In this section we show how to compute a minimum area convex polygon with bounded diameter and k points inside. We do this by trying all pairs of points $p, q, \in P$, assuming they form the diameter, and then computing an optimal convex polygon under this assumption. The algorithm is an extension of the basic, $O(n^4k)$ time algorithm from [5]. At the end of this section we show how to use this algorithm to find a convex polygon with bounded diameter and area and the maximum number of points inside. The full version [8] contains omitted proofs.

After choosing a pair p, q, we first select all points of P that are at most $|pq|$ from both p and from q. They are the points in a lune formed by the intersection of two disks with radius $|pq|$, centered at p and at q, see Figure 3. For ease of description and without loss of generality, we let pq be vertical with p below q. We now need to form the right chain of a convex polygon, clockwise (cw) from q to p, and the left chain, clockwise from p to q. While

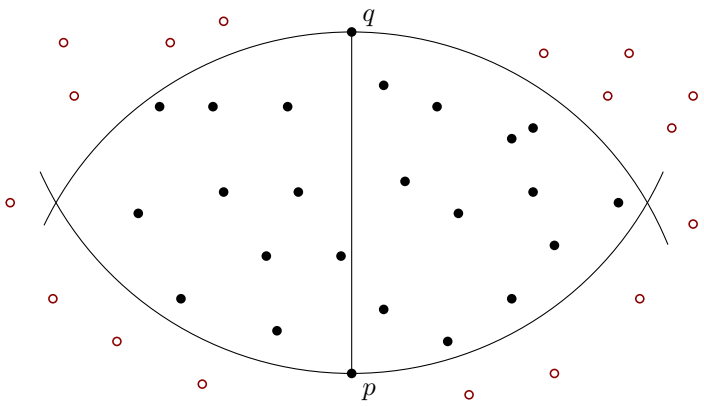


Figure 3 The black points lie inside the lune, whereas the red points do not. The latter are ignored in the dynamic program that assumes p, q is the diameter.

doing so, we must make sure that we do not consider solutions that use a point ℓ on the left chain that is too far from a point r on the right chain. Note that two points on the right chain can never be too far from each other, and the same is true for the left chain (since we restricted our points to the ones inside the lune).

To solve the problem for a given p, q we combine the dynamic programming solution from [5] with a rotating calipers algorithm to make sure that all antipodal pairs (one from the left chain and one from the right chain) have distance no more than $|pq|$.

Let $R[r, r', \ell, k]$ be the minimum area of a convex polygon with pq as a diameter, where r is the ccw neighbor of p, r' is the ccw neighbor of r, ℓ is the ccw neighbor of q, that has exactly k points inside, and where ℓ is antipodal to r and to r'. Analogously, let $L[\ell, \ell', r, k]$ be the minimum area of a convex polygon with pq as a diameter, where ℓ is the ccw neighbor of q, ℓ' is the ccw neighbor of ℓ, r is the ccw neighbor of p, that has exactly k points inside, and where r is antipodal to ℓ and to ℓ'. We allow r' to be q and ℓ' to be p. Note that p and q are antipodal: there is a pair of horizontal lines through p and through q that have any convex polygon with p, q as diameter in between. We study antipodality more closely in the following lemma.

▶ **Lemma 1.** *Let C be a convex polygon with p, q as diameter, with r as ccw neighbor of p and ℓ as ccw neighbor of q. Furthermore, assume that C has no two sides that are parallel. Then we have at least one of the following cases:*

- *r is antipodal to only q, to only ℓ, or to both but no other points.*
- *ℓ is antipodal to only p, to only r, or to both but no other points.*

From the lemma above, it follows that ℓ is not antipodal to r' or r is not antipodal to ℓ'. Now consider an optimal solution C_{opt} with a given diameter pq, and let ℓ, r, ℓ' and r' be as in the proof. If ℓ is not antipodal to r', then we can see C_{opt} as being composed of a triangle $\triangle \ell \ell' q$ and an optimal subsolution C'_{opt} where ℓ' is the ccw neighbor of q. Since ℓ is antipodal to no more than r and possibly p, by induction we can assume that C'_{opt} has all antipodal pairs no larger than the diameter $|pq|$, we need to test only the distance $|\ell r|$ to conclude that C_{opt} also has all antipodal pairs close enough. If r is not antipodal to ℓ', the argument is symmetric.

If $|r\ell| > |pq|$ then $R[r, ., \ell, .] = \infty$. Similarly, if $|r'\ell| > |pq|$ then $R[., r', \ell, .] = \infty$.

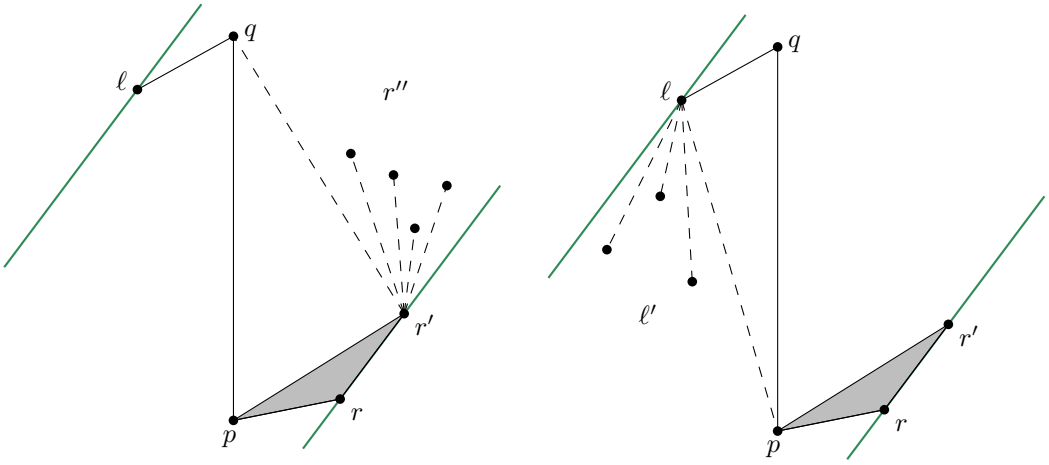


Figure 4 The two cases in the recurrence that defines $R[r, r', \ell, k]$. The grey triangle is left out when using the optimal substructure.

The recurrence optimizes over two options: use a subproblem of the R-type or use a subproblem of the L-type. The optimization of the R-type using a step on the right side is:

$$R[r,\, r',\, \ell,\, k] = A(p, r, r') + \min_{r''} R[r',\, r'',\, \ell,\, k - K(p, r, r') - 1]\,.$$

Here the r'' must be such that $r''r'r$ is a right turn at r', vertex r'' is right of the line through r' and q, and $|r''\ell| \leq |pq|$. In such a case, r' is antipodal to only ℓ. We can choose r'' to be q. If $r' = q$, then this option cannot be chosen because the right boundary is finished. Finally, ℓ can be p to capture the case that the left side is empty.

The optimization of the R-type using a step on the left side is:

$$R[r,\, r',\, \ell,\, k] = A(p, r, r') + \min_{\ell'} L[\ell,\, \ell',\, r',\, k - K(p, r, r') - 1]\,.$$

Here, ℓ' must be such that $q\ell\ell'$ is a left turn at ℓ, vertex ℓ' is to the left of the line through ℓ and p and right of the line through ℓ that is parallel to the line through r and r' (showing that ℓ' and r are not antipodal), and $|\ell' r'| \leq |pq|$. We can choose ℓ' to be p, which would complete the left boundary. Also here, r' might be q.

As base case of the recursion, we take all triangles including the points p and q. In particular, we define $R[r_1, q, p, k] = A(p, r_1, q)$ if $k = 3 + K(p, r_1, q)$, and ∞ otherwise.

The definition of $L[\ell, \ell', r, k]$ is symmetric, having analogous options. Note that since q is the unique highest point and p the unique lowest point, "left" and "right" of any line through p (or q) and another point is well-defined.

Intuitively, the algorithm constructs an optimal solution as follows, see Figure 5. Consider a *triangulation* of the solution C for a given diameter p, q where p has chords to all vertices on the right and q has chords to all vertices on the left. Let $p = r_0, r_1, \ldots, r_j = q$ be the vertices incident to edges on the right boundary. Then the edges $r_1r_2, \ldots, r_{j-1}r_j$ are the sequences of edges of the triangles in the right half that are opposite to p in these triangles. Let $q = \ell_0, \ldots, \ell_h = p$ be defined analogously for the left half.

Consider the sorted sequence (by positive turn angle from the positive x-axis vector) of edges $\vec{e}_1, \ldots, \vec{e}_{j+h-2}$ from $\overrightarrow{r_1r_2}, \ldots, \overrightarrow{r_{j-1}r_j}$ and $\overrightarrow{\ell_2\ell_1}, \overrightarrow{\ell_3\ell_2}, \ldots, \overrightarrow{\ell_h\ell_{h-1}}$. This sorted sequence defines an order on the triangles in the left and right halves of C that is a merge of the two sequences. The solution C can be deconstructed by peeling off the triangles according to

that ordered sequence. If triangle $\triangle pr_1r_2$ is first in the sequence, then C is composed of this triangle and the convex polygon C where that triangle is removed. As we argue next, this solution is found when computing $R[r_1, r_2, \ell_1, k]$, where k is the number of points of P in C.

▶ **Lemma 2.** *Let C be an optimal solution, and let $t_1 = \triangle pr_1r_2$ be the first triangle in the triangulation as described above. Vertex ℓ_1 is antipodal to r_1 and r_2.*

As a consequence of this lemma, r_1 is not antipodal to ℓ_2, so removing r_1 from consideration the recursive problem does not remove any antipodal pairs in C that must be checked.

▶ **Lemma 3.** *An optimal solution with pq as diameter is considered during the algorithm.*

Proof. Let C be an optimal solution, and let its triangulation be as described above. Also assume (by symmetry) that the first triangle in the sequence $t_1 = \triangle pr_1r_2$. Observe that C thus has: (i) pq as a diameter, (ii) r_1 as the ccw neighbor of p, (iii) r_2 as the ccw neighbor of r_1, (iv) ℓ_1 as the ccw neighbor of q, (v) k points inside it (for some $k \in \mathbb{N}$), and (vi) ℓ_1 antipodal with both r_1 and r_2 (by Lemma 2). Therefore, by definition of R, the area of C is $R[r_1, r_2, \ell_1, k]$. Symmetrically, it follows that if t_1 lies on the left it is a solution to an L subproblem. We now establish by induction that our definitions of R and L are correct.

If t_1 is the only triangle in C, it thus follows that we are in a base case, and $C = t_1$ indeed has area $R[r_1, r_2, \ell_1, k] = A(p, r_1, r_2) = A(p, r_1, q)$ and contains $k = 3 + K(p, r_1, r_2)$ points.

If there exists a triangle t_2, and this triangle also lies on the right, then it directly follows from the induction hypothesis that the area of C without triangle t_1 is $R[r_2, r_3, \ell_1, k - K(p, r_1, r_2) - 1]$, and thus our definition of $R[r_1, r_2, \ell_1, k]$ correctly computes the area of C. Symmetrically, if t_2 lies on the left, the induction hypothesis gives us that the area of C without triangle t_1 is $L[\ell_1, \ell_2, r_2, k - K(p, r_1, r_2) - 1]$. It again follows that our definition of $R[r_1, r_2, \ell_1, k]$ is correct. ◀

▶ **Lemma 4.** *No solution is determined that is not convex or where p, q is not a diameter.*

The two tables R and L require $O(n^3k)$ storage, each entry is filled in $O(n)$ time if we know the point count in each triangle, and there are $O(n^2)$ choices for p, q. Hence we obtain:

▶ **Theorem 5.** *Given a set of n points in the plane, a diameter bound D, and an integer $k \geq 3$, a smallest convex polygon that has diameter at most D and contains k points inside or on the boundary can be computed in $O(n^6k)$ time and $O(n^3k)$ space.*

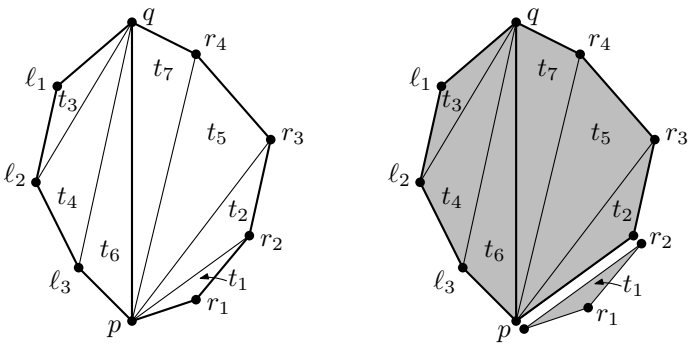


Figure 5 A solution C seen as a decomposition into triangles that are consecutively removed.

Proof. The algorithm that attains this bound has already been described, except for determining how many points of P lie in any triangle, as used in the recursion. We can do an easy preprocessing step where we determine for each of the $O(n^2)$ edges how many points of P lie vertically below that edge. This allows us to retrieve the point count inside a triangle in constant time by using the three edges of the triangle, and adding or subtracting these counts. This takes $O(n^3)$ time, which is clearly dominated by the time to run the dynamic programming steps. ◀

The final algorithm from [5] can fill in table entries in constant time. Unfortunately, we were unable to use a similar approach (e.g. rotating around r filling in the R-entries for every point r') as the table entry depends on a second point ℓ' on the left to guarantee the bound on the diameter. It seems hard to maintain the appropriate point ℓ' as r' varies. Also, rotating around r and ℓ simultaneously did not seem to work.

By running the algorithm with some value k, we also determine the smallest solutions for all values $3, \ldots, k$. This allows us to assume an maximum diameter D and maximum area A, and determine the convex polygon with most points inside within the same time bound. Start with $k = 4$, run the algorithm above, and if the area is smaller than A, double k and repeat. If the area is larger than A for the current k, we can look up the largest point count for which the area is at most A.

▶ **Corollary 6.** *Given a set P of n points in the plane, a diameter bound D and an area bound A, a convex polygon with diameter at most D and area at most A that contains the maximum number of points of P can be computed in $O(n^6k)$ time and $O(n^3k)$ space, where k is the number of points in the optimal solution.*

3 Experimental setup

The second part of this paper contains the experimental part of the research. We answer a few questions relating to the resources used by the algorithm from [5], referred to as the area-only (A) algorithm, and the new version that also enforces an upper bound on the diameter, referred to as the area-diameter (AD) algorithm. We use AD_d to refer to the AD algorithm with maximum diameter set to d. When testing on data from the application itself, we also compare with the algorithm from [9], referred to as the area selection (AS) algorithm like in their paper. We are interested in answering the following questions:

- How efficient in terms of running time and storage requirements are the A and AD_d algorithms? How does the AD_d algorithm's efficiency depend on the diameter?
- What diameters are realized by the A algorithm in the optimal solutions?
- How do the number of enclosed points compare in the optimal solutions of the A and AD_d algorithms?
- What are the found results in terms of area, diameter, and point count, of the three algorithms, now including the AS algorithm [9], on real-world data of the application that motivated this research?

Note that the first question relates to the resources used by the algorithms, whereas the other three questions relate to the output features. To answer these questions, we implemented the algorithms and selected data to be used. In the next subsections we discuss the data and the implementation aspects. The next section discusses the results and answers the research questions.

Table 1 The ten real-world point sets used in our experiments. We report the data set index (I), number of points n, and Average Nearest Neighbor distance (ANN) in mm with standard deviation.

I	n	ANN	I	n	ANN
0	699	0.21 ± 0.14	5	474	0.30 ± 0.23
1	623	0.32 ± 0.22	6	412	0.35 ± 0.23
2	594	0.28 ± 0.19	7	382	0.34 ± 0.23
3	526	0.29 ± 0.21	8	377	0.33 ± 0.27
4	492	0.32 ± 0.25	9	362	0.48 ± 0.34

3.1 Data and settings

We use three types of data for the experiments:

- Uniformly distributed point sets inside a square of size 20×20 mm^2 of increasing cardinalities $100, 110, \ldots, 200$.
- Normally distributed point sets of 100 points centered inside a square of size 20×20 mm^2 of increasing standard deviations ranging from $\sigma = 0.5, 1.0, 1.5, \ldots, 6.5$ mm. Points outside the square are discarded and re-generated.
- Medical data used in [9]. We selected ten data sets with up to 700 points, in particular the ten most populated sets.

We will always search for an area of at most 4 mm^2.[1] For the AD algorithm, we examine maximum diameters of 2, 3, 4, 5, and 6 mm. Note that for a diameter of 2 mm, the maximum area possible is π mm^2, so in this setting, we cannot "use" the full allowed area of 4 mm^2.

Since the first two types of data are randomly generated, we generated 100 point sets in each setting and report averages over these 100 runs.

The real-world data is obtained by preprocessing the data set provided by the Department of Digital Pathology at UMC Utrecht, containing 1982 point sets representing the mitotic figures identified in Whole-Slide-Images (WSIs) from actual patients [9]. The identification was carried out by the classification model introduced in [11], which produced 2-dimensional points with detection accuracies, since the points have a different likeliness of being a mitotic cell. We apply a preprocessing step to filter out any point with detection accuracy below 0.86 and convert the coordinates from pixels to millimeters. We consider the top 10 most populated point sets for our experiments. Table 1 provides an overview of this data. The average nearest neighbor distances provide an indication of how clustered the points are.

Both the synthetic generation and preprocessing of real-world data are fully automated in a Python script which we provide in a repository.

3.2 Implementation

We implemented the A algorithm from [5] and the AD algorithm in C++[2]. We implemented all other necessary data structures and algorithms that are not already in the std ourselves. The implementations do not use parallelism and run in a single thread. The experiments were run on a MacBook Pro using an Apple M1 Pro (3228 MHz) and 16GB (6400 MT/s

[1] While [9] lists 2 mm^2 as the desired area of the region, personal communication with the authors of the paper showed that they use 4 mm^2 currently in the application.

[2] Our code is available at `https://github.com/gianmarcopicarella/area-diameter-algorithm`.

LPDDR5) RAM. It runs macOS Sonoma 14.2.1, and we compiled it for 64 bits using clang 19.1.6 with the -O3 option. We used the Google Benchmark library to set up our experiments and perform data collection. The data is then post-processed in Python with several scripts that we provide in our repository. Regarding the Area Selection method (AS), we used the Python implementation made available by [9]. We use this implementation mostly to compare output, not to compare resources, although we report times.

The AS algorithm from [9] first partitions the data in patches of 3×3 mm, allowing overlap so that solutions across the boundary of a patch are not missed. For each patch, if it contains at most 25 points, a brute-force method determines the optimum. Otherwise, a heuristic is used that takes the convex hull of a patch and incrementally discards points from the convex hull, until the criteria are met. More precisely, it considers each convex hull point p, analyzes the decrease in area and cardinality if it were discarded, but also if it were replaced by any point inside the triangle formed by p and its two neighbors (details in [9]).

The A and AD algorithms use a few optimizations. First, we preprocess for quickly determining the point count in triangles, so that the dynamic program can avoid spending more than constant time. We use only $O(n^2)$ time for preprocessing and storage by computing the number of points below each of the $\binom{n}{2}$ possible edges. To represent all table entries of the AD algorithm, we used hashing, since many table entries are not used. In particular, we use an array with k hash tables for R and for L: to retrieve $R[r, r', \ell, k']$ we inspect the k'-th hash table and use the indices of r, r' and ℓ as the key, and similar for L. In the A algorithm, most table entries are used, so the A algorithm does not benefit as much from hashing. Hence it is not used. Finally, both the A and AD algorithms use an early exit strategy. For the AD algorithm point pairs are treated in decreasing order of number of points in the lune. If a solution of cardinality k is known, then all point pairs whose lune contains fewer points can be skipped. For the A algorithm, the early exit strategy is much less strong, but we can still skip processing points p as bottommost point if there are not enough points above it to improve the current best solution. Any preprocessing time is included in the timings. The implementations handle degenerate cases like collinear points properly.

4 Experimental results

We first present and discuss the results on synthetic data, starting with the resources needed by the A and AD algorithms. We then discuss the output results. Finally, we discuss the results on real-world data.

Resources. In Figure 6, the running time for synthetic point sets generated with uniform and normal (Gaussian) distributions is displayed. We see the AD algorithm makes very good use of the bounded diameter when compared with the A algorithm. Despite requiring $O(n^6k)$ time worst case, the pruning possible by bounding the diameter in the AD algorithm easily outweighs the much better $O(n^3k)$ running time of the A algorithm. This is true for the point set sizes tested. The situation is different for the normal distribution. When $\sigma \leq 1.5$, the 100 points are clustered around the center and the AD algorithm must try many more options, leading to a worse performance than the A algorithm. Both for the uniform and the Gaussian distribution, we can clearly see that the AD algorithm is more efficient for smaller diameters than for larger diameters, which is expected. In the full version [8] we also show test results on larger point sets, up to 500 points for the uniform distribution, but due to computational demands we averaged over only 5 repetitions. The patterns appear the same; the AD algorithms are still more efficient.

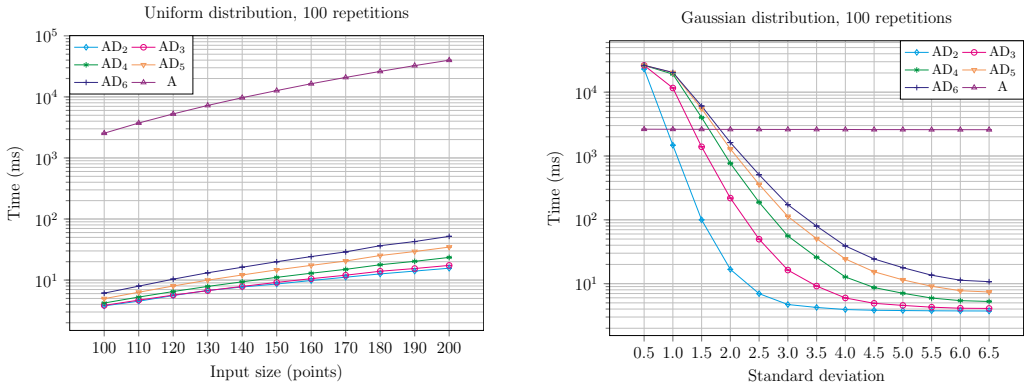


Figure 6 Left: average running time over 100 uniformly distributed data sets of sizes 100 to 200. Right: average running time over 100 normally distributed data sets of 100 points with varying standard deviations.

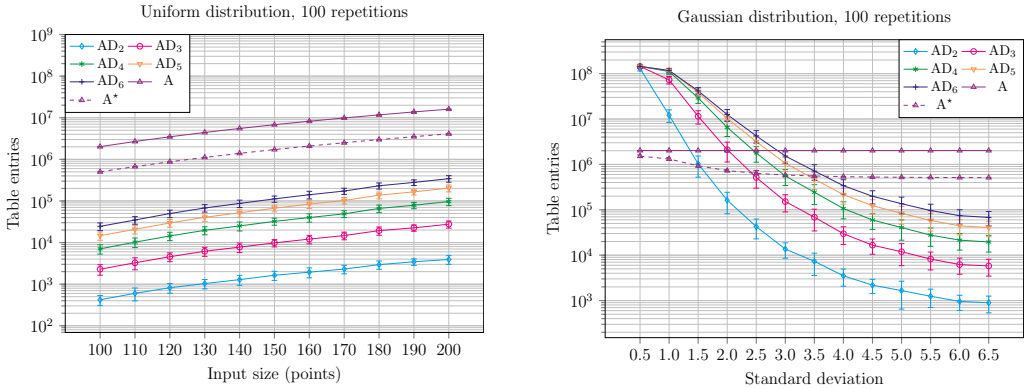


Figure 7 Left: average number of entries over 100 uniformly distributed data sets of sizes 100 to 200. Right: average number of entries over 100 normally distributed data sets of 100 points with varying standard deviations.

To measure the memory usage of the algorithms we consider the number of table entries that are used by the algorithms. See Figure 7. The results are similar to those of the running times. In the basic implementation of the A algorithm, a fixed size table is allocated, and the number of entries does not depend on the specific point set, only on its size. To also let the A algorithm benefit from fewer entries, we consider a version of the A algorithm that stores repeated entries found during the rotational sweep around q just once. Whenever we use the same T_q^* to fill $T[q, r, k]$, we can store it as a sequence of values that are all T_q^*, at the expense of extra query time for table entries. This is shown by the dashed purple curve. We would use a factor 3–4 fewer entries, except for the Gaussian distribution with lower values of σ. We also studied the memory usage instead of the number of entries in the tables, but since this showed the same patterns, we left out these graphs.

Output. We can compare the results of running the algorithms on cardinality (point count), area, and diameter. Since all algorithms are optimal for their settings, it is clear that the cardinality for the A algorithm is highest, followed by the cardinalities of the AD algorithm by decreasing diameter.

The output numbers cardinality, area, and diameter, averaged, are shown in Figures 8–10 for the uniform and normal distributions.

Indeed the A algorithm finds larger cardinality solutions; the differences between different diameter bounds are also clear. It is perhaps surprising that the highest cardinality solutions are typically not compact, but rather elongated. This is confirmed in Figure 10, where the A algorithm always finds solutions with diameters 8–18 mm. With an area close to 4 mm^2, these regions must be elongated. Further experiments [8] confirm these findings up to 500 points, albeit based on just 5 repetitions.

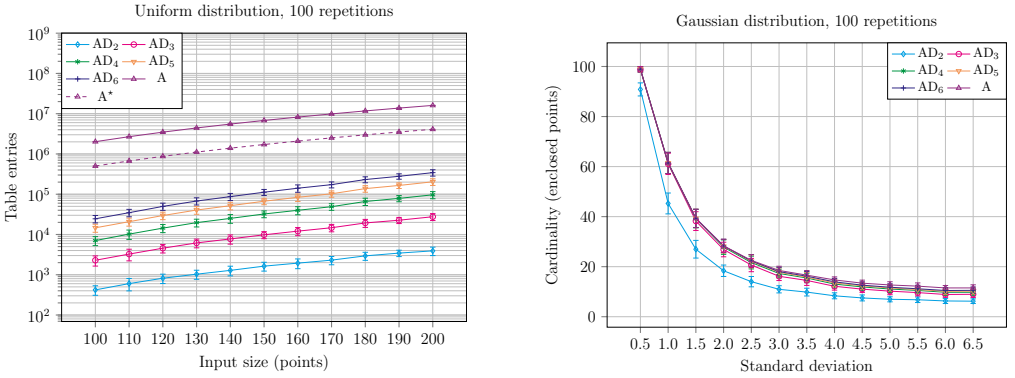


Figure 8 Left: average cardinality over 100 uniformly distributed data sets of sizes 100 to 200. Right: average cardinality over 100 normally distributed data sets of 100 points with varying standard deviations.

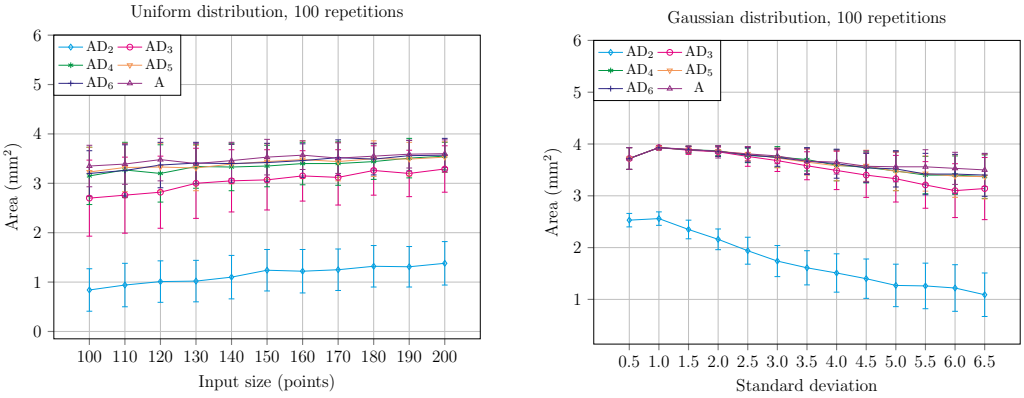


Figure 9 Left: average area over 100 uniformly distributed data sets of sizes 100 to 200. Right: average area over 100 normally distributed data sets of 100 points with varying standard deviations.

Real-world data. We consider the output of the algorithms on the data that inspired this research, where we include the area selection (AS) algorithm from [9]. That algorithm uses patches for efficiency reasons, and to be able to run an exact, exponential-time algorithm on patches that contain few points. Interestingly, patches are also a way to limit the diameter that can be found by the A algorithm, namely at most the diameter of a patch. Since patches are 3×3 mm, the A algorithm and the AS algorithm may find regions with diameter up to 4.24 mm. We use a diameter of 4 mm in the AD algorithm, which does not use patches.

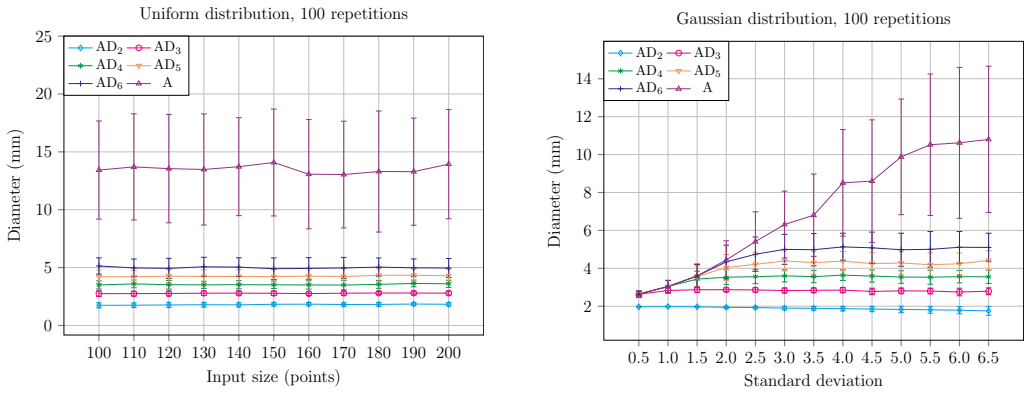


Figure 10 Left: average diameter over 100 uniformly distributed data sets of sizes 100 to 200. Right: average diameter over 100 normally distributed data sets of 100 points with varying standard deviations.

Table 2 The runtime (seconds) of the Area-Diameter (AD) algorithm, the area selection (AS) algorithm and the A algorithm applied to the real-world data summarized in Table 1. AS is run for step sizes $s = 1.98$ and $s = 0.5$; A is run within each patch generated with step size $s = 0.5$.

I	AD_4	AS ($s = 1.98$)	AS ($s = 0.5$)	A ($s = 0.5$)
0	494.19	**5.24**	72.83	13.7
1	14.7	17.15	274.82	**2.84**
2	29.38	19.89	218.36	**3.66**
3	24.43	19.12	263.07	**3.17**
4	8.83	13.83	229.33	**2.08**
5	21.76	9.16	141.13	**4.51**
6	2.08	9.32	161.07	**1.63**
7	6.19	13.66	174.71	**1.82**
8	37.18	9.66	77.41	**2.58**
9	**0.19**	4.83	52.02	1.47

Patches are shifted with overlap to cover the whole region. We show results of the AS algorithm with a rather large step size of 1.98 mm, and results of the A and AS algorithms with a step size of 0.5 mm.[3]

The results are shown in Tables 2 and 3. In the ten examples, the four algorithms usually find the same rough region, but seldom the exact same convex solution. One example is shown in Figure 11; the full version [8] shows all ten data sets. In two cases, the AS algorithm with $s = 1.98$ finds a very different region, confirming that that step size of a moving window matters and must be set small enough. The results of the two AS methods show how many more points can be enclosed by using a smaller step size. The results of AS and A with step size $s = 0.5$ show how well the AS heuristic works compared to the optimal algorithm A (optimal per patch). The timings show that A is faster than AS, while being optimal per patch as well. While AD_4 finds better solutions without patches, it is clearly slower.

[3] The step size of 1.98 mm is based on the actual usage of the algorithm currently by the authors of [9].

Table 3 Comparison of the results obtained by applying the area-diameter algorithm (AD), the area selection method (AS) and the A algorithm to all patches of the point sets summarized in Table 1. AS is run for step sizes $s = 1.98$ and $s = 0.5$; A is run for step size $s = 0.5$. All the values for area and diameter are expressed in mm^2 and mm respectively. Best counts are shown bold; values k and $\hat{k}$ use different diameter bounds and are not directly comparable.

I	$\mathbf{AD_4}$			$\mathbf{AS}$ $(s = 1.98)$			$\mathbf{AS}$ $(s = 0.5)$			$\mathbf{A}$ $(s = 0.5)$		
	k	area	dia.	$\hat{k}$	area	dia.	$\hat{k}$	area	dia.	$\hat{k}$	area	dia.
0	**76**	3.99	3.15	68	3.42	2.89	70	3.74	2.88	74	3.9	3.07
1	**42**	3.83	3.99	36	3.84	2.74	39	3.97	2.9	40	3.94	2.87
2	**39**	3.93	2.72	33	3.99	2.85	**39**	3.98	2.66	**39**	3.93	2.72
3	**42**	3.94	3.1	34	3.99	2.71	39	3.97	3.15	40	3.88	3.1
4	**42**	3.88	2.99	38	3.87	2.82	40	3.99	2.83	**42**	3.98	2.88
5	**44**	3.88	3.7	37	3.82	3.31	39	3.76	3.17	39	3.82	3.01
6	**31**	3.98	3.85	27	3.72	2.73	28	4.0	3.45	28	3.94	3.38
7	**34**	3.89	3.22	30	3.94	2.65	32	3.92	3.22	**34**	3.92	3.22
8	**61**	3.91	3.94	57	3.86	2.67	59	3.97	2.72	60	3.92	3.01
9	**28**	3.86	3.49	23	3.97	3.41	27	3.97	2.74	27	3.97	2.74

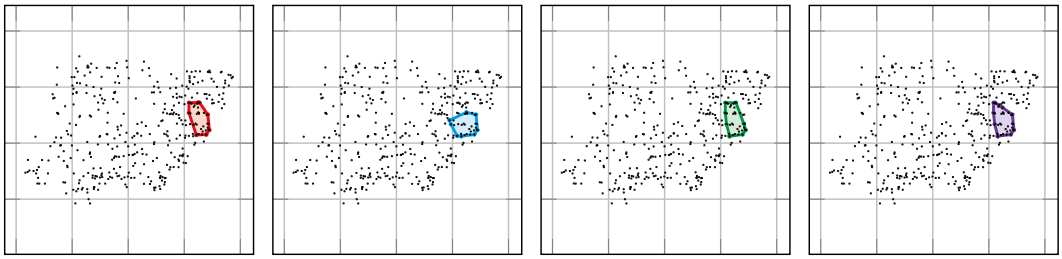


Figure 11 Convex areas found by AD (a), AS using $s = 1.98$ (b), AS using $s = 0.5$ (c) and A using $s = 0.5$ (d) for point set 7. The line spacing is just for reference and is set to 5 mm.

5 Conclusions and further research

We have shown that maximum cardinality convex subsets with bounded area and diameter can be computed in polynomial time by a dynamic programming method that embeds rotating calipers. The algorithm runs in $O(n^6k)$ time, a cubic factor higher than the algorithm that does not consider the diameter.

The original application worked with a bounded area and aspect ratio, which is similar but not quite the same as area and diameter. It is unclear whether a polynomial-time algorithm exists that computes maximum cardinality subsets with bounded area and aspect ratio, or bounded area and perimeter. These are open theoretical questions for future research.

We have also shown that the new algorithm may well be faster than the algorithm that ignores diameter in practical situations, because optimal solutions without a diameter constraint are often very elongated. With a bounded diameter, we can often prune significantly. Of course this depends heavily on the specific data set. We have analyzed this experimentally for uniformly distributed data and Gaussian distributed data.

For the data from the application, we have shown that we can compute an optimal solution in the practical situation within a reasonable amount of time, where the problem is phrased as a well-defined optimization problem and the solution need not use patches.

References

1 Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. *Journal of Algorithms*, 12(1):38–56, 1991. doi:10.1016/0196-6774(91)90022-Q.

2 Esther M. Arkin, Samir Khuller, and Joseph S. Mitchell. Geometric knapsack problems. *Algorithmica*, 10(5):399–427, 1993. doi:10.1007/BF01769706.

3 Timothy M Chan and Sariel Har-Peled. Smallest k-enclosing rectangle revisited. *Discrete & Computational Geometry*, 66(2):769–791, 2021. doi:10.1007/S00454-020-00239-3.

4 David Eppstein. New algorithms for minimum area k-gons. In *Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '92, pages 83–88, 1992. URL: http://dl.acm.org/citation.cfm?id=139404.139422.

5 David Eppstein, Mark Overmars, Günter Rote, and Gerhard Woeginger. Finding minimum area k-gons. *Discrete & Computational Geometry*, 7(1):45–58, 1992. doi:10.1007/BF02187823.

6 Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 226–331, 1996.

7 Sariel Har-Peled and Soham Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41:147–157, 2005. doi:10.1007/S00453-004-1123-0.

8 Gianmarco Picarella, Marc van Kreveld, Frank Staals, and Sjoerd de Vries. Computing largest subsets of points whose convex hulls have bounded area and diameter. *CoRR*, abs/2507.04933, 2025. arXiv:2507.04933.

9 Nikolas Stathonikos, Marc Aubreville, Sjoerd de Vries, Frauke Wilm, Christof A Bertram, Mitko Veta, and Paul J van Diest. Breast cancer survival prediction using an automated mitosis detection pipeline. *Journal of Pathology: Clinical Research*, 10(6), 2024. doi:10.1002/2056-4538.70008.

10 Godfried T. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE MELECON*, volume 83, page A10, 1983.

11 Frauke Wilm, Christian Marzahl, Katharina Breininger, and Marc Aubreville. Domain adversarial retinanet as a reference algorithm for the mitosis domain generalization challenge. In Marc Aubreville, David Zimmerer, and Mattias Heinrich, editors, *Biomedical Image Registration, Domain Generalisation and Out-of-Distribution Analysis*, pages 5–13, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-030-97281-3_1.

Fréchet Distance in Unweighted Planar Graphs

Ivor van der Hoog
IT University of Copenhagen, Denmark

Thijs van der Horst
Utrecht University, The Netherlands
TU Eindhoven, The Netherlands

Eva Rotenberg
IT University of Copenhagen, Denmark

Lasse Wulf
IT University of Copenhagen, Denmark

Abstract

The Fréchet distance is a distance measure between trajectories in $\mathbb{R}^d$ or walks in a graph G. Given constant-time shortest path queries, the Discrete Fréchet distance $\mathrm{D_G}(P, Q)$ between two walks P and Q can be computed in $O(|P| \cdot |Q|)$ time using a dynamic program. Driemel, van der Hoog, and Rotenberg [SoCG'22] show that for weighted planar graphs this approach is likely tight, as there can be no strongly-subquadratic algorithm to compute a 1.01-approximation of $\mathrm{D_G}(P, Q)$ unless the Orthogonal Vector Hypothesis (OVH) fails.

Such quadratic-time conditional lower bounds are common to many Fréchet distance variants. However, they can be circumvented by assuming that the input comes from some well-behaved class: There exist $(1+\varepsilon)$-approximations, both in weighted graphs and in $\mathbb{R}^d$, that take near-linear time for c-packed or κ-straight walks in the graph. In $\mathbb{R}^d$ there also exists a near-linear time algorithm to compute the Fréchet distance whenever all input edges are long compared to the distance.

We consider computing the Fréchet distance in unweighted planar graphs. We show that there exist no strongly-subquadratic 1.25-approximations of the discrete Fréchet distance between two disjoint simple paths in an unweighted planar graph in strongly subquadratic time, unless OVH fails. This improves the previous lower bound, both in terms of generality and approximation factor. We subsequently show that adding graph structure circumvents this lower bound: If the graph is a regular tiling with unit-weighted edges, then there exists an $\tilde{O}((|P| + |Q|)^{1.5})$-time algorithm to compute $\mathrm{D_G}(P, Q)$. Our result has natural implications in the plane, as it allows us to define a new class of well-behaved curves that facilitate $(1+\varepsilon)$-approximations of their discrete Fréchet distance in subquadratic time.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Fréchet distance, planar graphs, lower bounds, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.24

Related Version *Full Version*: `https://arxiv.org/abs/2504.17342`

Funding This work was supported by the Independent Research Fund Denmark grant 2020-2023 (9131-00044B) "Dynamic Network Analysis", the Carlsberg Foundation Young Researcher Fellowship CF21-0302 "Graph Algorithms with Geometric Applications", the VILLUM Foundation grant (VIL37507) "Efficient Recomputations for Changeful Problems", and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987.



© Ivor van der Hoog, Thijs van der Horst, Eva Rotenberg, and Lasse Wulf;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

The Fréchet distance is a widely used metric for measuring the similarity between trajectories. It is often illustrated through a metaphor: imagine a person walking along one trajectory and their dog following another, connected by a leash. The Fréchet distance is the minimum possible leash length over all synchronized traversals of both trajectories.

This metric has numerous applications, particularly in movement data analysis [10, 18, 24, 15, 5, 13, 25, 33]. It is versatile, having been applied to handwriting recognition [29], coastlines [26], geometric shapes in geographic information systems [16], and trajectories of moving objects such as vehicles, animals, and athletes [28, 30, 6, 13]. We consider the *discrete* Fréchet distance, where trajectories are modeled as sequences of discrete points.

Alt and Godau [2] were the first to analyse the Fréchet distance from a computational perspective. They considered trajectories as polygonal curves in $\mathbb{R}^d$ with n and m vertices, and compute the continuous Fréchet distance in $O(mn \log(n+m))$ time. Later, Buchin, Buchin, Meulemans and Mulzer [11] introduced a faster randomized algorithm, achieving a running time of $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ on a real-valued pointer machine and $O(n^2 \log\log n)$ on a word RAM. For the discrete Fréchet distance, Eiter and Mannila [20] presented an $O(nm)$ time algorithm under constant-time distance computations. Agarwal, Avraham, Kaplan, and Sharir [1] later improved this to $O(nm(\log\log nm)/\log nm)$ in the word RAM.

Driemel, van der Hoog, and Rotenberg [19] initiate the study of Fréchet distance in graphs. They consider as input a (weighted) graph G where a trajectory is a walk in the graph. The distance between two vertices is the length of their shortest path. Given a constant-time distance oracle, the discrete Fréchet distance $\mathrm{D_G}(P, Q)$ between any two walks in G can then be computed in $O(nm)$ time using the algorithm by Eiter and Mannila [20].

Conditional Lower Bounds. Several conditional lower bounds exist for computing the Fréchet distance or its constant-factor approximations. These results rely on complexity assumptions such as the Orthogonal Vector Hypothesis (OVH) and the Strong Exponential Time Hypothesis (SETH) [32]. Bringmann [7] established that no algorithm can compute the (discrete or continuous) Fréchet distance between two polygonal curves of n vertices in $O(n^{2-\delta})$ time for any $\delta > 0$, unless OVH fails. The same lower bound holds for small constant-factor approximations. Bringmann's proof originally involved self-intersecting curves in the plane, but later work by Bringmann and Mulzer [9] extended the result to intersecting curves in $\mathbb{R}^1$. Buchin, Ophelders, and Speckmann [12] further demonstrated that the same lower bound applies for any better-than-3-approximation algorithm for pairwise disjoint planar curves in $\mathbb{R}^2$, or intersecting curves in $\mathbb{R}^1$.

Bringmann's lower bound [7] holds also for unbalanced inputs: given two curves with n and m vertices in the plane, no algorithm can compute the Fréchet distance in $O((nm)^{1-\delta})$ time assuming OVH. Driemel, van der Hoog and Rotenberg [19] give a similar lower bound for paths in a weighted planar graph. They prove that, unless OVH fails, no $O((nm)^{1-\delta})$-time algorithm can 1.01-approximate the discrete Fréchet distance between arbitrary paths in a weighted planar graph – even if the ratio between smallest and highest weight is bounded by a constant. This lower bound applies only to weighted graphs and does therefore not exclude the existence of a fast algorithm on unweighted planar graphs. A closer inspection of their argument reveals that their proof cannot simply be adapted to the unweighted case by subdividing long edges often enough. In this case, new vertices get introduced to the graph and their arguments break down.

Avoiding lower bounds through well-behaved curves. These lower bounds can be circumvented if the input is well-behaved. Driemel, Har-peled and Wenk [17] consider three classes of curves in $\mathbb{R}^d$ which are well-behaved as long as their corresponding behavioural parameter is low. These are c-packed curves [17, 8], ϕ-low density curves [17], and κ-bounded curves [3, 4, 17]. They show algorithms to compute a $(1+\varepsilon)$-approximation between two well-behaved curves whose running time is near-linear in n and m. Gudmundsson, Mirzanezhad, Mohades, and Wenk consider the special case where all edges of the input curves are long with respect to their Fréchet distance [23]. In this case, the Fréchet distance can be computed in $O((n+m)\log(n+m))$ time [23]. Driemel, van der Hoog, and Rotenberg [19] show that if the input are walks in a graph G and one of the walks is restricted to a c-packed or κ-straight path, then there exists $(1+\varepsilon)$-approximations that take near-linear time.

Contribution. Driemel, van der Hoog, and Rotenberg [19] ruled out a strongly subquadratic algorithm for a 1.01-approximation between paths in a weighted graph, while the construction by Buchin, Ophelders, and Speckmann [12] rules out a strongly subquadratic algorithm for a 3-approximation between walks in an unweighted planar graph. We consider the discrete Fréchet distance between paths in an unweighted planar graph. We prove (Theorem 5) that no strongly-subquadratic 1.25-approximation algorithm exists unless OVH fails, thereby improving both the generality (and also the approximation factor for planar graphs).

Next, we prove that adding graph structure circumvents this lower bound. If G is an unweighted regular tiling of the plane, then there exists an $\tilde{O}((n+m)^{1.5})$-time algorithm to compute the Fréchet distance $\mathrm{D_G}(P,Q)$, between curves of length n and m. Our results have natural implications in the plane, for a special class of well-behaved curves ((ε,δ)-curves, to be defined). For those, we can compute the Fréchet distance under the L_1 metric (respectively, approximate it for any L_c metric) in $\tilde{O}(\frac{\sqrt{\delta}}{\varepsilon}(n+m)^{1.5})$ time (respectively, $\tilde{O}(\frac{\delta}{\varepsilon^2}(n+m)^{1.5})$ time). For completeness, we explain the continuous analogy to (ε,δ)-curves in the full version of this paper. We also compare this new curve class to existing well-behaved curve classes in the full version.

2 Preliminaries

Let G be an unweighted graph where $V(G)$ is its vertex set. We define $P=(p_1,\ldots,p_n)$ and $Q=(q_1,\ldots,q_m)$ as two paths in G. A path cannot visit a vertex twice. For integers $i,i'\in[n]$ with $i\leq i'$ we denote by $P[i:i']$ the subpath of P from p_i to $p_{i'}$. We define the discrete Fréchet distance $\mathrm{D_G}(P,Q)$ between P and Q as follows [19]: A sequence $(a_i,b_i)_{i\in[k]}$ of pairs of indices is a *monotone walk* if for all $i\in[k-1]$ we have $a_{i+1}\in\{a_i,a_i+1\}$ and $b_{i+1}\in\{b_i,b_i+1\}$. Let $\mathbb{F}_{n,m}$ be the set of all monotone walks from $(1,1)$ to (n,m). The cost of $F\in\mathbb{F}_{n,m}$ is the maximum of $\mathrm{dist}(p_i,q_j)$ over $(i,j)\in F$, where dist() denotes the length of the shortest path between vertices. $\mathrm{D_G}(P,Q)$ is the minimum cost of a walk in $\mathbb{F}_{n,m}$.

$$\mathrm{D_G}(P,Q)=\min_{F\in\mathbb{F}_{n,m}}\mathrm{cost}(F)=\min_{F\in\mathbb{F}_{n,m}}\max_{(i,j)\in F}\mathrm{dist}(p_i,q_j).$$

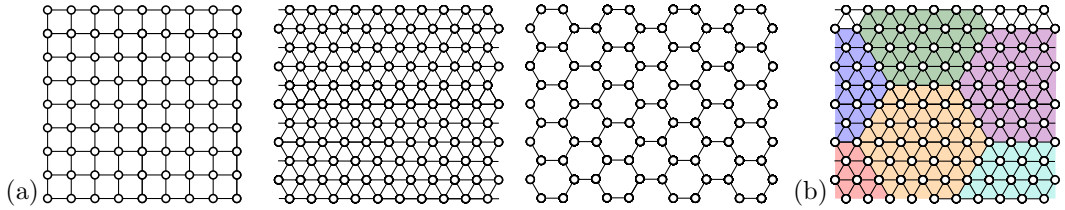


Figure 1 (a) The three regular plane tilings. (b) A hexagonal super-tiling of a triangular tiling.

Decision variant. We focus on the decision variant where the input includes some $\Delta \geq 0$ and the goal is to output whether $\mathrm{D_G}(P, Q) \leq \Delta$. The Δ-*free space matrix* M_Δ is the n by m matrix where $M_\Delta[i, j]$ is 0 if $\mathrm{dist}(p_i, q_j) \leq \Delta$ and 1 otherwise. We follow geometric convention, where i denotes the column of M_Δ (column i corresponds to some $p_i \in P$. Column entries with value 0 correspond to $q_j \in Q$ with $\mathrm{dist}(p_i, q_j) \leq \Delta$). Eiter and Mannila [20] define a directed graph over M_Δ where there is an edge from $M_\Delta[a, b]$ to $M_\Delta[c, d]$ if and only if $c \in \{a, a+1\}$, $d \in \{b, b+1\}$ and $M_\Delta[a, b] = M_\Delta[c, d] = 0$. They prove that $\mathrm{D_G}(P, Q) \leq \Delta$ if and only if there exists a directed path from $(1, 1)$ to (n, m) in this graph.

Orthogonal vectors. Our conditional lower bound reduces from the Orthogonal Vector Hypothesis. The input consists of two ordered sets of d-dimensional Boolean vectors U, W of sizes $|U| = n$ and $|W| = m$. We denote for $u \in U$ by $u_i \in \{0, 1\}$ its i'th component. The *OV problem* is to find out if some vector of U is orthogonal to some vector of W.

▶ **Definition 1** (Williams [32]). *The orthogonal vector hypothesis (OVH) states that for all $\delta > 0$, there exists constants ω and $1 > \gamma > 0$ such that the OV problem for vectors of dimension $d = \omega \log n$ and $m = n^\gamma$ cannot be solved in $O((nm)^{1-\delta})$ time.*

We consider each vector in U (or W) as a string of d bits. We denote by $\mathrm{str}(U)$ and $\mathrm{str}(W)$ the concatenation of all strings in U and W, respectively. A *substring* of some string $s_1 s_2 \ldots s_k$ is the sequence $s_a s_{a+1} \ldots s_{b-1} s_b$ for some $1 \leq a \leq b \leq k$.

Regular tilings. A *regular tiling* T is an infinite plane graph where each face is a regular polygon and each face has the same degree. There exist exactly three regular tilings [22]: the triangular, square, and hexagonal tiling (Figure 1). Without loss of generality, we assume that our tilings are axis-aligned. A face F of T is an open region bounded by the edges of T, we denote by $\overline{F}$ its closure. The *edge length* of T is the length of any edge. A tiling is *unit* if its edges have unit length. A path in T is a path in its corresponding graph. We always denote by T a unit tiling where $(0, 0)$ is a vertex of T. For any pair of vertices p, q of T, the term $\mathrm{dist}(p, q)$ denotes the distance in the graph T. We assume that we can compute $\mathrm{dist}(p, q)$ in constant time. Our algorithm considers what we will call a *super-tiling* of T and a natural partition of the edges of P and Q into subpaths.

▶ **Definition 2** (Figure 1 (b)). *Given a regular tiling T of the plane, a* super-tiling $\mathbb{T}$ *of T is a regular tiling of the plane whose vertex set is a subset of the vertex set $V(T)$. The* boundary vertices $B(\mathbb{T})$ *are the vertices of T that lie on the boundary of some face of $\mathbb{T}$.*

▶ **Definition 3.** *Consider a regular tiling T, some super-tiling $\mathbb{T}$ and a path P in T, we define the* breakpoints $B(P, \mathbb{T})$ *as all vertices in $P \cap B(\mathbb{T})$.*

Given a path P in a tiling T and a super-tiling $\mathbb{T}$ of T, we want to consider how $\mathbb{T}$ splits P into smaller snippets, each snippet "belonging" to a face of $\mathbb{T}$. More formally:

▶ **Definition 4** (Figure 2). *Consider a regular tiling T and a super-tiling $\mathbb{T}$ of T. Given a path $P = (p_1, p_2, \ldots, p_n)$ in T, let $\mathbb{I} := \{i \in [n] \mid p_i \in B(P, \mathbb{T}) \text{ or } i \in \{1, n\}\}$. We define the* induced subpaths $S(P, \mathbb{T})$ *as all subpaths $P[x : y]$, for consecutive $x, y \in \mathbb{I}$. Note that for each induced subpath, there exists a (not necessarily unique) face F in $\mathbb{T}$ whose closure $\bar{F}$ contains it; we say the induced subpath is* assigned *to one such face (breaking ties arbitrarily).*

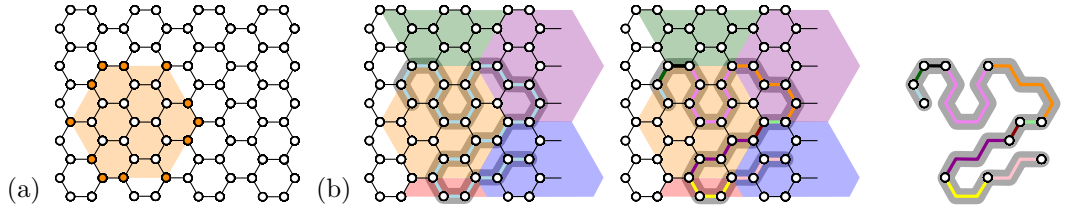


Figure 2 (a) For a face F in the super-tiling $\mathbb{T}$, we show the set $B(\mathbb{T}) \cap \overline{F}$. (b) Given a path P, we create a sequence of subpaths of P where consecutive subpaths overlap in a single vertex.

Submatrices of M_Δ. We show a subquadratic algorithm to decide whether $\mathrm{D_T}(P,Q) \leq \Delta$ if P and Q are paths in a tiling T. We compute a monotone walk from $(1,1)$ to (n,m) in M_Δ by efficiently propagating reachability information across *submatrices* of M_Δ. For integers $i, i' \in [n]$ and $j, j' \in [m]$ with $i \leq i'$ and $j \leq j'$ we denote by $M_\Delta[i:i', j:j']$ the submatrix of M_Δ that consist of all entries $M_\Delta[x,y]$ for $(x,y) \in [i,i'] \times [j,j']$. We define the *bottom-left* facets of $M_\Delta[i:i', j:j']$ as all entries $\{M_\Delta[i,y] \mid y \in [j,j']\}$ and $\{M_\Delta[x,j] \mid x \in [i,i']\}$. The *top-right* facets are defined as the entries $\{M_\Delta[i',y] \mid y \in [j,j']\}$ and $\{M_\Delta[x,j'] \mid x \in [i,i']\}$.

(ε, δ)-curves. Finally, we introduce a new class of well-behaved curves in the plane. Intuitively, an (ε,δ)-curve, for small δ, is a curve composed of short edges that does not revisit the same region of the plane too frequently. Formally, fix a universal constant $\gamma \in O(1)$. Let P be a curve in the plane, and let B_ε denote the set of all balls (in Euclidean metric) of radius ε centred at the vertices of P. We say that P is an (ε,δ)-curve if all its edges have length at most γ, and any point in the plane lies in at most δ balls from B_ε.

For any given pair (P,ε), δ is determined. As ε decreases, so may δ. If δ remains large for small ε, then many vertices of P are densely clustered – i.e., effectively, the curve frequently revisits the same locations. We present an exact algorithm for the Discrete Fréchet distance under the L_1 metric with a linear dependency on $\frac{\sqrt{\delta}}{\varepsilon}$. We assume that the input specifies a desirable value of ε, and that this yields a favourable corresponding δ. Imposing that δ is constant amounts to requiring that the curve does not revisit any ε-ball more than a constant number of times – a restriction that somewhat resembles the well-studied c-packed curves [17, 19, 8]. In the full version of this paper, we discuss in detail how our new class relates to the prior well-behaved curve classes.

3 A lower bound for paths in an unweighted planar graph

Let G be an unweighted planar graph and P and Q be paths in G. We show a lower bound, conditioned on OVH, that excludes any strongly subquadratic algorithm to compute a 1.25-approximation of $\mathrm{D_G}(P,Q)$. Since we are space-restricted, we present our argument only in the full version of this paper. Here, we assume that the reader is familiar with how these lower bounds are typically constructed [7, 12, 19, 9] and only mention how our construction differs from prior works. We start with some OV instance (U,W). We build an unweighted planar graph G and two paths P and Q in G such that $\mathrm{D_G}(P,Q) \leq 4$ if and only if the OV instance is a yes-instance and $\mathrm{D_G}(P,Q) \geq 5$ otherwise. One novelty of our approach is a preprocessing step, where we transform (U,W) into an equivalent instance with additional properties:

▶ **Lemma 1.** *An instance $U', W' \subseteq \{0,1\}^{d'}$ of OV can be preprocessed in $O(d'(n+m))$ time, resulting in a new instance $U, W \subseteq \{0,1\}^d$ with $d \in O(d')$ such that:*

- *a yes-instance stays a yes-instance and a no-instance stays a no-instance,*

- *for all* $u \in U$, $u_1 = u_d = 0$ *and for all* $w \in W$, $w_1 = w_d = 1$,
- *if the instance is a no-instance, then* $\forall u \in U$ *the vector* u *is not only non-orthogonal to every* $w \in W$*, but even non-orthogonal to all consecutive length-*d *substrings of* $\mathrm{str}(W)$.

This preprocessing significantly simplifies our proofs and may be of independent interest. Previous lower bound constructions [7, 19, 12, 9] fix some $\Delta \in \mathbb{R}$. Given an OV-instance (U, W), they construct curves (P, Q) by constructing for each $u \in U$ a subcurve $f(u)$ and each $w \in W$ a subcurve $g(w)$ (see Figure 3). The construction has a very strong property: Consider a traversal of P and Q where the person and the dog remain within distance Δ of one another. If the person is in $f(u)$ for some $u \in U$ and the dog is in $g(w)$ for some $w \in W$ then neither the person or the dog can stand still if the other moves.

Requiring the planar graph to be unweighted significantly restricts the freedom we have in engineering the pairwise distance between vertices. We are unable to recreate an equally strong property and instead obtain a much weaker statement: If the person is traversing the subpath $f(u)$ for some $u \in U$ and the dog is traversing $g(w)$ for some $w \in W$ and the person moves at least two steps then the dog has to move at least one. We rely upon our preprocessing to then still argue that there exists a traversal of P and Q such that the leash length is at most 4 if and only if there is an orthogonal pair $(u, w) \in U \times W$.

▶ **Theorem 5.** *Let* P *and* Q *be paths in an unweighted planar graph. Let* $|P| = n$ *and* $|Q| = m = n^\gamma$ *for some constant* $\gamma > 0$*. Then for all* $\delta \in (0, 1)$*, one cannot approximate* $\mathrm{D_G}(P, Q)$ *by a factor better than* 1.25 *in* $O((nm)^{1-\delta})$ *time unless OVH fails.*

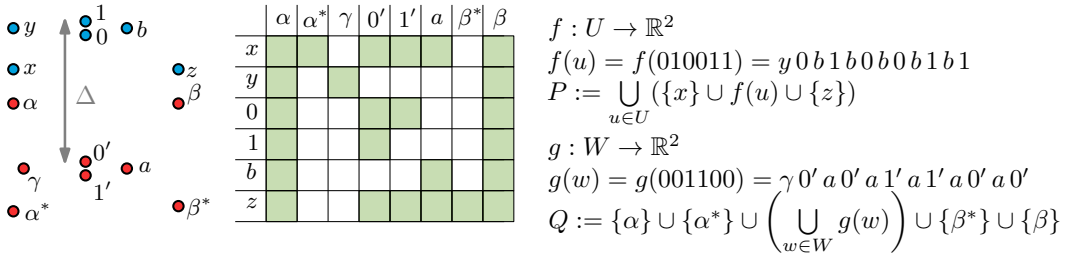


Figure 3 We illustrate the construction for planar curves by Bringmann [7]. Each $u \in U$ becomes a subcurve $f(u)$ of P and each $w \in W$ becomes a subcurve $g(w)$ of Q. Our table shows a green cell if the distance between the vertices is less than Δ. If the person is in a subcurve $f(u)$ and the dog is in $g(w)$ and the person moves from its vertex then the dog *must* also move.

4 Upper bound

We show that subquadratic algorithms *do* exist for unweighted planar graphs that exhibit additional structure. Specifically, our main result is as follows:

▶ **Theorem 6.** *Let* T *be a regular unit tiling and* (P, Q) *be paths in* T *with* $|P| = n$ *and* $|Q| = m$*. Given* $\Delta \in \mathbb{R}$*, we can output whether* $\mathrm{D_T}(P, Q) \leq \Delta$ *in* $O((n+m)^{1.5} \log(n+m))$ *time.*

We compute $\mathrm{D_T}(P, Q)$ as follows. Given (P, Q), we compute the smallest axis-aligned rectangles R_P and R_Q that contain P and Q, respectively, in $O(n + m)$ time. Let d_0 be the Euclidean length of the shortest segment (p, q) with $p \in R_P$ and $q \in R_Q$, and let d_1 be the Euclidean length of the longest such segment. Observe that $\mathrm{D_T}(P, Q) \in [\frac{d_0}{2}, 2d_1]$, that this range contains $O(n + m)$ integers, and that $\mathrm{D_T}(P, Q)$ is always integral. By performing a binary search over $[\frac{d_0}{2}, 2d_1] \cap \mathbb{Z}$, applying Theorem 6 at each step, we obtain:

▶ **Corollary 7.** *Let T be a regular unit tiling, and let P and Q be paths in T with $|P| = n$ and $|Q| = m$. Then $\mathrm{D}_{\mathrm{T}}(P, Q)$ can be computed in $O((n+m)^{1.5} \log^2(n+m))$ time.*

Additional definitions. To prove Theorem 6, we first introduce a few new concepts. Let T be a regular, unit, axis-aligned tiling. Based on T, we construct a super-tiling $\mathbb{T}$ of the same type as T, that is, if T is a square/triangular/hexagonal tiling, then $\mathbb{T}$ is also a square/triangular/hexagonal tiling. We state our results in general terms, but for ease of reading, the reader may assume T is a square tiling. For a (super) tiling $\mathbb{T}$, we define the *halfslabs* of any face F of $\mathbb{T}$ by picture (Figure 4).

▶ **Definition 8.** *Let T be a regular tiling, and let $\mathbb{T}$ be a super-tiling of T. We say that two faces F and F' of $\mathbb{T}$ are* well-separated *by a vertex $v \in V(T)$ if, for all $p \in V(T) \cap \overline{F}$ and $q \in V(T) \cap \overline{F'}$, there exists a shortest path from p to q in T that passes through v.*

Having faces F, F' be well-separated proves to be very useful, since in such a pair the distance between some vertex $p \in V(T) \cap \overline{F}$ and $q \in V(T) \cap \overline{F'}$ can be well-understood. We provide an efficient algorithm to compute the pairwise Fréchet distance between subpaths of P and Q whenever their corresponding faces in $\mathbb{T}$ are well-separated. We identify a sufficient condition that ensures that some pair F, F' is well-separated, based on the *alignment* of the two:

▶ **Definition 9.** *Let T be a regular tiling. Two faces F and F' of T are* aligned *if F' intersects a halfslab of F (or F intersects a halfslab of F').*

▶ **Lemma 10.** *Let T be an axis-aligned regular unit tiling, and let $\mathbb{T}$ be a super-tiling of T of the same type as T. Any two faces F, F' of $\mathbb{T}$ that are not aligned are well-separated by a vertex $v \in V(T)$.*

Proof. Consider first the case where T is a square tiling (and so $\mathbb{T}$ is too). The situation is depicted in Figure 5 (a). If the faces F, F' are not aligned, then F' is contained in one of the four regions (R_1, R_2, R_3, R_4) of the plane formed by the halfslabs of F. Let v be the vertex at the apex of the region which contains F'. We claim that for any vertices $p \in V(T) \cap \overline{F}$ and $q \in V(T) \cap \overline{F'}$, there exists a shortest path between p and q that passes through v. Hence F and F' are well-separated. The claim is easy to see from the properties of shortest paths in grid graphs. We provide an alternative proof of the claim, with the advantage that it can be more easily generalized to the triangular and hexagonal grid. See Figure 5 (b) and consider for $i = 1, 2, \ldots$ the circles of radius i in the plane graph T centred around p, and q.

$$h_i := \{x \in V(T) : \mathrm{dist}(p, x) = i\}, \quad h'_i := \{x \in V(T) : \mathrm{dist}(q, x) = i\}.$$

We can see that h_i, h'_i can be described as diamonds with four sides. Let us among the four sides of h_i denote by X_i the side that is facing towards F'. Likewise, let us among the

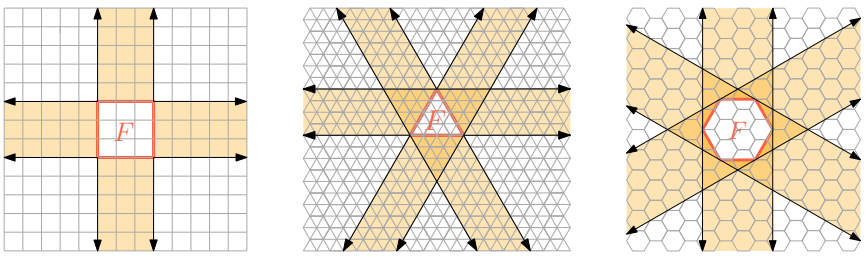


Figure 4 For a face F in a square/triangular/hexagonal (super-)tiling, its halfslabs are the infinite slabs enclosing F along the directions depicted here (orange).

four sides of h'_i denote by X'_i the side that is facing towards F. Note that all of X_i, X'_i are parallel for $i \in \mathbb{N}$. Let ℓ denote the line through vertex v parallel to the X_i. Let $a \in \mathbb{N}$ be minimal such that h_a touches ℓ. Since X_a is parallel to ℓ, we have $X_a \subseteq \ell$. Furthermore, no matter which $p \in V(T) \cap \overline{F}$ was chosen as the centre of the circles h_i, we always have $v \in X_a$. This follows due to the way and the angle in which the circles h_i expand. Analogously, let $b \in \mathbb{N}$ be minimal such that h'_b touches ℓ. Due to the way the h'_i expand and since F and F' are not aligned, we have $v \in X'_b$. Now, the circle h_a is entirely on one side of ℓ, the circle h'_b is entirely on the other side, and $v \in h_a \cap h'_b$. By definition of h_a, h'_b, this implies $\text{dist}(p, q) = a + b$ and there is a shortest path between p, q that passes through v.

For the case of triangular and hexagonal grids, the situation is depicted in Figures 6 and 7. Here the halfslabs divide the plane into 6 cone-shaped regions $R_1, \ldots, R_6$. Let v_j be the point at the apex of R_j for $j = 1, \ldots, 6$. Note that v_j is also a vertex of $V(T)$. For both the triangular and hexagonal grid, the definitions of $h_i, h'_i, X_i, X'_i, \ell, a, b$ can be made in an analogous matter. Analogously, it follows that $v_j \in h_a \cap h'_b$ which concludes the lemma. ◀

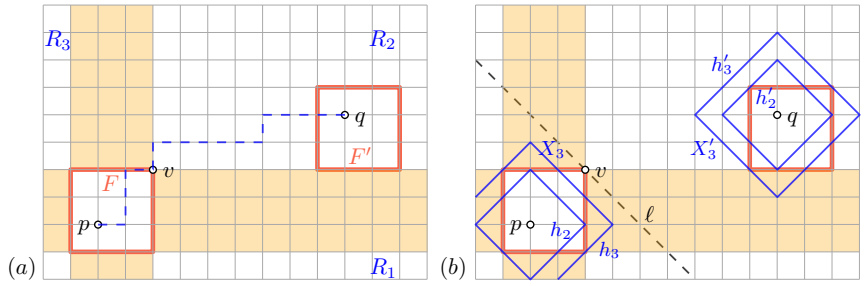


Figure 5 (a) The two faces F, F' of the square super-tiling $\mathbb{T}$ are non-aligned.

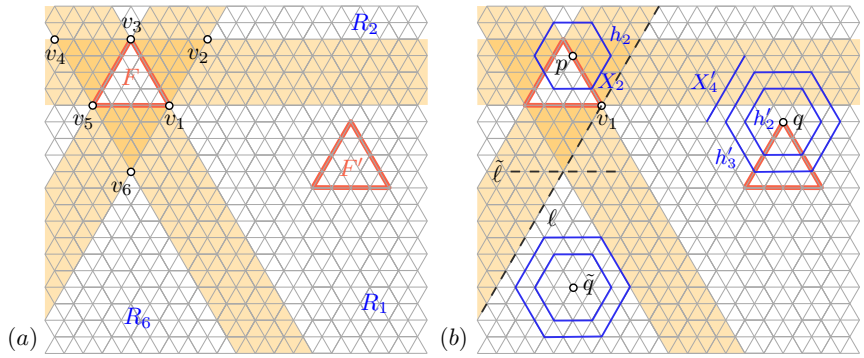


Figure 6 (a) Vertex v_1 well-separates non-aligned faces in a triangular grid. (b) Proof.

Algorithm overview. Given T and paths P and Q, we first compute a suitable super-tiling:

▶ **Lemma 11.** *Let T be a regular unit tiling, and let P and Q be two paths in T of lengths n and m. In $O((n+m)^{1.5})$ time, we can compute a super-tiling $\mathbb{T}$ such that:*

- *the edges of $\mathbb{T}$ have a length in $\Theta(\sqrt{n+m})$, and*
- *the total number of induced subpaths in $S(P, \mathbb{T}) \cup S(Q, \mathbb{T})$ (see Definition 4) is $O(\sqrt{n+m})$.*

Proof. Our proof is inspired by that of Chan and Rahmati [14, Lemma 1], who prove an analogous statement for grids and curves in higher-dimensional real space (without graphs). For the proof, we assume without loss of generality that T is axis-aligned.

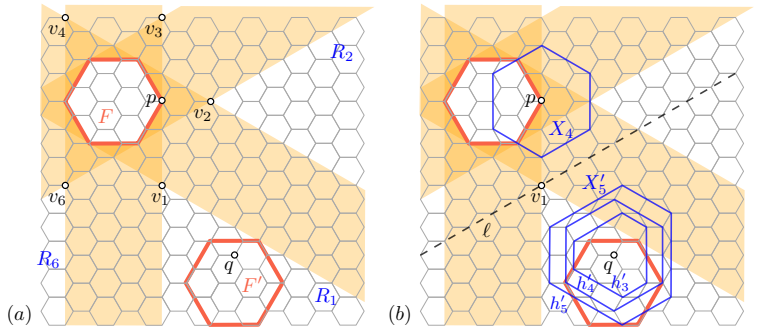


Figure 7 (a) Vertex v_1 well-separates non-aligned faces in a hexagonal grid. (b) Proof.

Let $f = \lceil \sqrt{n+m} \rceil$, and let $\mathbb{T}_0$ be any axis-aligned super-tiling of T with an edge length in $\Theta(f)$. For each $i \in [f-1]$, define $\mathbb{T}_i$ to be the super-tiling obtained by shifting $\mathbb{T}_0$ by the vector (i, i) if T is square, or by $(0, i\sqrt{3})$ if T is triangular or hexagonal. These shifted super-tilings remain axis-aligned super-tilings of T, as all their vertices align with those of T.

Since we shift in a direction not aligned with any edge of T, and since $i \leq f-1$, each edge or vertex of T lies on the boundary of $O(1)$ faces across the family $\{\mathbb{T}_i\}_{i=0}^{f-1}$. It follows that for any discrete vertex set $V \subseteq V(T)$ with $|V| = k$, there exists some $\mathbb{T}_i$ such that only $O(k/f)$ vertices lie on boundaries of faces in $\mathbb{T}_i$. Therefore, there exists a super-tiling $\mathbb{T} \in \{\mathbb{T}_0, \ldots, \mathbb{T}_{f-1}\}$ such that the total number of boundary vertices from P and Q is $O((n+m)/f)$, which implies $O((n+m)/f) = O(\sqrt{n+m})$ induced subpaths. To compute $\mathbb{T}$, we iterate over all vertices v of P and Q and, for each, determine the $O(1)$ super-tilings $\mathbb{T}_i$ with $v \in B(\mathbb{T}_i)$ by iterating over all $\mathbb{T} \in \{\mathbb{T}_0, \ldots, \mathbb{T}_{f-1}\}$. This takes $O((n+m)f)$ total time. We then select the super-tiling $\mathbb{T}_i$ that minimizes $|(P \cup Q) \cap B(\mathbb{T}_i)|$. ◀

Let $\mathbb{T}$ be the computed super-tiling of T. We partition P and Q into induced subpaths, denoted by $S(P, \mathbb{T})$ and $S(Q, \mathbb{T})$. These $O(\sqrt{n+m})$ subpaths naturally induce a subdivision of the free-space diagram M_Δ into $O(n+m)$ rectangular submatrices $M_\Delta[i\colon i', j : j']$, where $P[i : i'] \in S(P, \mathbb{T})$ and $Q[j : j'] \in S(Q, \mathbb{T})$ (see Figure 8(a)).

Our algorithm makes use of a wave-front method. We define a wave-front W as a monotone walk through the grid $[1, n] \times [1, m]$ that separates $(1, 1)$ from (n, m). At each grid point $(i, j) \in W$, we store a Boolean value indicating whether there exists a monotone walk F from $(1, 1)$ to (i, j) such that all points on F lie in free space: that is, $M_\Delta[x, y] = 0$ for all $(x, y) \in F$ (Figure 8(b)).

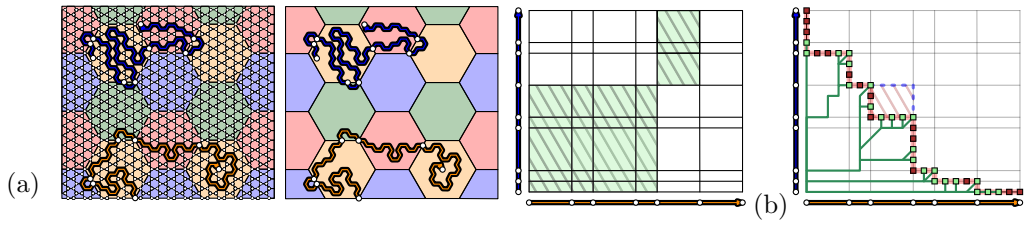


Figure 8 (a) For a given tiling T, we first compute a super-tiling $\mathbb{T}$ which partitions the edges of P and Q. These subpaths divide the Δ-free space matrix M_Δ into submatrices that correspond to aligned and non-aligned curve pairs (green dashed versus neutral). (b) We maintain a wave-front W which is a monotone curve through M_Δ that has a Boolean value at each entry. An *update* selects a submatrix $M_\Delta[i : i', j : j']$ and removes from W its bottom-left facets (adding its top-right facets).

An *update* to the wave-front W selects a pair of subpaths $(P[i:i'], Q[j:j']) \in S(P,\mathbb{T}) \times S(Q,\mathbb{T})$ such that the bottom and left facets of the submatrix $M_\Delta[i:i', j:j']$ lie on the current wave-front. The update removes these facets from W and inserts the top and right facets instead. To perform the update, we consider the faces F_P and F_Q of $\mathbb{T}$ that $P[i:i']$, respectively $Q[j:j']$, are assigned to, and proceed via a case distinction:

1. If F_P and F_Q are not aligned, then they are well-separated by a vertex v (by Lemma 10), and we show how to perform a fast update using v.
2. If F_P and F_Q are aligned, we perform a brute-force update and use a counting argument to bound the total running time of these updates.

We analyse both cases separately, observing that while some instances may involve only Case 1 or only Case 2 submatrices, we can independently bound the total cost for each case.

Case 1: Updating with $(P[i:i'], Q[j:j'])$ where F_P and F_Q are not aligned. We show an update algorithm that is near-linear in the perimeter of the submatrix $M_\Delta[i:i', j:j']$.

▶ **Lemma 12.** *Let $P[i:i'] \in S(P,\mathbb{T})$ and $Q[j:j'] \in S(Q,\mathbb{T})$ be subpaths whose faces F_P and F_Q in $\mathbb{T}$ are not aligned. Then the corresponding wave-front update can be performed in $O(k \log k)$ time, where $k = |P[i:i']| + |Q[j:j']|$.*

Proof. By Lemma 10, the subcurves $P[i:i']$ and $Q[j:j']$ are well-separated by some vertex $v \in V(T)$, which we can identify in constant time. Given v, we compute two curves in $\mathbb{R}$ whose Δ-free space matrix equals $M_\Delta[i:i', j:j']$. We define these curves as follows.

For a point $p \in P[i:i']$ we let $\bar{p} = \operatorname{dist}(p,v)$. Likewise, for a point $q \in Q[j:j']$, we let $\bar{q} = -\operatorname{dist}(q,v)$. Note the difference in sign. Because $P[i:i']$ and $Q[j:j']$ are well-separated by v, we have $\operatorname{dist}(p,q) = \operatorname{dist}(p,v) + \operatorname{dist}(v,q) = |\bar{p} - \bar{q}|$ for any $p \in P[i:i']$ and $q \in Q[j:j']$. We let $\bar{P} = (\bar{p}_i, \ldots, \bar{p}_{i'})$ and $\bar{Q} = (\bar{q}_j, \ldots, \bar{q}_{j'})$.

The Δ-free-space matrix $M_\Delta[i:i', j:j']$ is equal to the Δ-free space matrix of the sequences $\bar{P}$ and $\bar{Q}$ over the real line under the absolute value metric. Let $\bar{M}_\Delta$ be this matrix. We compute the top and right facets of $\bar{M}_\Delta$ in $O(k \log k)$ time with existing fast algorithms for 1D separated sequences [8, 31]. ◀

Case 2: Updating with $(P[i:i'], Q[j:j'])$ where F_P and F_Q are aligned. For aligned face pairs, we fall back on a brute-force update strategy based on *switching cells*, a concept introduced by Har-Peled, Knauer, Wang and Wenk. [4].

▶ **Definition 13** (Switching cells). *A cell (i,j) in M_Δ is a* switching cell *if $M_\Delta[i,j] = 0$, but either $M_\Delta[i,j-1] = 1$ or $M_\Delta[i,j+1] = 1$ (or both).*

The switching cells in a submatrix $M_\Delta[i:i', j:j']$ allow for a direct update of the wave-front:

▶ **Lemma 14** ([4]). *Let $P[i:i']$ and $Q[j:j']$ be two subpaths. Given all s switching cells in $M_\Delta[i:i', j:j']$, a brute-force wave-front update takes $O(|P[i:i']| + |Q[j:j']| + s)$ time.*

We apply the brute-force update algorithm to all subpath pairs $(P[i:i'], Q[j:j']) \in S(P,\mathbb{T}) \times S(Q,\mathbb{T})$ whose associated faces F_P and F_Q in $\mathbb{T}$ are aligned. We now show that the total number of switching cells encountered in these submatrices is not too large:

▶ **Lemma 15.** *Let T be a regular unit tiling, and let P and Q be two paths in T. Let $\mathbb{T}$ be any super-tiling of T with edge length $\Theta(\sqrt{n+m})$. Then there are at most $O(n\sqrt{n+m})$ pairs (p_i, q_j) such that $M_\Delta[i,j]$ is a switching cell and p_i and q_j lie on aligned faces of $\mathbb{T}$. We can compute these pairs in $O(n\sqrt{n+m}\log m)$ time.*

Proof. Fix a vertex p_i and let F_p be a face of $\mathbb{T}$ such that $p_i \in \overline{F_p}$. Since P and Q are paths in a unit tiling, a switching cell $M_\Delta[i,j]$ occurs only if q_j lies at distance exactly Δ from p_i.

Let $B_i(\Delta) \subseteq V(T)$ denote the set of all vertices at distance exactly Δ from p_i. This set lies on the boundary of a shape whose geometry depends on the tiling type:

- $B_i(\Delta)$ is the boundary of a diamond if T is a square tiling.
- $B_i(\Delta)$ is the boundary of a regular hexagon if T is a triangular tiling.
- $B_i(\Delta)$ is the boundary of a (non-regular) hexagon if T is a hexagonal tiling.

These regions are convex. Consider a face F' of $\mathbb{T}$ that is aligned with F_p. The intersection of $B_i(\Delta)$ with $\overline{F'}$ contains at most $O(\sqrt{n+m})$ vertices. Each of these vertices could correspond to at most one vertex $q_j \in Q$ where (i,j) is a switching cell. There are only $O(1)$ faces of $\mathbb{T}$ that are aligned with F_p, whose closures intersect $B_i(\Delta)$. Hence, there are at most $O(\sqrt{n+m})$ such q_j values per p_i. Summing over all n vertices of P gives the desired $O(n\sqrt{n+m})$ bound.

What remains is to compute these pairs (p_i, q_j). We preprocess Q in a membership query data structure. For each $p_i \in P$, we select its assigned face F_p in constant time and we identify the $O(1)$ faces F' of $\mathbb{T}$ that are aligned with F_p and intersect $B_i(\Delta)$ in constant time. For every such face F', we iterate over the vertices v in $\overline{F'} \cap B_i(\Delta)$. We find if there exists a $q_j \in Q$ with $v = q_j$ in $O(\log m)$ time through a membership query. If so, then we compute $\text{dist}(p_i, q_{j+1})$ and $\text{dist}(p_i, q_{j-1})$ to determine whether $M_\Delta[i,j]$ is a switching cell. ◀

We are now ready to prove our main theorem:

▶ **Theorem 6.** *Let T be a regular unit tiling and (P,Q) be paths in T with $|P| = n$ and $|Q| = m$. Given $\Delta \in \mathbb{R}$, we can output whether $\mathrm{D}_{\mathrm{T}}(P,Q) \leq \Delta$ in $O((n+m)^{1.5}\log(n+m))$ time.*

Proof. Given T, P, and Q, we begin by applying Lemma 11 to compute in $O((n+m)^{1.5})$ time a super-tiling $\mathbb{T}$ and the corresponding subpaths $S(P,\mathbb{T})$ and $S(Q,\mathbb{T})$. The number of subpath pairs $(P[i:i'], Q[j:j']) \in S(P,\mathbb{T}) \times S(Q,\mathbb{T})$ is $O(n+m)$. For each pair, we determine if the corresponding faces F_P and F_Q in $\mathbb{T}$ are aligned, which takes $O(n+m)$ total time.

The product set $S(P,\mathbb{T}) \times S(Q,\mathbb{T})$ induces a grid of $O(n+m)$ rectangular submatrices of M_Δ with $O(\sqrt{n+m})$ rows and columns. Thus:

$$\sum_{\text{submatrices } M' \text{ of } M_\Delta} |\text{facets}(M')| = \sum_{(P[i:i'],Q[j:j']) \in S(P,\mathbb{T}) \times S(Q,\mathbb{T})} (i'-i+j'-j) \qquad \in O((n+m)^{1.5}). \quad (1)$$

Next, we initialize the wave-front W by computing the first row and first column of M_Δ, i.e., all entries $M_\Delta[i,1]$ for $i \in [n]$ and $M_\Delta[1,j]$ for $j \in [m]$, in $O(n+m)$ time. We then iterate diagonally through all pairs $(P[i:i'], Q[j:j'])$ in $S(P,\mathbb{T}) \times S(Q,\mathbb{T})$ and perform a wave-front update for each pair.

If the two faces F_P and F_Q of $\mathbb{T}$ assigned to $P[i:i']$ and $Q[j:j']$ are *not aligned* then we apply Lemma 12. The total time spent in this case equals:

$$\sum_{(P[i:i'],Q[j:j']) \text{ where } (F_P,F_Q) \text{ are not aligned}} O((i'-i+j'-j))\log(i'-i+j'-j)$$

By Equation 1, this sum is upper bound by $O((n+m)^{1.5}\log(n+m))$.

If the two faces F_P and F_Q of $\mathbb{T}$ assigned to $P[i:i']$ and $Q[j:j']$ are aligned then we apply Lemma 14 instead. The total time spent is:

$$\sum_{P[i:i'],Q[j:j']) \text{ where } (F_P,F_Q) \text{ are aligned}} O((i'-i+j'-j) + \text{switching cells}(M_\Delta[i:i', j:j']))$$

The first term and second term are bounded by $O((n+m)^{1.5}\log(n+m))$ by Equation 1 and Lemmas 14+15, respectively. Once the wave-front W reaches the entry (n,m), we have determined whether $\mathrm{D_T}(P,Q) \leq \Delta$, proving the theorem. ◀

5 The discrete Fréchet distance under the L_1 metric

We extend our techniques for paths in tilings to curves in the plane. Fix some universal constant $\gamma > 0$ and fix some value $\varepsilon \in (0,1)$. We consider $\mathbb{R}^2$ under the L_1 metric, and two polygonal curves P and Q with n and m vertices, respectively. In full generality, we only require that all edges in P have length at most γ and that Q is an (ε,δ)-curve. Denote their discrete Fréchet distance under the L_1 metric by $D_F(P,Q)$. We show that we can compute $D_F(P,Q)$ in $\tilde{O}\left(\frac{\sqrt{\delta}}{\varepsilon}(n+m)^{1.5}\right)$ time. We adapt this algorithm in the full version of this paper, to make it work for any L_c metric, by allowing some inaccuracy.

Our approach. Given some $\Delta \in \mathbb{R}$, we first show how to decide whether $D_F(P,Q) \leq \Delta$ using the wave-front propagation technique from Section 4, with suitable modifications for curves in the plane. Let $t > 1$ be a parameter to be determined later. We define an axis-aligned square tiling $\mathbb{T}$ of the plane where each square has edge length t.

▶ **Definition 16.** *We define the* breakpoints $B(P,\mathbb{T})$ *of P as the set of vertices $p \in V(P)$ such that some edge of P incident to p intersects an edge of $\mathbb{T}$.*

Note that $B(P,\mathbb{T})$ may contain overlapping but distinct vertices of P (since an (ε,δ)-curve may visit the same point up to δ times). If we remove all edges from P which intersect some edge of $\mathbb{T}$, we see that the breakpoints induce a partition of the vertices of P into pairwise vertex-disjoint subcurves $P[i:i']$, each fully contained in the closure $\overline{F}$ of a face $F \in \mathbb{T}$. We refer to these subcurves as the *induced subcurves* $S(P,\mathbb{T})$ and we assign each induced subcurve to a face in $\mathbb{T}$. We prove that for a suitable choice of $\mathbb{T}$, the total number of induced subcurves satisfies $|S(P,\mathbb{T}) \cup S(Q,\mathbb{T})| = O((n+m)/t)$.

We then apply our wave-front algorithm over the product set $S(P,\mathbb{T}) \times S(Q,\mathbb{T})$. For a pair of subcurves $(P[i:i'],Q[j:j'])$, let F_P and F_Q denote their respective containing faces in $\mathbb{T}$. We distinguish two cases:

- If F_P and F_Q are not aligned, we show that $(P[i:i'],Q[j:j'])$ is well-separated by a corner of either F_P or F_Q. We then apply Lemma 12 to perform a wave-front update in $O((i'-i+j'-j)\log(n+m))$ time.
- If F_P and F_Q are aligned, we apply the brute-force algorithm of Lemma 14, but with a more careful analysis of the number of switching cells.

Balancing the choice of t is critical: a larger t reduces the number of wave-front updates, while a smaller t reduces the number of switching cells encountered. We choose t to optimize the overall running time.

Choosing a tiling. We construct a tiling $\mathbb{T}$ such that it creates few induced subcurves.

▶ **Lemma 17.** *Given $t \geq 1$, we can compute in $O((n+m)t)$ time a square tiling $\mathbb{T}$ of edge length t such that the number of induced subcurves satisfies $|S(P,\mathbb{T}) \cup S(Q,\mathbb{T})| = O((n+m)/t)$.*

Proof. The proof mirrors that of Lemma 11. Let $\mathbb{T}_0$ be an axis-aligned square tiling with edge length t, and let $\mathbb{T}_i$ denote the tiling obtained by shifting $\mathbb{T}_0$ by (i,i) for $i \in [t-1]$. We required that each edge e of P or Q has length at most $\gamma \in O(1)$. Thus each edge intersects the boundary of at most $O(1)$ of the shifted tilings. It follows that there exists an index i such that $\mathbb{T}_i$ induces only $O((n+m)/t)$ breakpoints across P and Q.

To find $\mathbb{T}_i$, we iterate over all $O(n+m)$ edges of P and Q. For each edge, we test for intersection with each of the t candidate tilings in constant time. This gives an overall running time of $O((n+m)t)$ to find the tiling $\mathbb{T}_i$ that minimizes $|B(P,\mathbb{T}_i) \cup B(Q,\mathbb{T}_i)|$. ◀

Wave-front updates. Unlike in Section 4, the subcurves in $S(P,\mathbb{T})$ are vertex-disjoint. Thus, we never have a submatrix $M_\Delta[i:i', j:j']$ whose bottom-left facet coincides with the current wave-front W. However, we can always find a submatrix $M' := M_\Delta[i:i', j:j']$ where each of the cells on the bottom-left facet of M' is adjacent to a cell in W. We extend W to cover it in $O(i'-i+j'-j)$ time via brute force and thereby define wave-front updates analogously.

Case 1: Updating with $(P[i:i'], Q[j:j'])$ where F_P and F_Q are not aligned. Our key observation is that, even though the edges of $P[i:i']$ and $Q[j:j']$ do not coincide with $\mathbb{T}$, a pair of faces (F_P, F_Q) of $\mathbb{T}$ are still well-separated whenever they are not aligned:

▶ **Lemma 18.** *Let $P[i:i'] \in S(P,\mathbb{T})$ and $Q[j:j'] \in S(Q,\mathbb{T})$ be subcurves whose faces F_P and F_Q in $\mathbb{T}$ are not aligned. Then the corresponding wave-front update can be performed in $O(k \log k)$ time, where $k = |P[i:i']| + |Q[j:j']|$.*

Proof. Assume, without loss of generality, that the bottom-left corner of F_P lies above and to the right of the top-right corner of F_Q. Then any shortest L_1 path from a point in $\overline{F_P}$ to one in $\overline{F_Q}$ may be rerouted through that corner. The rest follows from Lemma 12. ◀

Case 2: Updating with $(P[i:i'], Q[j:j'])$ where F_P and F_Q are aligned. We upper bound the number of switching cells $M_\delta[i,j]$ where p_i and q_j lie in aligned faces of $\mathbb{T}$:

▶ **Lemma 19.** *Let $\mathbb{T}$ be the tiling from Lemma 17. Then there are at most $O(\frac{\delta n(t+1)}{\varepsilon^2})$ pairs (p_i, q_j) such that $M_\Delta[i,j]$ is a switching cell and p_i and q_j lie in aligned faces of $\mathbb{T}$. We can compute these pairs in $O(\frac{\delta n(t+1)}{\varepsilon^2} \log m)$ time.*

Proof. We preprocess Q by snapping its vertices to a square grid G of edge length ε^{-1}, forming Q'. Each vertex in Q' corresponds to $O(\delta)$ original vertices since Q is an (ε,δ)-curve. We store Q' in a spatial data structure (e.g., a lexicographically sorted balanced binary tree), where a query point z returns all $q_j \in Q$ corresponding to z in $O(\delta + \log n)$ time.

Recall that all edges of Q have length at most $\gamma \in O(1)$. Fix a vertex p_i and let F_p be a face of $\mathbb{T}$ that contains p_i. Let $B_i(\Delta)$ be the metric circle (under the L_1 metric) of radius Δ centred at p_i. Let Γ denote a ball with radius 2γ and $B_i^*(\Delta)$ be the metric annulus that is the Minkowski-sum $\Gamma \oplus B_i(\Delta)$. Observe that an edge (q_j, q_{j+1}) of Q intersects $B_i(\Delta)$ only if one of its corresponding vertices q'_j and q'_{j+1} of Q' lies in $B_i^*(\Delta)$.

There are $O(1)$ faces F' of $\mathbb{T}$ that are aligned with F_p and that intersect $B_i^*(\Delta)$. Let $A := B_i^*(\Delta) \cap F'$. Distinguishing between $\Delta > 1$ and $\Delta \leq 1$, we see that A has an area of $O(\max\{1,t\}\gamma)$ and therefore A contains at most $O(\frac{\gamma \max\{1,t\}}{\varepsilon^2}) = O(\frac{t+1}{\varepsilon^2})$ vertices of G. We iterate over the vertices in $B_i^*(\Delta) \cap \overline{F'} \cap V(G)$ and for each we perform a membership query on Q'. This returns $O(\frac{\delta(t+1)}{\varepsilon^2})$ vertices of Q'. For each reported vertex q'_j we test if $M_\Delta[i,j]$ is a switching cell in constant time. Thus, for any $p_i \in P$, we compute all switching cells $M_\Delta[i,j]$ that correspond to the lemma statement in $O(\frac{\delta(t+1)}{\varepsilon^2} \log m)$ time. ◀

▶ **Theorem 20.** *Let P and Q be two curves in $\mathbb{R}^2$ under the L_1 metric with $|P| = n$ and $|Q| = m$. Fix a universal constant $\gamma \in O(1)$. For $\varepsilon > 0$ (with $\varepsilon \in \omega(\sqrt{\frac{\delta}{n}})$) and $\Delta \in \mathbb{R}$, if P has edge length at most γ and Q is an (ε,δ)-curve, then we can compute whether $D_F(P,Q) \leq \Delta$ in $O(\frac{\sqrt{\delta}}{\varepsilon}(n+m)^{1.5} \log(n+m))$ time.*

Proof. Let $t \geq 1$ be a value that we set later. Given P and Q, we apply Lemma 17 to compute in $O((n+m)t)$ time our tiling $\mathbb{T}$ and the $O((n+m)/t)$ induced subcurves $S(P,\mathbb{T})$ and $S(Q,\mathbb{T})$. The product set $S(P,\mathbb{T}) \times S(Q,\mathbb{T})$ induces a grid over M_Δ with $O((n+m)/t)$ rows and columns. Thus:

$$\sum_{\text{submatrices } M' \text{ of } M_\Delta} |\text{facets}(M')| = \sum_{(P[i:i'],Q[j:j']) \in S(P,\mathbb{T}) \times S(Q,\mathbb{T})} (i'-i+j'-j) \qquad \in O((n+m)^2/t). \quad (2)$$

We iteratively perform wave-front updates with the input $(P[i:i'], Q[j:j'])$. Let F_P and F_Q denote the corresponding two faces of $\mathbb{T}$. If F_P and F_Q are *not aligned* then we apply Lemma 14. The total time spent in this case equals:

$$\sum_{(P[i:i'],Q[j:j']) \text{ where } (F_P,F_Q) \text{ are not aligned}} O((i'-i+j'-j)) \log(n+m)$$

By Equation 2, this sum is upper bound by $O(\frac{(n+m)^2}{t} \log(n+m))$.

If F_P and F_Q are aligned then we apply Lemma 15 instead. The total time spent is:

$$\sum_{P[i:i'],Q[j:j']) \text{ where } (F_P,F_Q) \text{ are aligned}} O((i'-i+j'-j) + \text{switching cells}(M_\Delta[i:i', j:j']))$$

The first term is upper bounded by $O(\frac{(n+m)^2}{t})$ by Equation 2. The second term is upper bounded by $O(\frac{\delta n(t+1)}{\varepsilon^2} \log m)$ by combining Lemma 14 and 19. It follows, assuming that $\frac{\varepsilon}{\sqrt{\delta}}\sqrt{n}$ goes to infinity, that if we choose $t \in \Theta(\frac{\varepsilon \cdot (n+m)}{\sqrt{\delta n}})$ the total running time is upper bounded by $O(\frac{\sqrt{\delta}}{\varepsilon}(n+m)^{1.5} \log(n+m))$. ◀

Computing $D_F(P,Q)$. Given P and Q, there is a set of $O(nm)$ values S, which can be computed in $O(nm)$ time, such that $D_F(P,Q) \in S$. The classical algorithm to compute $D_F(P,Q)$ performs binary search over S. Since we want to use subquadratic time, we cannot compute the set S explicitly. Instead, we represent S as the *Cartesian sum* $X \oplus Y$ of two sets of $O(n+m)$ values. The Cartesian sum of X and Y is the multiset $X \oplus Y = \{\{x+y \mid x \in X, y \in Y\}\}$. We can use existing techniques [21, 27] for selection in Cartesian sums to binary search over the Cartesian sum $X \oplus Y$ without explicitly computing its $O(|X| \cdot |Y|)$ elements.

Specifically, we define X and Y as follows. The set X contains, for each point $(x,y) \in P$, the four values $x+y$, $-x+y$, $x-y$ and $-x-y$. The set Y is defined analogously for points in Q. For all $(p,q) \in P \times Q$, the distance between p and q is the sum of an element in X and an element in Y. That is, their distance is an element of $X \oplus Y$. Hence, $D_F(P,Q) \in X \oplus Y$.

After sorting X and Y, we can compute the i^{th} smallest value in $X \oplus Y$, for any given i, in $O(|X|+|Y|) = O(n+m)$ time [21, 27]. We binary search over the integers $1, \ldots, |X| \cdot |Y|$, and for each considered integer i, we compute the i^{th} smallest value Δ in $X \oplus Y$. Then we use our decision algorithm (Theorem 20) to decide whether $D_F(P,Q) \leq \Delta$ to guide the search to $D_F(P,Q)$. Thus, we obtain the following result:

▶ **Corollary 21.** *Given $\gamma > 0$, let P and Q be curves in $\mathbb{R}^2$ under L_1 with $|P|+|Q| = n+m$. For any $\varepsilon > 0$, if P has edge length at most γ and Q is an (ε,δ)-curve, then $D_F(P,Q)$ can be computed in $O(\frac{\sqrt{\delta}}{\varepsilon}(n+m)^{1.5} \log^2(n+m))$ time.*

References

1 Pankaj K Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, 2014. doi:10.1137/130920526.

2 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995. doi:10.1142/S0218195995000064.

3 Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004. doi:10.1007/s00453-003-1042-5.

4 Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet distance for curves, revisited. In *European Symposium on Algorithms (ESA)*, pages 52–63, 2006. doi:10.1007/11841036_8.

5 Yoonsik Bang, Jiyoung Kim, and Kiyun Yu. An improved map-matching technique based on the Fréchet distance approach for pedestrian navigation services. *Sensors*, 16(10), 2016. doi:10.3390/s16101768.

6 Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *International Conference on Very Large Data Bases (VLDB)*, pages 853–864, 2005. doi:10.5555/1083592.1083691.

7 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

8 Karl Bringmann and Marvin Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *International Journal of Computational Geometry & Applications*, 27(1-2):85–120, 2017. doi:10.1142/S0218195917600056.

9 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *Journal of Computational Geometry*, 7(2):46–76, 2016. doi:10.4230/LIPIcs.SOCG.2015.739.

10 Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristan, Rodrigo I Silveira, Frank Staals, and Carola Wenk. Clustering trajectories for map construction. In *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 1–10, 2017. doi:10.1145/3139958.3139964.

11 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017. doi:10.1137/1.9781611973402.103.

12 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2887–2901, 2019. doi:10.5555/3310435.3310614.

13 Maike Buchin, Bernhard Kilgus, and Andrea Kölzsch. Group diagrams for representing trajectories. *International Journal of Geographical Information Science*, 34(12):2401–2433, 2020. doi:10.1145/3283207.3283208.

14 Timothy Chan and Zahed Rahmati. An improved approximation algorithm for the discrete Fréchet distance. *Information Processing Letters (IPL)*, 138:72–74, 2018. doi:10.1016/J.IPL.2018.06.011.

15 Daniel Chen, Anne Driemel, Leonidas Guibas, Andy Nguyen, and Carola Wenk. Approximate map matching with respect to the Fréchet distance. In *ACM Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–83, 2011. doi:10.1137/1.9781611972917.8.

16 Thomas Devogele. A new merging process for data integration based on the discrete Fréchet distance. In *Advances in spatial data handling*, pages 167–181, 2002. doi:10.1007/978-3-642-56094-1_13.

17 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete Computational Geometry*, 48(1):94–127, 2012. doi:10.1007/s00454-012-9402-z.

18 Anne Driemel, Amer Krivošija, and Christian Sohler. Clustering time series under the Fréchet distance. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 766–785. SIAM, 2016. doi:10.5555/2884435.2884490.

19 Anne Driemel, Ivor van der Hoog, and Eva Rotenberg. On the discrete Fréchet distance in a graph. In *International Symposium on Computational Geometry (SoCG)*, pages 36:1–36:18, 2022. doi:10.4230/LIPICS.SOCG.2022.36.

20 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

21 Greg N. Frederickson and Donald B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM Journal of Computing*, 13(1):14–30, 1984. doi:10.1137/0213002.

22 Branko Grünbaum and Geoffrey Colin Shephard. *Tilings and patterns*. Courier Dover Publications, 1987. doi:10.5555/19304.

23 Joachim Gudmundsson, Majid Mirzanezhad, Ali Mohades, and Carola Wenk. Fast Fréchet distance between curves with long edges. *International Journal of Computational Geometry & Applications*, 29(02):161–187, 2019. doi:10.1145/3191801.3191811.

24 Richard Kenefic. Track clustering using Fréchet distance and minimum description length. *Journal of Aerospace Information Systems*, 11(8):512–524, 2014. doi:10.2514/1.I010170.

25 Maximilian Konzack, Thomas McKetterick, Tim Ophelders, Maike Buchin, Luca Giuggioli, Jed Long, Trisalyn Nelson, Michel A Westenberg, and Kevin Buchin. Visual analytics of delays and interaction in movement data. *International Journal of Geographical Information Science*, 31(2):320–345, 2017. doi:10.5555/3048386.3048392.

26 Ariane Mascret, Thomas Devogele, Iwan Le Berre, and Alain Hénaff. Coastline matching process based on the discrete Fréchet distance. In *Progress in Spatial Data Handling*, pages 383–400. Springer, 2006. doi:10.1007/3-540-35589-8_25.

27 Andranik Mirzaian and Eshrat Arjomandi. Selection in X+Y and matrices with sorted rows and columns. *Information Processing Letters*, 20(1):13–17, 1985. doi:10.1016/0020-0190(85)90123-1.

28 Roniel Sousa, Azzedine Boukerche, and Antonio Loureiro. Vehicle trajectory similarity: Models, methods, and applications. *ACM Computing Surveys*, 53(5), 2020. doi:10.1145/3406096.

29 E Sriraghavendra, K Karthik, and Chiranjib Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Ninth International Conference on Document Analysis and Recognition (ICDAR)*, pages 461–465, 2007. doi:10.1109/ICDAR.2007.4378752.

30 Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. A survey of trajectory distance measures and performance evaluation. *Proceedings of the VLDB Endowment*, 29(1):3–32, 2020. doi:10.1007/s00778-019-00574-9.

31 Thijs van der Horst, Marc van Kreveld, Tim Ophelders, and Bettina Speckmann. The geodesic Fréchet distance between two curves bounding a simple polygon. *CoRR*, abs/2501.03834, 2025. doi:10.48550/arXiv.2501.03834.

32 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science (TCS)*, 348(2-3):357–365, 2005. doi:10.1016/J.TCS.2005.09.023.

33 Dong Xie, Feifei Li, and Jeff M Phillips. Distributed trajectory similarity search. *Proceedings of the VLDB Endowment*, 10(11):1478–1489, 2017. doi:10.14778/3137628.3137655.

Instance-Optimal Imprecise Convex Hull

Sarita de Berg
IT University of Copenhagen, Denmark

Ivor van der Hoog
IT University of Copenhagen, Denmark

Eva Rotenberg
IT University of Copenhagen, Denmark

Daniel Rutschmann
IT University Copenhagen, Denmark

Sampson Wong
University of Copenhagen, Denmark

Abstract

Imprecise measurements of a point set $P = (p_1, \ldots, p_n)$ can be modelled by a family of regions $F = (R_1, \ldots, R_n)$, where each imprecise region $R_i \in F$ contains a unique point $p_i \in P$. A *retrieval* models an accurate measurement by replacing an imprecise region R_i with its corresponding point p_i.

We construct the convex hull of an imprecise point set in the plane, by determining the cyclic ordering of the convex hull vertices of P as efficiently as possible. Efficiency is interpreted in two ways: (i) minimising the number of retrievals, and (ii) the computation time to determine the set of regions that must be retrieved.

Previous works focused on only one of these two aspects: either minimising retrievals or optimising algorithmic runtime. Our contribution is the first to simultaneously achieve both. Let $r(F, P)$ denote the minimal number of retrievals required by any algorithm to determine the convex hull of P for a given instance (F, P). For a family F of n constant-complexity polygons, our main result is a reconstruction algorithm that performs $\Theta(r(F, P))$ retrievals in $O(r(F, P) \log^3 n)$ time.

Compared to previous approaches that achieve optimal retrieval counts, we improve the runtime per retrieval from polynomial to polylogarithmic. We extend the generality of previous results to simple k-gons, to pairwise disjoint disks with radii in $[1, k]$, and to unit disks where at most k disks overlap in a single point. Our runtime scales linearly with k.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases convex hull, imprecise geometry preprocessing model, partial information

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.25

Related Version *Full Version*: `https://arxiv.org/abs/2504.02611`

Funding This research is supported by the Carlsberg Foundation Fellowship CF21-0302 "Graph Algorithms with Geometric Applications" and the VILLUM Foundation grants VIL37507 "Efficient Recomputations for Changeful Problems".
Ivor van der Hoog: This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987.
Sampson Wong: This project has recieved funding from the European Union's Skłodowska-Curie Actions Postdoctoral Fellowship grant agreement No 101146276.

1 Introduction

Imprecision is inherent in real-world data. It arises from rounding errors in floating-point computations, inaccuracies in measurement, and delayed sampling in GPS devices. In many scenarios, imprecise data can be refined at a cost by computing exact values or by taking additional samples. This is formalised in the model of *imprecise geometry* introduced by Held and Mitchell [13], and studied further in [8–10, 13, 17, 19, 25].


© Sarita de Berg, Ivor van der Hoog, Eva Rotenberg, Daniel Rutschmann, and Sampson Wong;
licensed under Creative Commons License CC-BY 4.0



Imprecise geometry. An imprecise point set [13] is defined as a family of regions $F = (R_1, \ldots, R_n)$, where each region R_i contains a unique but unknown point p_i. A *realisation* $P \sim F$ is a sequence $P = (p_1, \ldots, p_n)$ where $p_i \in R_i$. An input instance is a pair (F, P). A *retrieval* operation reveals the precise location of a point p_i, replacing R_i with p_i.

Let $F \odot_P B$ denote the family F after retrieving all $R_i \in B$, where $B \subset F$. The aim of a *reconstruction algorithm* is to identify a subset $B \subset F$ such that for all realisations $P_1, P_2 \sim F \odot_P B$, the algorithm's output (e.g., the cyclic order of the region indices of the points around the convex hull) is identical for P_1 and P_2. Figure 1 illustrates an example. After retrieving the subset $B = \{R_3, R_4\}$, the cycling ordering of the convex hull vertices is identically (p_1, p_3, p_2, p_4) for all realisations of $F \odot_P B$.

We evaluate reconstruction algorithms by three criteria: the preprocessing time, the total number of retrievals, and the running time per retrieval. Next, we consider instance-optimal algorithms, which minimise the total number of retrievals in the strictest sense.

Worst-case results. Many geometric problems have been studied in imprecise geometry, including Delaunay triangulations [8, 13, 19, 25], convex hulls [9, 10], Gabriel graphs [19], and onion decompositions [17]. For these geometric problems, it is known that any reconstruction algorithm must, in the worst case, use $\Omega(n)$ retrievals. Most of the existing algorithms assume that the regions in F are pairwise disjoint unit disks [8, 9, 13, 17, 19, 25]. Under these assumptions, deterministic worst-case optimal algorithms exist with $O(n \log n)$ preprocessing time, $O(n)$ retrievals, and $O(n)$ reconstruction time. Ezra and Mulzer [10] allow F to consist of arbitrary lines and reconstruct the convex hull using $O(n)$ retrievals and $O(n \cdot \alpha(n))$ expected time where $\alpha(n)$ denotes the inverse Ackermann function. Buchin, Löffler, Morin and Mulzer [3] permit overlapping unit disks of *ply* k (i.e., at most k disks intersect each point in the plane). We summarize these results in the top three rows of Table 1.

Instance-optimality. In many instances, worst case bounds are overly pessimistic. Intuitively, an algorithm is instance-optimal if, for every input (F, P), no other algorithm performs better on that instance. Constructing instance-optimal algorithms is challenging, as they must match every alternative algorithm, including those tailored for specific instances.

Afshani, Barbay, and Chan [1] proved that, for many geometric problems including constructing the convex hull, instance-optimal reconstruction time is unachievable. Likewise, it is easy to see that instance-optimality in the number of retrievals cannot be guaranteed in general: E.g., let F consist of two overlapping intervals R_1 and R_2 in $\mathbb{R}$, with $R_1 \setminus R_2 \neq \emptyset$ and $R_2 \setminus R_1 \neq \emptyset$. Define P_1 with $p_1 \in R_1 \setminus R_2$ and $p_2 \in R_1 \cap R_2$, and P_2 with $p_1 \in R_1 \cap R_2$

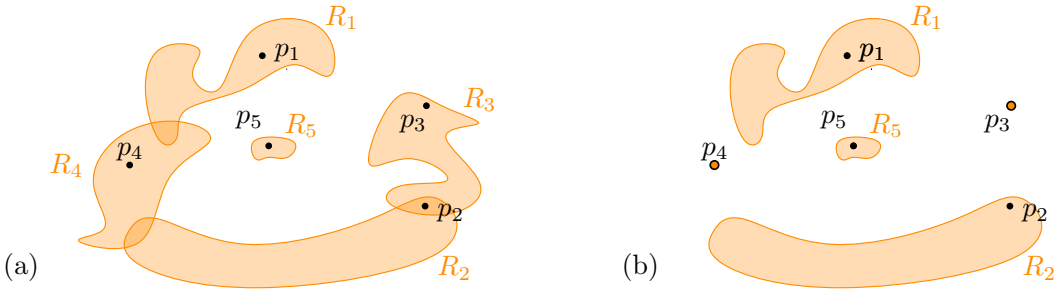


Figure 1 (a) A family of regions $F = R_1, \ldots, R_5$ and a sequence $P = (p_1, \ldots, p_5)$ with $P \sim F$. (b) If we retrieve R_3 and R_4 to obtain F' then for all $P' \sim F'$ the convex hull equals (p_1, p_3, p_2, p_4). Note that if p_3 would lie in $R_2 \cap R_3$ instead, then retrieving only R_3 and R_4 does not suffice.

and $p_2 \in R_2 \setminus R_1$. In (F, P_1), retrieving R_1 suffices; in (F, P_2), retrieving R_2 suffices. No algorithm can distinguish between the two instances without making a retrieval, and hence must use two retrievals on one of the instances. Since exact instance-optimality cannot be obtained, we instead aim for *asymptotic* instance-optimality.

Formally, we denote for any input instance by $r(F, P)$ the minimum integer such that there exists a subset $B \subset F$ of size $r(F, P)$ where all realizations $P' \sim (F \odot_P B)$ have the same vertex-ordering along the convex hull of P'. We note that $r(F, P)$ is, equivalently, the optimal number of retrievals. We say that an algorithm is *instance-optimal* if for all inputs (F, P) it uses $\Theta(r(F, P))$ retrievals. We highlight that $r(F, P)$ depends on both the region family F and the specific realisation P. To illustrate, suppose F consists of nested rectangles $R_n \subset R_{n-1} \subset \ldots \subset R_1$. If $P \sim F$ is such that the convex hull of (p_1, p_2, p_3, p_4) contains all of R_5, then retrieving just those four suffices: $r(F, P) = 4$. But if all p_i coincide then $r(F, P) = n$.

Prior instance-optimal work. Only a few instance-optimal reconstruction algorithms are known (see Table 1). Bruce, Hoffmann, Krizanc, and Raman [2] presented the first such algorithm for the convex hull. Their approach is computationally expensive, using an unspecified but superlinear time per retrieval (we discuss this in detail in the full version). Subsequent works show that polylogarithmic time per retrieval is achievable, albeit on geometric structures that are much simpler than the convex hull.

Van der Hoog, Kostitsyna, Löffler, and Speckmann [21] introduced the first near-linear instance-optimal algorithm, solving the sorting problem for one-dimensional intervals. Their algorithm preprocesses the intervals in $O(n \log n)$ time and uses $\Theta(r(F, P))$ retrievals, with at most logarithmic time per retrieval. Later, Van der Hoog, Kostitsyna, Löffler, and Speckmann [22] extended this approach to two-dimensional inputs for Pareto front reconstruction, under the assumption that F consists of pairwise disjoint axis-aligned rectangles. Their method also guarantees at most logarithmic time per retrieval.

Here, we show that polylogarithmic time per retrieval is achievable for reconstructing the convex hull with an instance-optimal number of retrievals. Moreover, our work generalises previous works in that we can handle overlapping regions in two dimensions, whereas previous works could only support one-dimensional inputs or disjoint two dimensional inputs.

Simultaneous work. Simultaneously and independently of this paper, Löffler and Raichel [18] developed an algorithm for reconstructing the convex hull when F is a set of unit disks of ply k. Their number of retrievals (Table 1) is not instance-optimal but rather worst-case optimal for every instance F. Formally, they use $O(w(F))$ retrievals for some region-dependent value $w(F)$ with $r(F, P) \leq w(F) \leq n$. We discuss their result in more detail in the full version.

Instance optimality in other fields. The study of instance-optimal algorithms extends beyond computational geometry. A prime example is sorting under partial information. Given a partial order O over a set X and an unknown linear extension L, the goal is to sort X using a minimum number of comparisons, where a comparison queries the order of a pair $(x, y) \in X$ under L. This is directly analogous to retrievals. Kahn and Saks [16] introduced an exponential-time algorithm that is instance-optimal in the number of comparisons. Since then, significant progress has been made on improving the runtime of such algorithms [5, 15], culminating in near-linear time algorithms that are instance-optimal in both runtime and comparisons [12,23,24]. Another example is the bidirectional shortest path problem. Haeupler, Hladík, Rozhoň, Tarjan, and Tětek [11] show an algorithm that finds the shortest path between two nodes s and t using an instance-optimal number of edge-weight comparisons.

Table 1 Previous results that compute a *sorting*, Pareto *front*, convex *hull*, or *Delaunay* triangulation. The bottommost result is simultaneous and independent work from ours.

Shapes	Overlap	Structures	Preprocess	Retrievals	Reconstruction time	Source
Unit disks	No	many	$O(n \log n)$	$O(n)$	$O(n)$	[8, 9, 13, 17, 19, 25]
Disks	k	Delaunay	$O(n \log n)$	$O(n)$	$O(n \log k)$ rand.	[4]
Lines	Yes	hull	$O(n \log n)$	$O(n)$	$(n \cdot \alpha(n))$ rand.	[10]
Intervals	Yes	sorting	$O(n \log n)$	$\Theta(r(F,P))$	$O(r(F,P) \cdot \log n)$	[21]
Axis rect.	No	front	$O(n \log n)$	$\Theta(r(F,P))$	$O(r(F,P) \cdot \log n)$	[22]
Smooth	Yes	front, hull	$O(\text{poly}\, n)$	$\Theta(r(F,P))$	$O(r(F,P) \cdot \text{poly}\, n)$	[2]
$O(1)$-gons	Yes	hull	$O(n \log^3 n)$	$\Theta(r(F,P))$	$O(r(F,P) \cdot \log^3 n)$	Thm 33
k-gons	Yes	hull	$O(kn \log^3 n)$	$\Theta(r(F,P))$	$O(r(F,P) \cdot k \log^3 n)$	Thm 34
$[1,k]$-disks	No	hull	$O(kn \log^3 n)$	$\Theta(r(F,P))$	$O(r(F,P) \cdot k \log^3 n)$	Full version
Unit disks	k	hull	$O(kn \log^3 n)$	$\Theta(r(F,P))$	$O(r(F,P) \cdot k \log^3 n)$	Full version
Unit disks	k	hull	$O(k^3 n)$	$O(w(F))$	$O(k^3 \cdot w(F))$	[18]

Our contributions. We present the first algorithm for convex hull reconstruction that is instance-optimal while only requiring polylogarithmic time per retrieval. If F is a family of n constant-complexity simple polygons, we preprocess F in $O(n \log^3 n)$ time and reconstruct the convex hull of any $P \sim F$ using $\Theta(r(F,P))$ retrievals and $O(r(F,P) \log^3 n)$ total time. Our approach applies to simple k-gons with a linear factor in k in space and time, to pairwise disjoint disks with radii in $[1,k]$, and to unit disks of ply k (see Table 1). Compared to previous approaches for convex hulls that achieve optimal retrieval counts [2], we exponentially improve the runtime per retrieval from polynomial to polylogarithmic. Compared to near-linear time algorithms for convex hulls (or, Delaunay triangulations) [8–10, 13, 18, 19, 25], we significantly reduce the number of retrievals used and broaden the generality of input families. The latter comes at a cost of a polylogarithmic slowdown if $r(F,P)$ is linear in n.

Organisation. In Section 2, we provide some preliminaries. Then, in Section 3, we present our general algorithm for reconstructing the convex hull and prove its optimality (Theorem 18). In Section 4, we present a data structure for simple k-gons and give an instance-optimal algorithm that uses $O(kn)$ space and $O(r(F,P) \cdot k^2 \log^3(kn))$ time (Theorem 33). In the full version we improve the dependency on k to near-linear (Theorem 34). In the full version, we study pairwise disjoint disks with a radius in $[1,k]$, or, unit disks with ply k.

2 Preliminaries

Recall that an imprecise point set is defined as a family of geometric regions $F = (R_1, \ldots, R_n)$, and a realisation $P \sim F$ is defined as a sequence P of n points with $p_i \in R_i$. We explicitly allow for duplicate points in P and do not assume general position. The algorithm has two phases [13]: a *preprocessing phase*, where only F is available, and a *reconstruction phase*, in which we may perform a *retrieval* on any $R_i \in F$, replacing R_i with the point p_i. To maintain generality, we require only that each R_i is a point or a closed, connected, bounded region whose boundary is a simple closed piecewise-C^1 curve. Non-point regions must have connected interiors and coincide with the closure of their interiors; see Figure 2. Next, we define retrievals, convex hulls, vertices, edges, reconstruction strategies, and instance-optimality.

Retrievals. Given a family of regions F, a *retrieval* operation on a non-point region R_i replaces R_i with its associated point p_i (thereby updating F). We write $F \odot_P A$ for the family resulting from retrieving all regions in $A \subseteq F$. We denote by $F - A$ the family obtained by removing all regions in A from F. This removal is used in our analysis when we identify regions whose corresponding points cannot appear on the convex hull.

Critical assumptions. Since all regions in F are closed, we claim that we must assume P may contain duplicates and require that the convex hull of a point sequence includes all collinear and duplicate points. To illustrate, let F consist of n identical unit squares, and suppose $P \sim F$ includes four points forming the corners of one square. A lucky algorithm might retrieve these four points first and construct the convex hull [14]. If we assumed general position or excluded duplicates from the convex hull, this algorithm could then conclude that all remaining points cannot lie on the convex hull and terminate early. However, such behaviour cannot be guaranteed against an adversary. Since all regions are indistinguishable, an adversary may simply give these four points last. Alternatively, if F consisted of open regions, we could assume general position and exclude duplicates.

Upper quarter convex hull. We focus on the *upper quarter convex hull* of P. The other three quarters can be constructed analogously, and the full convex hull results from combining them. Let $\mathrm{CH}(P)$ denote the upper quarter convex hull: the boundary of the smallest convex set containing $P \cup \{(-\infty, -\infty), (+\infty, -\infty)\}$. We assume that $(R_{n+1}, R_{n+2}) = (\{(-\infty, -\infty)\}, \{(+\infty, -\infty)\})$, with corresponding points (p_{n+1}, p_{n+2}). These are always included in F and P. For any family of regions F, we denote by $\mathrm{OCH}(F)$ the *upper quarter outer convex hull*: the boundary of the smallest convex area enclosing all regions in F. Thus, $\mathrm{CH}(P)$ and $\mathrm{OCH}(F)$ represent convex hulls over point sequences and region families, respectively. As convex hulls define boundary curves, we may say that a point $p \in \mathbb{R}^2$ lies *on*, *inside*, or *outside* $\mathrm{CH}(P)$ or $\mathrm{OCH}(F)$.

Vertices and edges. We define convex hull vertices and edges of $\mathrm{OCH}(F)$. Although intuitive in simpler settings, these concepts require care due to the generality of F.

▶ **Definition 1.** *A point $p \in R$ is a* vertex *of a region R if p lies on the boundary of R and every open line segment through p contains a point not on the boundary of R. For any $R \in F$, $V(R)$ is the (infinite) set of vertices of R.*

Intuitively, the edges of $\mathrm{OCH}(F)$ connect successive vertices lying on its boundary. Due to potential overlaps between regions, vertices may coincide. As illustrated in Figure 2(c), a single edge may therefore be considered to connect different pairs of overlapping vertices. To avoid such degeneracy, we define $V(F) = \bigcup_{R \in F} V(R)$ as a set, and define edges robustly:

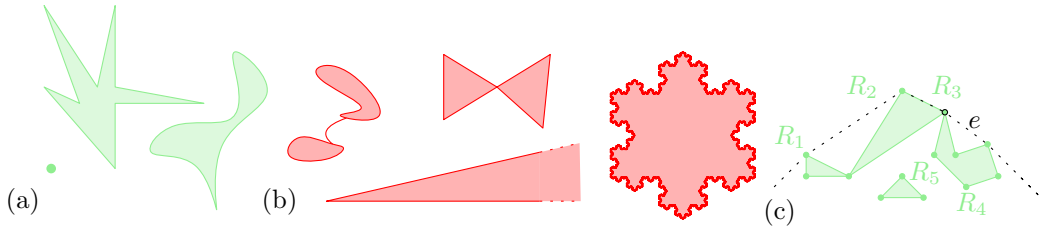


Figure 2 (a) Regions may have bends and sharp corners, and, be points. (b) Regions may not be unbounded, non-simple, connected by a line, or have infinitely many sharp corners. (c) Vertices between regions may coincide, and point regions (R_3) may coincide with vertices of other regions.

▶ **Definition 2.** *We define an* edge (s,t) *of* $\mathrm{OCH}(F)$ *to be a pair of distinct vertices in* $V(F)$ *where the subcurve of* $\mathrm{OCH}(F)$ *from* s *to* t *contains no vertices in* $V(F)$ *other than* s *and* t*.*

Note that a disk has infinitely many vertices but no edges under these definitions. We also define when a region appears on the outer convex hull:

▶ **Definition 3.** *We say a region* R appears on $\mathrm{OCH}(F)$ *if there is a point* p *on* $\mathrm{OCH}(F)$ *with* $p \in R$*. (Observe that we may always pick* p *to be a vertex of* R*.)*

Reconstruction algorithms. We will retrieve regions until all realisations $P' \sim F$ yield the same ordering of points along their convex hull. We formalise this via a partial order:

▶ **Definition 4.** *Given* $P \sim F$*, let* $\preceq(\mathrm{CH}(P))$ *be the partial order on* $[n]$ *induced by the left-to-right traversal of* $\mathrm{CH}(P)$*, i.e. for* $p_a, p_b \in P$*, we have* $a \prec b$ *if and only if* p_a *and* p_b *both lie on* $\mathrm{CH}(P)$ *and* p_a *lies strictly to the left of* p_b*.*

We say a family F is *finished* if all realisations $P_1, P_2 \sim F$ satisfy $\preceq(\mathrm{CH}(P_1)) = \preceq(\mathrm{CH}(P_2))$. A *reconstruction algorithm* retrieves some $B \subseteq F$ such that $F \odot_P B$ is finished. Let $r(F,P)$ denote the minimum number of retrievals needed by any such algorithm.

▶ **Observation 5.** *For any fixed* $P \sim F$*, let* $A \subset F$ *be a smallest subset of* F *such that* $F \odot_P A$ *is finished. Then* $|A|$ *is a tight lower bound for* $r(F,P)$*.*

A reconstruction algorithm is *instance-optimal* if for all inputs (F,P) it retrieves $\Theta(r(F,P))$ regions before F is finished. We distinguish two types of reconstruction algorithms:

- A *reconstruction strategy* is any such algorithm, analysed only by the number of retrievals.
- A *reconstruction program* is a reconstruction algorithm executed on a pointer machine or RAM, and is analysed by both retrievals and instructions.

The previous instance-optimal reconstruction strategy. Bruce, Hoffmann, Krizanc, and Raman [2] present an instance-optimal reconstruction strategy for region families F consisting of points and closed piecewise-C^1 regions. They do not present a corresponding reconstruction program. When F consists of disks or polygons, a reconstruction program may be derived, albeit with unspecified polynomial-time costs per retrieval. We further discuss their algorithm in the full version. We rely on a key concept of their paper, i.e. a witness.

▶ **Definition 6.** *Let* F *be a family of regions and let* $A \subseteq F$*. Any* $A \subset F$ *is a* witness *if for all* $P' \sim (F - A)$*, the family* $A \cup P'$ *is not finished.*

We observe that a reconstruction strategy is instance-optimal if, at each step, it retrieves all regions from a constant-size witness set of F.

2.1 Explicit dynamic planar convex hull

We rely on a data structure to maintain the upper quarter convex hull of a dynamic point set. Overmars and van Leeuwen [20] provide the best-known solution, achieving deterministic update time $O(\log^2 n)$. Their data structure, known as the *Partial Hull Tree* (PHT), is now textbook material. We follow their terminology:

▶ **Definition 7** (PHT from [20])**.** *Given a two-dimensional point set* S*, the* Partial Hull Tree *stores* S *in a leaf-based balanced binary tree* T *(sorted by* x*-coordinates). For each interior node* $\nu \in T$*, denote by* $\mathrm{CH}(\nu)$ *the (upper quarter) convex hull of all points in the leaves of the subtree rooted at* v*. For each* $\nu \in T$*, with children* (x,y) *and parent* w*, the PHT stores:*

- *The* bridge $e(\nu)$ *(the unique edge in* $\mathrm{CH}(\nu)$ *that is not in* $\mathrm{CH}(x)$ *and* $\mathrm{CH}(y)$*), and*
- *a* concatenable queue $\mathbb{E}^*(\nu)$ *(a balanced tree of the edges in* $CH(\nu) - CH(w)$*).*

We denote by $\mathbb{E}(\nu)$ *a balanced tree over* $CH(\nu)$ *and by* $V[\nu]$ *the vertices in the subtree of* ν*.*

Note that for the root ρ of T, $\mathbb{E}(\rho) = \mathbb{E}^*(\rho)$.

3 Witnesses and an instance-optimal reconstruction strategy

We introduce a new geometric classification over the edges of $\mathrm{OCH}(F)$. This leads to an alternative instance-optimal reconstruction strategy, detailed in Algorithm 1. Our results hold in full generality for any family F of points and closed regions whose boundaries are simple, closed, piecewise-C^1 curves. In particular, each region is connected and bounded, and any non-point region has a connected interior equal to its closure; see Figure 2. We define a classification over vertex pairs (s,t), which we apply to edges of $\mathrm{OCH}(F)$. However, we formulate the classification for arbitrary vertex pairs, as this generality will be useful for our data structures. We say that two regions $R_a, R_b \in F$ are *strictly vertically separated* if there exists a vertical line ℓ where R_a and R_b lie in opposite open half-planes defined by ℓ.

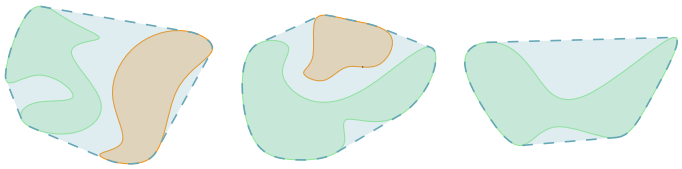


Figure 3 Three examples of a band (blue) between regions (green, brown).

▶ **Definition 8** (Band. Fig. 3)**.** *For any pair of regions* (R_a, R_b) *(where* $a = b$ *is allowed) we define* $\mathrm{band}(R_a, R_b)$ *as the area enclosed by the (full) outer convex hull of* $\{R_a, R_b\}$*.*

▶ **Definition 9** (Fig. 4)**.** *We say* $(s,t) \in V(R_a) \times V(R_b)$ *with* $s \neq t$ *and* $R_a, R_b \in F$*, is:*

- canonical *in* F *if the following holds for all* $x \in \{s,t\}$*. Either:*
  - x *is exclusively a vertex of point regions, or*
  - x *is a vertex of a unique region that is not a point region.*
- dividing *in* F *if* (s,t) *is canonical and either:*
  - $a = b$*, or*
  - R_a, R_b *are strictly vertically separated.*
- occupied *in* F *if* (s,t) *is dividing and:*
  - $\mathrm{band}(R_a, R_b)$ *contains a vertex in* $V(F - R_a - R_b) - \{s\} - \{t\}$*.*

In our reconstruction strategy, we retrieve regions until no non-canonical, non-dividing, or occupied edges remain. This alone does not guarantee that F is finished, so we introduce a final characterisation based on convex chains.

▶ **Definition 10** (Spanning chain. Fig. 5)**.** *A convex chain* $C = (q, r, \ldots, s, t)$ *is* spanning *in* F *if the following three conditions hold:*

- *all edges on* C *are dividing in* F*,*
- $q \in V(R_a)$*,* $r, s \in V(R_b)$*,* $t \in V(R_c)$ *where* R_b *intersects the inside of* $\mathrm{OCH}(\{R_a, R_c\})$*,*
- *all vertices of* C *between and including* r *and* s *are in* $V(R_b)$*.*

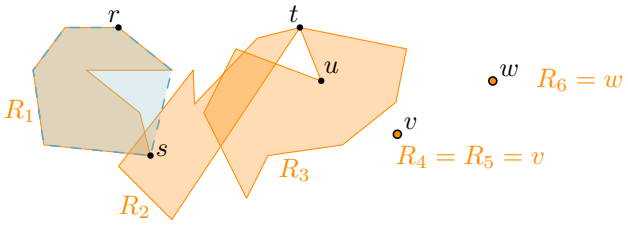


Figure 4 We illustrate all (sub) cases of vertex pairs corresponding to Definition 9:

- All pairs that include t are non-canonical, as t is a vertex of more than one non-point region.
- All pairs in $\{r, s, u, v, w\}^2$ are all canonical, as v is exclusively a vertex of two point regions.
- The pair (u, v) is non-dividing, as R_3 and R_4 are not vertically separated.
- The pairs (s, u), (r, u), (r, v), (s, v), and any pair in $\{r, s, u, v\} \times \{w\}$ are dividing because their regions are vertically separated. (r, s) is dividing because they are vertices of the same region.
- The pair (r, s) is not occupied since $\text{band}(R_1, R_1)$ only contains vertices of R_1. The pair (v, w) is not occupied since $\text{band}(R_4, R_6)$ only contains vertices v and w.
- The pair (u, w) is occupied as $V(F - R_3 - R_6)$ contains the vertex $t \in R_2$. The pairs (s, u), (r, u), (r, v), (s, v), (r, w), and (s, w) are also occupied.

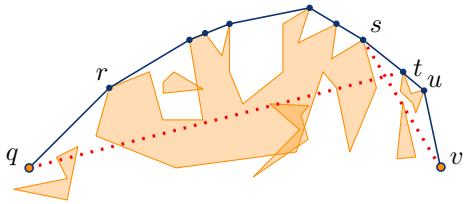


Figure 5 The convex chain from q to t is spanning. The chain from r to v is not spanning.

We use $\text{OCH}(\{R_a, R_c\})$ rather than $\text{band}(R_a, R_c)$ in Definition 10 to ensure meaningful behaviour even when R_a and R_c are point regions (since $\text{band}(R_a, R_c)$ would then be empty). Algorithm 1 processes edges of $\text{OCH}(F)$ by a priority based on Definitions 9 and 10:

$$\text{non-canonical} \gg \text{canonical but non-dividing} \gg \text{occupied} \gg \text{in spanning chain.}$$

In the following two subsections, we first show that each case in Algorithm 1 retrieves a witness. Then, we prove that if Algorithm 1 is instance-optimal.

3.1 Proving that Algorithm 1 retrieves only witnesses

The first case of Algorithm 1 considers a non-canonical edge, which implies that there is a point on $\text{OCH}(F)$ that lies in two regions that are not both point regions.

▶ **Lemma 11** (Proof in the full version). *Let $s \in \text{OCH}(F)$ be a point that lies in two regions R_a, R_b that are not both point regions. Then $\{R_a, R_b\}$ is a witness.*

The second case of Algorithm 1 considers an edge (s, t) of $\text{OCH}(F)$ that is canonical but not dividing in F. Let $s \in V(R_a)$ and $t \in V(R_b)$. Because (s, t) is canonical but not dividing it holds that $a \neq b$ and R_a and R_b are distinct regions that are not strictly vertically separated. The following lemma implies that $\{R_a, R_b\}$ is indeed a witness.

▶ **Lemma 12** (Proof omitted). *Let $R_a, R_b \in F$ with $R_a \neq R_b$ appear on $\text{OCH}(F)$. If R_a and R_b are not strictly vertically separated, then $\{R_a, R_b\}$ is a witness.*

Algorithm 1 A data structure friendly reconstruction strategy: Reconstruct(F).

if $\exists$ an edge (s,t) of OCH(F) that is non-canonical in F **then**
  Let $s \in V(R_a)$ and $s \in V(R_b)$ (or t) where R_a is not a point region
  Reconstruct($F \odot_P \{R_a, R_b\}$)
else if $\exists$ an edge (s,t) of OCH(F) that is canonical but not dividing in F **then**
  Let $s \in V(R_a)$ and $t \in V(R_b)$ for $a \neq b$
  Reconstruct($F \odot_P \{R_a, R_b\}$)
else if $\exists$ an edge (s,t) of OCH(F) that is occupied in F **then**
  Let $s \in V(R_a)$ and $t \in V(R_b)$ for $a \neq b$
  Let R_i with $R_i \neq R_a$ and $R_i \neq R_b$ be a region with $V(R_i) \cap \text{band}(R_a, R_b) \neq \emptyset$
  Reconstruct($F \odot_P \{R_a, R_b, R_i\}$)
else if $\exists$ a contiguous subchain $C = (q, r, \ldots, s, t)$ of OCH(F) spanning in F **then**
  Let R_a, R_b, R_c be as in the definition of spanning chains
  Reconstruct($F \odot_P \{R_a, R_b, R_c\}$)
else
  Return OCH(F)
end if

The third case of Algorithm 1 considers an occupied edge. The next lemma shows that any choice of region that has a vertex in the occupied band gives rise to a witness.

▶ **Lemma 13** (Proof omitted). *Let (s,t) be an edge of* OCH(F) *that is occupied in F. Let $s \in V(R_a)$ and $t \in V(R_b)$ for $a \neq b$. Let R_i be a region not equal to R_a or R_b with a vertex $q \in V(R_i) \cap \text{band}(R_a, R_b)$. Then $\{R_a, R_b, R_i\}$ is a witness.*

The final case of Algorithm 1 considers any remaining spanning subchains of OCH(F).

▶ **Lemma 14** (Proof omitted). *Let $(q, r, \ldots, s, t)$ be a contiguous subchain of* OCH(F) *that is spanning in F. Let R_a, R_b, R_c be as in Definition 10. Then $\{R_a, R_b, R_c\}$ is a witness.*

3.2 Proving that Algorithm 1 is instance-optimal

We now prove that Algorithm 1 is instance-optimal. That is, the algorithm retrieves the minimal number of regions (up to constant factors) necessary to determine the ordering of points on the convex hull, for any realisation $P \sim F$. We first observe that Algorithm 1 only terminates when F is *terminal*, meaning: every edge of OCH(F) is dividing, no edge of OCH(F) is occupied, and no contiguous subchain of OCH(F) is spanning.

For the remainder of this section, let A denote the multiset of regions that appear on OCH(F). If F is terminal, the regions in A satisfy the following structure:

▶ **Lemma 15.** *If F is terminal, then all regions in A are strictly vertically separated from each other, with the exception of pairs of regions that are duplicates of the same point.*

Proof. Walk along OCH(F) from left to right, considering edges (s,t) with $s \in R_a$, $t \in R_b$, $a \neq b$. As F is terminal, this edge is dividing, hence R_a lies strictly to the left of R_b. All previously encountered regions lie strictly left of R_a, hence also strictly to the left of R_b. Moreover, this edge is not occupied, so R_b is disjoint from all regions not equal to R_b. ◀

With Lemma 15, we define for each region R that appears on OCH(F) its left (or right) neighbour as the first region on OCH(F) that lies *strictly* left (or right) of R. Note that two duplicate point regions on OCH(F) have the same neighbours. Next, we show that the points corresponding to regions in A always appear on the convex hull (Lemma 16), and that no points corresponding to regions in $F - A$ appear on the convex hull (Lemma 17).

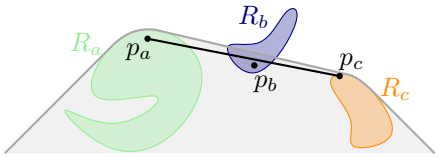


Figure 6 p_b must be above $p_a p_c$, as otherwise there is a spanning contiguous subchain of $\mathrm{OCH}(F)$.

▶ **Lemma 16.** *If F is terminal, then for any $P' \sim A$ each point in P' lies on* $\mathrm{CH}(P')$.

Proof. Let $P' \sim A$ and $R_b \in A - \{(-\infty,-\infty),(\infty,-\infty)\}$ be arbitrary. Let R_a and R_c be the left and right neighbour of R_b on $\mathrm{OCH}(F)$. Observe that these neighbours are well-defined as $(-\infty,-\infty)$ and $(\infty,-\infty)$ lie on $\mathrm{OCH}(F)$. Suppose by contradiction that p_b lies below the segment $p_a p_c$. Then R_b intersects the inside of $\mathrm{OCH}(\{R_a, R_c\})$, see Figure 6, so there is a contiguous subchain of $\mathrm{OCH}(F)$ that is spanning in F. This contradicts F being terminal. Since R_b was arbitrary, this shows that any three consecutive points in P' form a convex-down angle. Hence, P' spans a convex polygon, so all points in P' appear on $\mathrm{CH}(P')$. ◀

▶ **Lemma 17.** *If F is terminal, then for any $P \sim F$ only points of regions in A lie on* $\mathrm{CH}(P)$.

Proof. Let $P' \sim A$ be arbitrary. Suppose by contradiction that there is a region $R_i \in F - A$ with a point $p_i \in R_i$ on or outside of $\mathrm{CH}(P')$. Since the inside of $\mathrm{CH}(P')$ is convex, we may require p_i to be an extreme point of R_i, so $p_i \in V(R_i)$. By Lemma 15 and Lemma 16, there exist consecutive regions $R_a, R_b \in A$ for which p_i lies on or above $p_a p_b$. As $p_a p_b$ lies fully inside $\mathrm{band}(R_a, R_b)$, the point p_i lies in or directly above $\mathrm{band}(R_a, R_b)$. We make a case distinction, where p_i lies directly above R_b (or R_a), or, on the upper tangent of R_a, R_b.

Suppose first that p_i lies directly above a point $q \in R_b$. Let R_a, R_c be the left and right neighbour of R_b on $\mathrm{OCH}(F)$, respectively. Let r be the point on $\mathrm{OCH}(F)$ directly above p_i. Then $r \in \mathrm{band}(R_b, R_c)$ (or $r \in \mathrm{band}(R_a, R_b)$), hence also $p_i \in \mathrm{band}(R_b, R_c)$ as bands are convex and p_i lies on qr. Hence, F is not terminal. Similarly, p_i being directly above a point in R_a implies that F is not terminal. Furthermore, p_i cannot lie above the upper tangent of R_a, R_b since $R_i \notin A$. This shows that p_i does not lie above $\mathrm{band}(R_a, R_b)$, hence $p_i \in \mathrm{band}(R_a, R_b)$, so F is not terminal. ◀

▶ **Theorem 18.** *Algorithm 1 is an instance-optimal reconstruction strategy.*

Proof. First, we show that when Algorithm 1 terminates, the set F is finished. Let F be terminal. Let $A \subseteq F$ be the multiset of regions that appear on $\mathrm{OCH}(F)$. Let $P' \sim A$ and $Q' \sim F - A$ be arbitrary. By Lemma 17, no points in Q' are on $\mathrm{CH}(P' \cup Q')$. By Lemma 16, the points in P' all appear on $\mathrm{CH}(P' \cup Q')$. Moreover, together with Lemma 15, this implies that for all $P_1 \sim A$ and $P_2 \sim A$, $\preceq(\mathrm{CH}(P_1)) = \preceq(\mathrm{CH}(P_2))$. So, F is finished. Hence, Algorithm 1 is a correct reconstruction strategy.

We now show instance-optimality. By Lemmas 11, 12, 13 and 14, the algorithm retrieves a constant-size witness in each iteration. Let $X_1, \ldots, X_k$ be the sets of non-point regions of these witnesses. Since the witnesses have constant size, Algorithm 1 does $O(k)$ retrievals. A retrieval turns a region into a point region, hence the X_i are pairwise disjoint. On the other hand, if $(F \odot_P A)$ is finished for some $A \subseteq F$, then A needs to contain at least one element from each X_i (by the definition of witness). As the X_i are disjoint, this forces $|A| \geq k$. Hence, Algorithm 1 is instance optimal by Observation 5. ◀

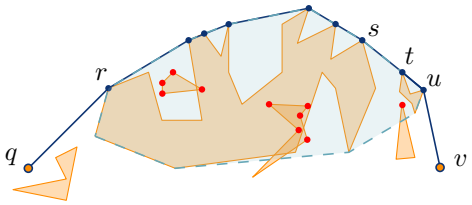


Figure 7 The convex chain from q to v is hit in F since (s,t) is occupied due to the red vertices.

4 Regions that are simple k-gons

We present a reconstruction program that executes our strategy from Algorithm 1 in polylogarithmic time per retrieval. To this end, we restrict F to a family of n (possibly overlapping) simple polygons, each with at most k vertices. Recall that $V(F) = (v_1, \ldots, v_m)$ denotes the set of all $m \in O(kn)$ *distinct* vertices in F. After each retrieval, we update F by replacing R_i with p_i, and update V by deleting $O(k)$ vertices from V and adding p_i to V.

▶ **Observation 19.** *For any family of regions F,* $\mathrm{OCH}(F)$ *has the same edges as* $\mathrm{CH}(V(F))$.

Consider a PHT T of $V(F)$ (Definition 7). After one retrieval, let T' be the updated PHT. We define the *recourse set* as the set symmetric difference of all bridges in T and all bridges in T'. *Recourse* is defined as the maximum size, across all possible retrievals, of any recourse set. A PHT has $O(\log m)$ recourse per update, and hence $O(k \log m)$ recourse per retrieval.

Augmenting the Partial Hull Tree. Within each leaf of T, corresponding to a vertex $v \in V(F)$, we maintain two doubly linked lists. The *point region list* stores all point regions that are v. The *non-point region list* stores the other regions $R \in F$ where $v \in V(R)$. Recall that Algorithm 1 retrieves regions corresponding to the endpoint of edges that are:

$$\text{non-canonical} \gg \text{canonical but non-dividing} \gg \text{occupied} \gg \text{in spanning chain.}$$

For every node $\nu \in T$ with bridge $e(\nu) = (s,t)$, we maintain pointers to the leaves containing s and t, along with Boolean flags indicating whether $e(\nu)$ is canonical or dividing. An update to F may make $O(n)$ edges occupied. So instead, we maintain a different property (Fig. 7):

▶ **Definition 20.** *A convex chain $C = (q, r, \ldots, s, t, \ldots, u, v)$ of vertices is* hit *in F if:*

- *all edges of C are dividing,*
- *the edge (s,t) is occupied in F,*
- *$s \in V(R_a), t \in V(R_b)$ and $a \neq b$, and*
- *$q \notin V(R_a)$, $r, \ldots, s \in V(R_a)$, $t, \ldots, u \in V(R_b)$ and $v \notin V(R_b)$.*

Recall Definition 7, where for $\nu \in T$, $\mathbb{E}(\nu)$ denotes a balanced binary tree on $\mathrm{CH}(\nu)$. The PHT stores in ν a *concatenable queue* $\mathbb{E}^*(\nu)$ which is $\mathbb{E}(\nu)$ minus all edges that are in $\mathbb{E}(w)$ where w is the parent of ν. We further augment the data structure by maintaining four balanced trees associated with $\mathbb{E}^*(\nu)$, where edges are ordered by their appearance in $\mathbb{E}^*(\nu)$:

1. a tree $\Lambda^*(\nu)$ storing all edges $(s,t) \in \mathbb{E}^*(\nu)$ that are non-canonical in F,
2. a tree storing all edges $(s,t) \in \mathbb{E}^*(\nu)$ that are canonical and non-dividing in F,
3. a tree storing all contiguous subchains of $\mathbb{E}^*(\nu)$ that are spanning in F,
4. a tree storing all contiguous subchains of $\mathbb{E}^*(\nu)$ that are hit in F.

We only give the first tree a name since all others are maintained in similar fashion. We denote by Λ_ν a balanced tree on all non-canonical edges in $\mathbb{E}(\nu)$.

▶ **Observation 21** (by Definition 9). *For any canonical bridge (s,t) in T, there exists a unique area $R(s)$ that contains s. $R(s)$ is either a point, or a unique region in F and can be found in $O(1)$ time by checking the regions lists stored at one of the leaves that (s,t) points to.*

We store for every region in F their maximum and minimum x-coordinate, this way we can test if a pair of regions is vertically separated in constant time.

Candidate chains. Our data structure maintains for each $\nu \in T$, a collection of special edges and special chains. We observe that these chains have a special structure:

▶ **Definition 22.** *Let $\nu \in T$ and consider $\mathbb{E}(\nu)$ (not $\mathbb{E}^*(\nu)$). Any contiguous subchain $C = (q, s, \ldots, r)$ of $\mathbb{E}(\nu)$ is a* candidate chain *if all edges of C are dividing in F and:*
- *$C = (q, s, r)$ with $q \in V(R_a)$, $s \in V(R_b)$, $r \in V(R_c)$, and $q, r \notin V(R_b)$, or,*
- *all interior vertices of C are in $V(R_b)$ and $q, r \notin V(R_b)$,*
  (note that this implies that all interior vertices are not in $V(R_x)$ for $x \neq b$).

▶ **Observation 23.** *Any subchain of $\mathbb{E}(\nu)$ that is* spanning *in F is also a candidate chain.*

▶ **Observation 24.** *Any subchain $(q, r, \ldots, s, t, \ldots, u, v)$ of $\mathbb{E}(\nu)$ is* hit *in ν only if $(q, r, \ldots, s, t)$ and $(s, t, \ldots, u, v)$ are candidate chains.*

▶ **Lemma 25.** *Let $\nu \in T$. There are at most two candidate chains that contain an edge (s,t) of $\mathbb{E}(\nu)$ and, given $\mathbb{E}(\nu)$ and our data structure, we can find these in $O(k)$ time.*

Proof. Suppose two candidate chains C and C' share (s,t). By the definition of candidate chains, (s,t) must be the final edge of C and the first edge of C'. First, identify the up to two length-3 chains in $\mathbb{E}(\nu)$ containing (s,t). Check each such chain in $O(1)$ time for the dividing property by consulting the Boolean flags. If all edges are dividing, we apply Observation 21 to test whether the chain satisfies the conditions of a candidate chain.

Next, consider candidate chains of length ≥ 4 in which (s,t) is an interior edge. Check, in $O(1)$ time via Observation 21, whether both s and t lie in a unique region R_b. If so, perform an in-order traversal of $\mathbb{E}(\nu)$ – moving up to $O(k)$ steps left from s and up to $O(k)$ steps right from t – to find the maximal contiguous chain with all edges dividing and interior vertices contained in $V(R_b)$. The chain ends when we encounter an edge whose endpoints do not lie entirely in $V(R_b)$. This identifies the unique candidate chain C of length ≥ 4 containing (s,t) as an interior edge. To find chains where (s,t) is the first or last edge, apply the above procedure to the edges immediately before and after (s,t). ◀

Update time. After each retrieval, we update bridges for all nodes along $O(k)$ root-to-leaf paths in T. For each bridge (s,t), we update its leaf pointers with constant overhead. Corollaries 27, 28, 30, and 32 show how to determine whether a bridge is canonical, dividing, part of a spanning chain, or part of a hit chain in F, respectively, in $O(k \log^2 m)$ time. Since the recourse per retrieval is $O(k \log m)$, this yields an overall update time of $O(k^2 \log^3 m)$.

▶ **Observation 26** (by Definition 9). *For any node $\nu \in T$, its bridge $e(\nu) = (s,t)$ is* canonical *in F if and only if both leaves containing s and t meet one of the following conditions:*
- *The non-point region list is empty, or,*
- *The point region list is empty and the region list contains exactly one region.*

▶ **Corollary 27.** *We can dynamically maintain, in $O(k \log^2 m)$ time, for each node $\nu \in T$ a balanced binary tree $\Lambda^*(\nu)$ of all edges $(s,t) \in \mathbb{E}^*(\nu)$ that are non-canonical in F.*

Proof. Test, using Observation 26, whether any bridge is canonical in constant time. Since the recourse is $O(k \log m)$, the set of all bridges can be updated without incurring asymptotic overhead.

To maintain $\Lambda^*(\nu)$ for each node $\nu \in T$, we follow the technique from [20] for concatenable queues. For any node ν with child x, compute $\Lambda(x)$ from $(\Lambda(\nu), e(\nu), \Lambda^*(x))$ in $O(\log m)$ time by splitting $\Lambda(\nu)$ at the bridge $e(\nu)$, and joining the result with $\Lambda^*(x)$. Traverse all $O(k)$ updated root-to-leaf paths, using the split operation, in $O(k \log^2 m)$ total time. Then, traverse the paths bottom-up. At each node ν with children x and y, we have access to the newly updated trees $\Lambda(x)$ and $\Lambda(y)$. Compute $(\Lambda(\nu), \Lambda^*(x), \Lambda^*(y))$ by splitting $\Lambda(x)$ and $\Lambda(y)$ at the endpoints of the new bridge $e(\nu)$ and joining the result. ◀

▶ **Corollary 28.** *We can dynamically maintain in $O(k \log^2 m)$ time for each node $\nu \in T$ a balanced binary tree on all edges $(s,t) \in \mathbb{E}^*(\nu)$ that are canonical but non-dividing in F.*

Proof. The set of bridges has $O(k \log m)$ recourse. Once a bridge is known to be canonical, we apply Observation 21 to determine the unique areas $R(s)$ and $R(t)$ containing s and t. We then test in constant time whether $R(s)$ and $R(t)$ are vertically separated, using stored x-coordinates. The tree is maintained using the same procedure as in Corollary 27. ◀

▶ **Lemma 29.** *For any $\nu \in T$, given $e(\nu) = (s,t)$ and $\mathbb{E}(\nu)$, we may find the at most two contiguous subchains C of $\mathbb{E}(\nu)$ with $s,t \in C$ that are spanning in F in $O(k \log m)$ time.*

Proof. Apply Lemma 25 to obtain all $O(1)$ candidate chains in $\mathbb{E}(\nu)$ containing (s,t). Let $C = (q, \ldots, r')$ be such a chain. To test whether C is spanning in F, use Observation 21 to find the unique regions $R(q)$, R_b, and $R(r')$. Construct $\mathrm{OCH}(R(q) \cup R(r'))$ in $O(k \log k)$ time and perform a convex hull intersection test with R_b in $O(\log m)$ time using the algorithm of Chazelle [6]. The chain is spanning if and only if this test returns true. ◀

▶ **Corollary 30.** *We can dynamically maintain in $O(k^2 \log^2 m)$ time for each node $\nu \in T$ a balanced binary tree of all subchains of $\mathbb{E}^*(\nu)$ that are spanning in F.*

Proof. Whenever the set of leaves of a node $\nu \in T$ changes (which occurs for $O(k \log m)$ nodes), invoke Lemma 29 on $e(\nu)$ to maintain all chains that are spanning in F (and contiguous subchains of $\mathrm{CH}(\nu)$ but not of $\mathrm{CH}(x)$ or $\mathrm{CH}(y)$). The balanced binary tree associated to $\mathbb{E}^*(\nu)$ can be maintained in an identical manner as previous corollaries. ◀

▶ **Lemma 31.** *For any node $\nu \in T$ and any edge $(x,y) \in \mathbb{E}(\nu)$, we can find all contiguous subchains of $\mathbb{E}(\nu)$ that contain (x,y) and are hit in F in $O(k \log^2 m)$ time.*

Proof. Apply Lemma 25 to find all candidate chains containing (x,y). Then, for each candidate chain, consider the adjacent edge and apply the lemma again to construct all possible hit chains. For each resulting subchain, test whether the middle edge $\mathrm{i}(s,t)$ is occupied as follows: Construct $\mathrm{band}(R(s), R(t))$ in $O(k \log k)$ time [7]. Temporarily remove $R(s)$ and $R(t)$ from F, and delete s and t from $V(F)$, in $O(k \log^2 m)$ time. Update the corresponding PHT T' without updating auxiliary data. The hull $\mathrm{CH}(V(F) \setminus \{s,t\})$ is stored at the root of T'. By intersection testing between $\mathrm{band}(R(s), R(t))$ and $\mathrm{CH}(T')$ in $O(\log m)$ time [6], we can test if any hit chain is occupied. ◀

▶ **Corollary 32.** *We can dynamically maintain in $O(k^2 \log^3 m)$ time for each node $\nu \in T$ a balanced binary tree of all subchains of $\mathbb{E}^*(\nu)$ that are hit in F.*

Proof. As with Corollary 30, we invoke Lemma 31 on $O(k \log m)$ nodes where the leaf set changes. For each, we update the associated trees using split and join operations. ◀

▶ **Theorem 33.** *Let F be a family of n simple polygons, where each region in F has at most k vertices. We can preprocess F using $O(kn)$ space and $O(k^2 n \log^3(kn))$ time, such that given $P \sim F$ we reconstruct* $\mathrm{CH}(P)$ *using $O(kn)$ space and $O(r(F,P)k^2 \log^3(kn))$ time.*

Proof. By Corollaries 27, 28, 30, and 32, we may maintain our augmented Partial Hull Tree in $O(k^2 \log^3(kn))$ time per retrieval. This data structure maintains at its root four balanced trees storing all: non-canonical edges, canonical but non-dividing edges; subchains of $\mathrm{OCH}(F)$ that are spanning in F, or hit in F. We observe that Algorithm 1 only considers occupied edges if all edges on $\mathrm{OCH}(F)$ are dividing. Then, any occupied edge corresponds to a hit subchain. Thus, we may immediately use these trees to execute Algorithm 1. This procedure performs $O(r(F,P))$ retrievals and so the theorem follows. ◀

In the full version, we note that our approach has two bottlenecks. First, the algorithm from Lemma 25 may "skip" over $O(k)$ edges to find the largest candidate chain. By applying a technique similar to skip-lists, we improve its running time by a factor k. Secondly, our approach frequently uses a subroutine where for any two regions $R(s)$ and $R(t)$ we construct $\mathrm{band}(R(s), R(t))$ in $O(k \log k)$ time. We show that by storing for each region R_i its convex hull, this construction time becomes polylogarithmic and we obtain the following:

▶ **Theorem 34.** *Let F be a family of n simple polygons where each region in F has at most k vertices. We can preprocess F using $O(kn \log n)$ space and $O(kn \log^3(kn))$ time, such that given $P \sim F$ we reconstruct* $\mathrm{CH}(P)$ *using $O(r(F,P) \cdot k \log^3(kn))$ time.*

References

1. Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-optimal Geometric Algorithms. *Journal of the ACM (JACM)*, 2017.
2. Richard Bruce, Michael Hoffmann, Danny Krizanc, and Rajeev Raman. Efficient Update Strategies for Geometric Computing with Uncertainty. *Theory of Computing Systems (TOCS)*, 2005. doi:10.1007/s00224-004-1180-4.
3. Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing Imprecise Points for Delaunay Triangulation: Simplified and Extended. *Algorithmica*, 2011. doi:10.1007/s00453-010-9430-0.
4. Kevin Buchin and Wolfgang Mulzer. Delaunay Triangulations in $O(sort(n))$ Time and More. *Journal of the ACM (JACM)*, 2011. doi:10.1145/1944345.1944347.
5. Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël Jungers, and J. Ian Munro. Sorting under partial information (without the ellipsoid algorithm). *Combinatorica*, 33(6):655–697, December 2013. doi:10.1007/s00493-013-2821-5.
6. Bernard Chazelle and David P Dobkin. Detection is easier than computation. In *ACM Symposium on Theory Of Computing (STOC)*, 1980.
7. Mark de Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications, third edition.* Springer, 2008. doi:10.1007/978-3-540-77974-2.
8. Olivier Devillers. Delaunay Triangulation of Imprecise Points: Preprocess and Actually get a Fast Query Time. *Journal of Computational Geometry (JoCG)*, 2011. doi:10.20382/jocg.v2i1a3ff.ffinria-00595823.
9. William Evans and Jeff Sember. The Possible Hull of Imprecise Points. *Canadian Conference on Computational Geometry (CCCG)*, 2011. URL: https://www.cs.ubc.ca/~will/papers/possiblehull.pdf.
10. Esther Ezra and Wolfgang Mulzer. Convex Hull of Points Lying on Lines in $o(n \log n)$ Time after Preprocessing. *Computational Geometry*, 2013. doi:10.1016/j.comgeo.2012.03.004.

11 Bernhard Haeupler, Richard Hladík, Václav Rozhoň, Robert E Tarjan, and Jakub Tětek. Bidirectional dijkstra's algorithm is instance-optimal. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 202–215. SIAM, 2025. doi:10.1137/1.9781611978315.16.

12 Bernhard Haeupler, Richard Hladík, John Iacono, Václav Rozhoň, Robert E. Tarjan, and Jakub Tětek. *Fast and Simple Sorting Using Partial Information*, pages 3953–3973. SIAM, 2025. doi:10.1137/1.9781611978322.134.

13 Martin Held and Joseph Mitchell. Triangulating Input-constrained Planar Point Sets. *Information Processing Letters (IPL)*, 2008. doi:10.1016/j.ipl.2008.09.016.

14 Simon Kahan. A model for data in motion. In *ACM Symposium on Theory of Computing (STOC)*, pages 265–277, New York, NY, USA, 1991. Association for Computing Machinery. doi:10.1145/103418.103449.

15 Jeff Kahn and Jeong Han Kim. Entropy and sorting. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of Computing*, STOC '92, pages 178–187, New York, NY, USA, July 1992. Association for Computing Machinery. doi:10.1145/129712.129731.

16 Jeff Kahn and Michael Saks. Balancing poset extensions. *Order*, 1(2):113–126, June 1984. doi:10.1007/BF00565647.

17 Maarten Löffler and Wolfgang Mulzer. Unions of Onions: Preprocessing Imprecise Points for Fast Onion Decomposition. *Journal of Computational Geometry (JoCG)*, 2014. doi:10.1007/978-3-642-40104-6_42.

18 Maarten Löffler and Benjamin Raichel. Preprocessing disks for convex hulls, revisited. *arXiv preprint arXiv:2502.03633*, 2025. doi:10.48550/arXiv.2502.03633.

19 Maarten Löffler and Jack Snoeyink. Delaunay Triangulation of Imprecise Points in Linear Time after Preprocessing. *Computational Geometry*, 2010. doi:10.1016/j.comgeo.2008.12.007.

20 Mark H. Overmars and Jan van Leeuwen. Dynamically maintaining configurations in the plane (detailed abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 135–145. ACM, 1980. doi:10.1145/800141.804661.

21 Ivor van der Hoog, Irina Kostitsyna, Maarten Löffler, and Bettina Speckmann. Preprocessing Ambiguous Imprecise Points. *International Symposium on Computational Geometry (SoCG)*, 2019. doi:10.4230/LIPIcs.SoCG.2019.42.

22 Ivor van der Hoog, Irina Kostitsyna, Maarten Löffler, and Bettina Speckmann. Preprocessing imprecise points for the pareto front. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022.

23 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. Simpler optimal sorting from a directed acyclic graph. In *Symposium on Simplicity in Algorithms (SOSA)*. SIAM, 2025. doi:10.1137/1.9781611978315.26.

24 Ivor Van Der Hoog and Daniel Rutschmann. Tight bounds for sorting under partial information. In *Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2024. doi:10.1109/FOCS61266.2024.00131.

25 Marc van Kreveld, Maarten Löffler, and Joseph Mitchell. Preprocessing Imprecise Points and Splitting Triangulations. *SIAM Journal on Computing (SICOMP)*, 2010. doi:10.1137/090753620.

Faster Dynamic 2-Edge Connectivity in Directed Graphs

Loukas Georgiadis
University of Ioannina, Greece

Konstantinos Giannis
Gran Sasso Science Institute, L'Aquila, Italy

Giuseppe F. Italiano
LUISS University, Rome, Italy

Abstract

Let G be a directed graph with n vertices and m edges. We present a deterministic algorithm that maintains the 2-edge-connected components of G under a sequence of m edge insertions, with a total running time of $O(n^2 \log n)$. This significantly improves upon the previous best bound of $O(mn)$ for graphs that are not very sparse. After each insertion, our algorithm supports the following queries with asymptotically optimal efficiency:

- Test in constant time whether two query vertices v and w are 2-edge-connected in G.
- Report in $O(n)$ time all the 2-edge-connected components of G.

Our approach builds on the recent framework of Georgiadis, Italiano, and Kosinas [FOCS 2024] for computing the 3-edge-connected components of a directed graph in linear time, which leverages the minset-poset technique of Gabow [TALG 2016].

Additionally, we provide a deterministic decremental algorithm for maintaining 2-edge-connectivity in strongly connected directed graphs. Given a sequence of m edge deletions, our algorithm maintains the 2-edge-connected components in total time $n^{2+o(1)}$, while supporting the same queries as the incremental algorithm. This result assumes that the edges of a fixed spanning tree of G and of its reverse graph G^R are not deleted. Previously, the best known bound for the decremental problem was $O(mn \log n)$, obtained by a randomized algorithm without restrictions on the deletions.

In contrast to prior dynamic algorithms for 2-edge-connectivity in directed graphs, our method avoids the incremental computation of dominator trees, thereby circumventing the known conditional lower bound of $\Omega(mn)$.

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

Keywords and phrases Connectivity, dynamic algorithms, directed graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.26

Acknowledgements We thank the anonymous reviewers for their helpful comments.

1 Introduction

The design of dynamic graph algorithms is a classical area of research in theoretical computer science, where the input graph evolves through a sequence of updates, typically edge insertions and deletions. The goal of a dynamic algorithm is to update the solution to a problem more efficiently than recomputing it from scratch after each change. A problem is said to be *fully dynamic* if both insertions and deletions are allowed, and *partially dynamic* if only one type of update is permitted. The latter includes the *incremental* setting (only insertions) and the *decremental* setting (only deletions).

A fundamental problem in this domain is the computation and maintenance of edge-connected components in both undirected and directed graphs, driven by various practical and theoretical applications (see, e.g., [30]). Let $G = (V, E)$ be a strongly connected directed



© Loukas Georgiadis, Konstantinos Giannis, and Giuseppe F. Italiano;
licensed under Creative Commons License CC-BY 4.0



graph (digraph) with n vertices and m edges. An edge $e \in E$ is called a *strong bridge* if its removal disconnects the graph, i.e., $G \setminus e$ is no longer strongly connected. More generally, a subset $C \subseteq E$ is a *cut* if its removal disconnects the graph. If $|C| = k$, we refer to C as a k-sized cut of G. A directed graph is said to be k-edge-connected if it contains no $(k-1)$-cuts. Two vertices v and w are k-edge-connected, denoted $v \leftrightarrow_k w$, if there exist k edge-disjoint directed paths from v to w and k edge-disjoint directed paths from w to v. (Note that paths from v to w and from w to v need not be edge-disjoint with each other.) By Menger's Theorem [27], this is equivalent to requiring that every set of fewer than k edge deletions preserves strong connectivity between v and w. A *k-edge-connected component* of G is a maximal subset of vertices $U \subseteq V$ such that $u \leftrightarrow_k v$ for all $u, v \in U$. These components form a partition of V, since $\leftrightarrow_k$ is an equivalence relation [14].

Connectivity problems are significantly more challenging in directed graphs than in undirected ones (see, e.g., [8, 18, 23]). Until recently, it was known how to compute the k-edge-connected components of undirected graphs in linear time only for $k \leq 5$ [7, 10, 12, 20, 25, 28, 29, 33, 38]. In a very recent breakthrough, Korhonen [24] presented an $k^{O(k^2)}m$ time algorithm for computing the k-edge connected components of an undirected graph, which yields linear-time algorithms for any fixed k. In contrast, for directed graphs, linear-time algorithms are only known for $k \leq 3$ [13, 16, 33].

Despite significant progress in fully dynamic algorithms for several fundamental connectivity problems in undirected graphs (see, e.g., [19, 21, 22, 31, 37]), their directed counterparts remain substantially harder [4]. This difficulty is further underscored by conditional lower bounds [1, 17]. In particular, Abboud and Vassilevska [1] showed that any fully dynamic algorithm for maintaining whether a directed graph has more than two strongly connected components (SCCs) must incur $\Omega(m^{1-\epsilon})$ update or query time (for any constant $\epsilon > 0$) unless the Strong Exponential Time Hypothesis (SETH) fails. Due to such hardness results, much of the research has focused on partially dynamic scenarios. For the problem of dynamically maintaining the SCCs of a digraph, years of effort culminated in the following breakthrough results. For the decremental setting, Bernstein, Probst, and Wulff-Nilsen [4] developed a randomized algorithm (against an oblivious adaptive adversary) with $O(m \log^4 n)$ total expected time, while very recently van den Brand et al. [39] gave a deterministic algorithm with $m^{1+o(1)}$ total update time. For the incremental problem, also very recently, Chen et al. [6] gave a deterministic algorithm with $m^{1+o(1)}$ total update time. We note that the algorithms of [6, 39] explicitly maintain the SCCs, while [4] only supports queries of whether two vertices are strongly connected.

In this paper, we revisit the dynamic maintenance of 2-edge-connected components in directed graphs, a problem first explored by Georgiadis, Italiano, and Parotsidis [15], who presented an incremental algorithm with total time $O(mn)$ and space $O(m+n)$. After each insertion, their algorithm supports the following queries in asymptotically optimal time:

- *query*(v, w): Test in $O(1)$ time whether two vertices v and w are 2-edge-connected.
- *report*$()$: Report all 2-edge-connected components in $O(n)$ time.

Moreover, when the answer to a *query*(v, w) is negative, their algorithm returns in constant time a "witness", i.e., a strong bridge that appears in all paths from v to w or in all paths from w to v.

The decremental version was studied in [11], where a randomized algorithm with total running time $O(mn \log n)$ and space $O(n^2 \log n)$ was presented.

Our results

We present new deterministic, incremental and decremental algorithms for maintaining the 2-edge-connected components of a directed graph that significantly improve upon the prior time bounds for graphs that are not very sparse.

▶ **Theorem 1.** *We can maintain the 2-edge-connected components of a digraph with n vertices through a sequence of edge insertions in $O(n^2 \log n)$ total time. After each insertion, we can test in $O(1)$ time whether two vertices are 2-edge-connected and report the components in $O(n)$ time.*

We also achieve nearly the same asymptotic bound in the decremental setting, under certain assumptions on the edges that may be deleted.

▶ **Theorem 2.** *We can maintain the 2-edge-connected components of a strongly connected digraph with n vertices through a sequence of edge deletions in $n^{2+o(1)}$ total time, provided that the edges of a fixed spanning tree of G and a fixed spanning tree of the reverse graph G^R are not deleted. After each deletion, we can test in $O(1)$ time whether two vertices are 2-edge-connected and report the components in $O(n)$ time.*

The bound stated in Theorem 2 assumes that we maintain SCCs decrementally using the algorithm of van den Brand et al. [39]. If instead we use the algorithm of Bernstein, Probst, and Wulff-Nilsen [4], we obtain a randomized $O(n^2 \log^4 n)$-time algorithm that supports constant-time *query*(v, w) queries, but does not support *report*().

Both our incremental and our decremental algorithms require $O(n^2)$ space. Our results build on the recent framework of Georgiadis, Italiano, and Kosinas [13], which computes the 3-edge-connected components of a digraph in linear time using Gabow's *minset-poset* technique [8] to represent all minimum edge-cuts of a digraph. From these results, it follows that the 2-edge-connected components of a digraph G can be identified as the strongly connected components of two specially constructed *labeling graphs*. Although these labeling graphs can have size $O(mn)$, we show how to maintain a compact representation of size $O(n^2)$ that can be efficiently updated in the dynamic setting.

Our techniques are fundamentally different from those of [11, 15]. The algorithm of [15] relies on maintaining dominator trees incrementally, while [11] maintains the SCCs of $G \setminus v$ for every vertex v, using n instances of a decremental SCCs algorithm [26]. This also supports decremental dominator tree maintenance in $O(mn \log n)$ time and $O(n^2 \log n)$ space. Importantly, [11] showed that maintaining dominator trees incrementally or decrementally in total time $O((mn)^{1-\epsilon})$ (for some constant $\epsilon > 0$) is not possible unless the OMv Conjecture [17] fails. This bound applies even to algorithms that do not explicitly maintain the dominator tree but can answer parent queries. Consequently, both [11] and [15] are subject to this hardness barrier, whereas our new algorithms are not. However, unlike [15], our algorithms do not provide a witness edge when the answer to a *query*(v, w) is negative.

2 Preliminaries and notation

We assume that the reader is familiar with standard graph terminology. All graphs in this paper are directed, i.e., an edge $e = (u, v)$ in digraph G is directed from u, the *tail* of e, to v, the *head* of e. Also, we allow G to have multiple edges between the same pair of vertices. To simplify our bounds, we will assume that G may have no more than two copies of each edge, so that the number of edges is $m = O(n^2)$. (Notice that adding more than two copies of the same edge (u, v) does not affect 2-edge-connectivity.) If C is a set of edges, then C^R denotes

the set of the reversed edges from C. We also let $G^R = (V, E^R)$ denote the reverse graph of $G = (V, E)$, i.e., the digraph that results from G after reversing the orientation of all edges. For a graph or a set of edges F, we use $V(F)$ to denote the set of the endpoints of the edges in F. If F consists of a single edge e, we may simply write $V(e)$ (which denotes the set of the endpoints of e). Also, for a graph F, we use $E(F)$ to denote the set of edges in F.

Let $G = (V, E)$ be a digraph. For any $S, T \subseteq V$, we denote the set of edges whose tail is in S and their head in T by $E(S, T) = \{(u, v) \mid (u, v) \in E, u \in S, v \in T\}$. We denote the set of outgoing edges from a set $S \subseteq V$ to the rest of the graph by $\delta(S) = E(S, V \setminus S)$, and the number of such edges by $out(S) = |\delta(S)|$. Similarly, we denote the set of incoming edges from the rest of the graph to S by $\rho(S) = E(V \setminus S, S)$ and the number of such edges by $in(S) = |\rho(S)|$. When S contains only a single vertex u, we slightly abuse notation and write $out(u), in(u)$ instead of $out(\{u\}), in(\{u\})$.

For a set of edges $E' \subseteq E$ and a set of vertices $S \subseteq V$, $\rho_{E'}(S)$ denotes the set of edges in E' that enter S from $V \setminus S$.

2.1 Flow graphs, dominators, and bridges

A *flow graph* is a directed graph with a distinguished *start vertex* s such that every vertex is reachable from s. Let $G = (V, E)$ be a strongly connected graph. Since G is strongly connected, all vertices are reachable from s and reach s, so we can view both G and G^R as flow graphs with start vertex s. To avoid ambiguities, throughout the paper, we will denote those flow graphs respectively by G_s and G_s^R. Let G_s be a flow graph with start vertex s. An edge (u, v) is a *bridge* of G_s if all paths from s to v include (u, v).[1] A vertex u is a *dominator* of a vertex v (u *dominates* v) if every path from s to v in G_s contains u. The dominator relation is reflexive and transitive. Its transitive reduction is a rooted tree, the *dominator tree* D: u dominates v if and only if u is an ancestor of v in D. The dominator tree and the bridges of a flow graph can be computed in linear time [2, 5].

2.2 Cuts

A cut is a partition of the vertices of a graph into two disjoint subsets S, T. A cut determines a cut-set $E(S, T)$, since the removal of these edges makes all vertices in T unreachable from vertices in S. We may use the term "cut" interchangeably to denote either the partition (S, T) or the set of edges $E(S, T)$. (The meaning will always be clear from the context.) We say that a cut (S, T) is a k-sized cut if $out(S) = in(T) = k$; then we say S is a k-out set and T is a k-in set. A cut (S, T) is trivial if $|S| = 1$ or $|T| = 1$. A cut C *separates vertex* s *from vertex* t if all paths from s to t contain an edge in C. We refer to such a cut C as an *s-t cut*. Any partition of the vertices into two sets S and $T = V \setminus S$, such that $s \in S$ and $t \in T$ naturally defines an s-t cut. We also refer to the partition (S, T) as an s-t cut of G. The size of this cut is equal to $|E(S, T)|$, i.e., the number of edges directed from S to T. An *s-t mincut* is a cut (S, T) of minimum size such that $s \in S$ and $t \in T$. Consider a flow graph G_s and a k-in set T such that $s \notin T$. We call T a k-sized s-cut of G. Then, all paths from s to T contain an edge in $\rho(T)$. Clearly, any cut of G is an s-cut of G_s or an s-cut of G_s^R, which allows us to focus only on the s-cuts of G.

[1] Throughout the paper, to avoid confusion, we use consistently the term *bridge* to refer to a bridge of a flow graph and the term *strong bridge* to refer to a strong bridge in the original graph.

2.3 Trees and fundamental cycles

Let T be a rooted tree. Throughout, we assume that the edges of T are directed away from the root. For each directed edge (u, v) in T, we say that u is a parent of v (and we denote it by $t(v)$) and that v is a child of u. Every vertex except the root has a unique parent. If there is a (directed) path from vertex v to vertex w in T, we say that v is an ancestor of w and that w is a descendant of v, and we denote this path by $T[v, w]$. If $v \neq w$, we say that v is a proper ancestor of w and that w is a proper descendant of v, and denote by $T(v, w]$ the path in T to w from the child of v that is an ancestor of w.

A spanning tree T of a flow graph G_s with start vertex s is rooted at s and contains a unique path from s to any other vertex. For an edge $e \notin T$, we let $T(e)$ denote the fundamental cycle of e in T, i.e., the cycle that is formed in T when we add edge e, ignoring edge directions. Let $e = (u, v) \notin T$, and let z be the nearest common ancestor of u and v in T. Then $T(e) \setminus e$ consists of two directed paths, one from z to u, and another from z to v. (One of these paths may consist of a single vertex.) For a set $S \subseteq V(S)$, we let $LCA(S)$ denote the lowest common ancestor in T of all vertices in S.

Let G_s be a flow graph and let T be a spanning tree of G_s rooted at s. We let $\mathcal{N}$ denote the set of *non-tree edges* of G_s, i.e., $\mathcal{N} = E(G_s) \setminus T$. For any vertex $v \in G_s$, we let $\rho_{\mathcal{N}}(v)$ denote the set of non-tree edges that are incoming to v.

2.4 Minimum 1-in sets

Let G_s be a strongly connected digraph with a fixed start vertex s. A set of vertices S is called a 1-in set if $s \notin S$ and $in(S) = |E(V \setminus S, S)| = 1$. For every vertex $v \neq s$ that is contained in a 1-in set, we let $M(v)$ denote the (inclusion-wise) minimum 1-in set that contains v. We call this the M-set of v. Note that $M(v)$ is well-defined due to the submodularity of cuts. Specifically, we have the following.

▶ **Lemma 3.** *Let S and S' be two 1-in sets that contain a vertex $v \neq s$. Then, $S \cap S'$ is also a 1-in set.*

Proof. We have that $S \cap S'$ and $S \cup S'$ contain v but not s. By the submodularity of cuts, we have:

$$in(S \cap S') + in(S \cup S') \leq in(S) + in(S') \tag{1}$$

Since S and S' are 1-in sets, the right hand side of (1) equals 2. Since $S \cap S'$ and $S \cup S'$ are cuts that separate v from s, we have $in(S \cap S') \geq 1$ and $in(S \cup S') \geq 1$. Now (1) implies that $in(S \cap S') = 1$. This shows that $S \cap S'$ is a 1-in set. ◀

If v is not contained in a 1-in set, then we let $M(v) = V(G)$. We use $M_R(v)$ to denote the M-set of v in G_s^R. The importance of considering the M-sets (in both G_s and G_s^R) is demonstrated in the following proposition.

▶ **Proposition 4** ([13]). *Let G be a strongly connected digraph with a fixed start vertex s. Then, for any two vertices u and v, we have $u \leftrightarrow_2 v$ if and only if $M(u) = M(v)$ and $M_R(u) = M_R(v)$.*

According to Proposition 4, in order to compute the 2-edge-connected components of G, it is sufficient to compute the M-sets in G_s and G_s^R. As in [13], our approach is to compute the partition of $V(G)$ into the sets of vertices that have same M-set in G_s, and the sets of vertices that have the same M-set in G_s^R. It follows by Gabow [8], that this partition (w.r.t.

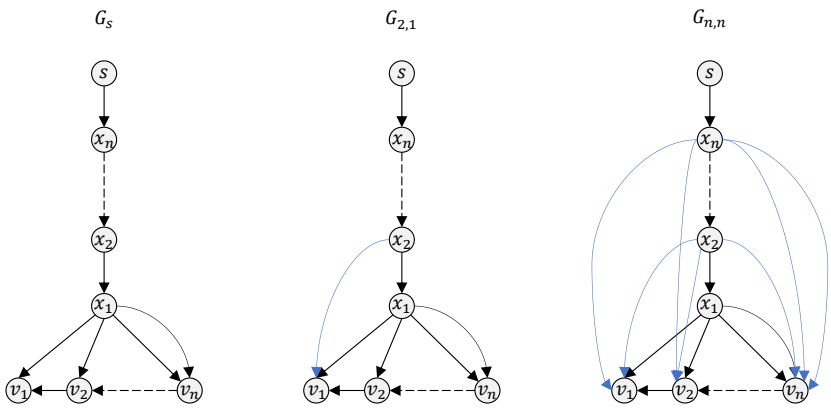


Figure 1 Example of flow graph and an insertion sequence that elicits $\Theta(n^2)$ changes of the $M(v_i)$ sets. Each graph $G_{k,l}$ is formed by adding the edges (x_i, v_j) for $1 \leq l \leq j$ and $2 \leq i \leq k$. Hence, $G_{k,l+1}$ results from $G_{k,l}$ after inserting the edge (x_k, v_{l+1}), and $G_{k+1,1}$ results from $G_{k,n}$ after inserting the edge (x_{k+1}, v_1). In G_s, we have $M(v_i) = \{x_1, v_i, v_{i+1}, \ldots, v_n\}$, for $1 \leq i \leq n$. In $G_{k,l}$, $M(v_i) = \{x_k, x_{k-1}, \ldots, x_1, v_i, v_{i+1}, \ldots, v_n\}$, for $1 \leq i \leq l$.

either G_s or G_s^R) corresponds to the strongly connected components of a *labeling graph* LG. So, our goal here is to show how to maintain this information efficiently in the incremental or decremental setting. Note that there are insertion sequences (or deletion sequences) that can cause $O(n^2)$ changes of the M-sets. See Figure 1.

First, we need to extend the definition of M-sets to edges of G_s as follows. Let e be an edge such that there is a 1-in set that contains both endpoints of e. Then $M(e)$ denotes the edge-set of the induced subgraph of the minimum 1-in set that contains both endpoints of e. To obtain the minimum 1-in set of each vertex v, [8] uses an augmented version of G_s, defined as follows. For every vertex $v \neq s$ of G_s, we introduce a new vertex v', two parallel edges of the form (v, v'), and two parallel edges of the form (v', v). Let us call $G_s^+ = (V, E)$ the resulting graph. Then, it follows that for every vertex $v \neq s$ of G_s, we have that $M(v)$ corresponds in a natural way to $M((v, v'))$ (in G_s^+).

Since, by Lemma 3, the $M(e)$ sets are closed under intersection, they admit a poset representation. For any edge $e \in E$, let $[e] = \{f : M(f) = M(e)\}$. We define the following relation on the edges of G_s: for $e, f \in E$, let $[e] \succ [f]$ if and only if $M(f) \subset M(e)$. Hence, $(\{[e]\}, \succ)$ forms a poset that represents all the minimum M-sets.

3 Computing minimum 1-in sets via Gabow's minset poset

Let $G = (V, E)$ be a strongly connected digraph, and let s be an arbitrary vertex of G. We view G as a flow graph with start vertex s. Recall that a 1-in set X is a set of vertices such that $X \subseteq V \setminus s$ and $in(X) = |E(V \setminus X, X)| = 1$. Consider the flow graph G_s, and let T be a spanning tree rooted at s. For simplicity, sometimes we slightly abuse notation and refer to T as the set of tree edges. For any set of vertices X, we denote by $T[X]$ the subgraph of T induced by X. We say that X is *cut* by T if $\rho(X) \subseteq T$ (that is, the only edges that enter X are tree edges of T), and X is *cospanned* by T if $T[X]$ is a tree. The next characterization follows from [8].

▶ **Lemma 5** ([8]). *Let G_s be a flow graph with start vertex s, and let T be a spanning tree of G_s. Any set of vertices $X \subseteq V \setminus s$ is a 1-in set of G_s if and only if it is cut and cospanned by T.*

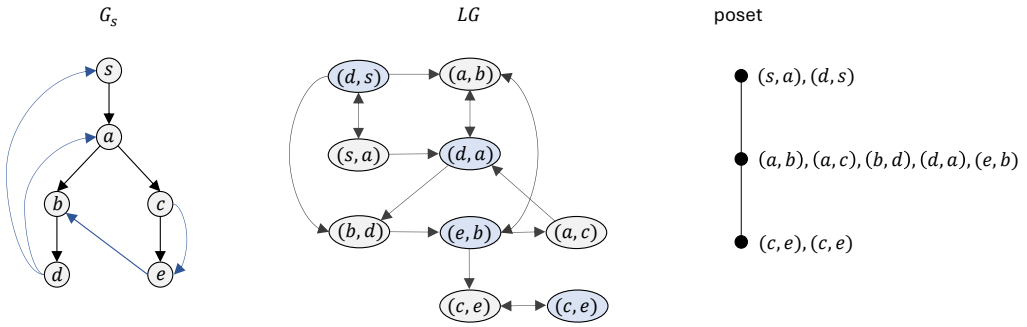


Figure 2 A strongly connected flow graph G_s, its corresponding labeling graph LG, and the minimum 1-in set poset. In G_s the tree edges are black and the non-tree edges are blue. In LG, the vertices that correspond to tree edges are gray and the vertices that correspond to non-tree edges are blue. We have $M((a,b)) = \{(a,b),(a,c),(b,d),(d,a),(e,b),(c,e),(c,e)\}$.

According to Lemma 5, if X is a 1-in set of G_s, then $T[X]$ is a tree with root r and the only edge entering X is the edge $(t(r), r) \in T$.

3.1 Labeling function and labeling graph LG

Gabow [8] defines a *labeling graph* LG with the property that the strongly connected components of LG correspond to the edges of G_s that have the same M-set. For any vertex $v \neq s$, there is at most one incoming edge to v that can be the incoming edge to $M(v)$, and the rest have the property that both of their endpoints also belong to $M(v)$. Thus, the idea is to assign a label to every edge e, which is essentially a set of pointers to edges that participate in the same M-set as e. Thus, the M-sets are given as the reachability sets of various edges with respect to this labeling. However, we cannot afford to explicitly compute the M-sets of all vertices, as their total size can be quadratic to $|V(G_s)|$.

In [8], the vertex set $V(LG)$ of LG consists of the edges of G, and the edges $(f,g) \in E(LG)$ are defined by a labeling function $\mathcal{L}_c : E \mapsto 2^E$. In our case, this labeling function becomes

$$\mathcal{L}_c(e) = \begin{cases} \rho_{\mathcal{N}}(u) \cup \rho_{\mathcal{N}}(v) & e = (u,v) \in T \\ T(e) & e \in \mathcal{N} \end{cases} \tag{2}$$

This function implies a labeling graph LG that has vertex set E (i.e., one vertex for each edge of G), and edges (e,f) where $e \in E$ and $f \in L(e)$. For $e, f \in E$, we say that f is a successor of e if there is a path from e to f in LG. The important property of the labeling graph LG is that the minimum set $M(e)$ of each edge e is equal to the set of all successors of e in LG [8]. This implies the following key proposition. (See Figure 2.)

▶ **Proposition 6** ([8, 13]). *Let G_s be a strongly connected flow graph with start vertex s, and let e and f be any two edges.*

(a) *If e is contained in a 1-in set, then $M(e) = M(f)$ if and only if e and f are strongly connected in LG.*

(b) *If e is not contained in a 1-in set, then e and f are strongly connected in LG if and only if f is also not contained in a 1-in set.*

By Proposition 6, we have that any two vertices v and w are 2-edge-connected if and only if (v,v') and (w,w') are strongly connected both in the labeling graph LG^+ of G_s^+ and in the labeling graph LG_R^+ of $(G_s^+)^R$.

Note that each tree edge $e = (v, w)$ has out-degree equal to $|\rho_{\mathcal{N}}(v)| + |\rho_{\mathcal{N}}(w)|$ in LG, while each non-tree edge f has out-degree equal to the number of tree edges in $T(f)$. Hence, LG has m vertices and $O(mn)$ edges. Nevertheless, Gabow [8] showed that the nodes $[e]$ of the corresponding poset $(\{[e]\}, \succ)$, which form a compact representation of these M-sets sufficient for our purposes, can be computed in $O(m)$ time by a clever implementation of an algorithm for computing the SCCs [7] of LG. (We remark that the node-finding algorithm of [8] does not compute the complete poset, which has $O(n)$ vertices and $O(n^2)$ edges.) The key idea is to use appropriate data structures, based on set merging [9], to avoid generating all edges of LG. This algorithm critically depends on a specific ordering of operations within the SCC algorithm, which makes it unsuitable for use in incremental or decremental settings.

4 Incremental algorithm

In this section, we present our incremental algorithm for maintaining the 2-edge-connected components of a digraph G. We consider, first, the case where G is strongly connected, and let T be a spanning tree of the corresponding flowgraph G_s, for an arbitrarily chosen start vertex s. Our algorithm operates on a modified labeling graph, denoted by $\widehat{LG}$, with the following properties: (i) $\widehat{LG}$ contains $O(n)$ vertices and $O(n^2)$ edges, (ii) the SCCs of $\widehat{LG}$ correspond to the poset nodes $[x]$ of the minimum 1-in sets of G_s, where $x \in V(G) \cup T$, and (iii) we can efficiently maintain $\widehat{LG}$ through a sequence of edge insertions.

4.1 Modified labeling graph

The modified labeling graph $\widehat{LG}$ is defined as follows. The vertex set of $\widehat{LG}$ consists of the tree edges of T and the vertices of G, i.e., $V(\widehat{LG}) = T \cup V(G)$. The edge set is defined by the following modified labeling function $\widehat{\mathcal{L}}_c : T \cup V(G) \mapsto 2^{T \cup V(G)}$:

$$\widehat{\mathcal{L}}_c(x) = \begin{cases} \{u, v\} & x = (u, v) \in T \\ \{e \in T : \exists f \in \rho_{\mathcal{N}}(v) \text{ such that } e \in T(f)\} & x = v \in V(G) \end{cases} \tag{3}$$

Hence, $(x, y) \in E(\widehat{LG})$ if and only if $y \in \widehat{\mathcal{L}}_c(x)$. Note that both LG and $\widehat{LG}$ are bipartite graphs, since any edge connects a tree edge $e \in T$ with a non-tree edge $f \in \mathcal{N}$ in the former, and a tree edge $e \in T$ with a vertex $v \in V(G)$ in the latter. While LG has m vertices and $O(mn)$ edges, $\widehat{LG}$ has $2n - 1$ vertices and $O(n^2)$ edges.

▶ **Lemma 7.** *For any two edges $f, g \in T$, g is reachable from f in LG if and only if g is reachable from f in $\widehat{LG}$.*

Proof. Suppose g is reachable from f in LG. Let P be a path from f to g in LG. Since LG is bipartite, the length of P is even. Consider two consecutive edges (x, y) and (y, z) on P, such that $x, z \in T$ and $y \in \mathcal{N}$. Also, let $x = (u, v)$. Then, by the definition of $\mathcal{L}_c$, $y \in \rho_{\mathcal{N}}(u) \cup \rho_{\mathcal{N}}(v)$, and $z \in T(y)$. Then, by the definition of $\widehat{\mathcal{L}}_c$, $\widehat{LG}$ contains (x, u) and (x, v), and either (u, z) or (v, z). In both cases, x reaches z in $\widehat{LG}$. Hence, it follows by induction on the length of P that if g is reachable from f in LG then g is also reachable from f in $\widehat{LG}$.

We show the contrapositive by similar arguments. Suppose g is reachable from f in $\widehat{LG}$. Let P be a path from f to g in $\widehat{LG}$. Since $\widehat{LG}$ is bipartite, the length of P is even. Consider two consecutive edges (x, y) and (y, z) on P, such that $x, z \in T$ and $y \in V(G)$. Also, let $x = (u, v)$. Then, by the definition of $\widehat{\mathcal{L}}_c$, $y \in \{u, v\}$, $z \in T(e)$ where $e \in \rho_{\mathcal{N}}(y)$. Then, by the definition of $\mathcal{L}_c$, LG contains the edges (x, e) and (e, z), and so x reaches z in LG. Hence, it follows by induction on the length of P that if g is reachable from f in $\widehat{LG}$ then g is also reachable from f in LG. ◀

An alternative but equivalent definition of $\widehat{LG}$ can be obtained by applying a sequence of vertex contractions to LG. Specifically, for each vertex $v \in V(G)$, we contract all non-tree edges in $\rho_{\mathcal{N}}(v)$ and subsequently eliminate any duplicate edges. While this may appear more intuitive, we adopt the original formulation as it enables the construction of $\widehat{LG}$ in $O(n^2)$ total time.

▶ **Corollary 8.** *For any two edges $f, g \in T$, $[f] = [g]$ if and only if f and g are strongly connected in $\widehat{LG}$.*

By Proposition 4 and Corollary 8, to determine whether two vertices u and v are 2-edge-connected in G, it suffices to check whether the edges (u, u') and (v, v') in the augmented graph G_s^+ are strongly connected in both the modified labeling graph of G_s^+ and that of its reverse $(G_s^+)^R$. The following lemma shows that, in fact, it is not necessary to work with the augmented graph explicitly.

▶ **Lemma 9.** *For any two vertices $u, v \in V(G)$, (v, v') is reachable from (u, u') in the modified labeling graph $\widehat{LG}^+$ of G_s^+ if and only if v is reachable from u in $\widehat{LG}$.*

Proof. Suppose $\widehat{LG}^+$ contains a path P from (u, u') to (v, v'). Since $\widehat{LG}^+$ is bipartite, and because the only non-tree edges entering (u, u') in G_s^+, except its copy (u, u'), are in $\rho_{\mathcal{N}}(u)$, the next vertex on P after (u, u') must be u. Consider now the penultimate vertex w on P. Then $w \in V(G)$, and from the definition of $\widehat{\mathcal{L}}_c$ (equation (3)), there is a non-tree edge $e \in \rho_{\mathcal{N}}(w)$ such that $(v, v') \in T(e)$. But this is possible only for $w = v$ and $e = (v', v)$. Hence, P contains a path from u to v, and so, $\widehat{LG}$ also contains a path from u to v.

Now suppose that $\widehat{LG}$ contains a path P from u to v. From the definition of $\widehat{\mathcal{L}}_c$ (equation (3)), $\widehat{LG}^+$ contains an edge from (u, u') to u. Also, since $(v', v) \in \rho_{\mathcal{N}}(v)$ and $(v, v') \in T((v', v))$, $\widehat{LG}^+$ also contains an edge from v to (v, v'). Hence, $\widehat{LG}^+$ contains a path from (u, u') to (v, v'). ◀

▶ **Corollary 10.** *Let G be a strongly connected digraph with a fixed start vertex s. Then, for any two vertices u and v, we have $u \leftrightarrow_2 v$ if and only if u and v are strongly connected in the modified labeling graph $\widehat{LG}$ of G_s and in the modified labeling graph $\widehat{LG}_R$ of G_s^R.*

4.2 Incremental construction of $\widehat{LG}$

Here we describe how to construct $\widehat{LG}$ incrementally as edges are added to a strongly connected graph G with a designated start vertex s. The labeling graph is defined with respect to a fixed spanning tree T of the flow graph G_s, rooted at s. (Similarly, we have a fixed spanning tree T_R of G_s^R, rooted at s, that defines the labeling graph of G_s^R.)

We initialize $\widehat{LG}$ by inserting a vertex for each $v \in V(G)$ and for each tree edge $e \in T$. Also, for each tree edge $e = (u, v)$, we add to $\widehat{LG}$ the edges (e, u) and (e, v).

Let v be a vertex of G, and let e be a tree edge of T. We say that e is *covered* by v if there is an edge $f \in \rho_{\mathcal{N}}(v)$ such that $e \in T(f)$, i.e., if $e \in \widehat{\mathcal{L}}_c(v)$. For each vertex $v \in V(G)$, we maintain the set of tree edges that are covered by v, using the following simple fact.

▶ **Lemma 11.** *Let v be a vertex such that $|\rho_{\mathcal{N}}(v)| \geq 1$, and let T_v be the set of tree edges that are covered by v. Then, T_v is a tree, rooted at the lowest common ancestor in T of all the vertices in $V(\rho_{\mathcal{N}}(v))$.*

Proof. Consider an edge $f \in \rho_{\mathcal{N}}(v)$. Then, v is contained in the fundamental cycle $T(f)$, so T_v is connected. Since $T_v \subseteq T$, T_v is a tree. Let $root_v$ be the root of T_v. If $\rho_{\mathcal{N}}(v)$ contains a single edge $e = (u, v)$, then T_v is rooted at $LCA(V(e))$. Now suppose $|\rho_{\mathcal{N}}(v)| > 1$. Let

$e = (u, v)$ and $e' = (w, v)$ be two edges in $\rho_{\mathcal{N}}(v)$. Let $z = LCA(u, v)$ and $z' = LCA(w, v)$. Since both z and z' are ancestors of v in T, we have $LCA(z, z') \in \{z, z'\}$. We conclude that $root_v = LCA(V(\rho_{\mathcal{N}}(v)))$. ◀

Lemma 11 allows us to use simple data structures to maintain T_v for each vertex v. Specifically, we do not maintain T_v explicitly, but keep a bit vector b_v, indexed by the vertices of $V(G)$ that indicates the vertices participating in T_v. Also, we store the current root of T_v. During the construction we maintain the following invariant (I): For any vertex u, we have $b_v[u] = 1$ if and only if $(t(u), u)$ is covered by v. Thus, $b_v[u] = 1$ if and only if $(t(u), u) \in \widehat{\mathcal{L}}_c(v)$, which means that $\widehat{LG}$ contains the edge $(v, (t(u), u))$. So, intuitively, we can view the bit vectors b_v as forming an adjacency matrix of the vertices $v \in V(G)$ in $\widehat{LG}$.

Initially, we set $b_v[u] = 0$, for all vertices $u \in V(G)$, and set $root_v = v$. (The root of T_v is the only vertex in T_v that is not marked in b_v.) Furthermore, we need to be able to test the ancestor-descendant relation in T. There are several simple $O(1)$-time tests of this relation [34]. The most convenient one for us is to number the vertices of T from 1 to n in preorder and compute the number of descendants of each vertex v. We denote these numbers by $pre(v)$ and $size(v)$, respectively. Then v is a descendant of u if and only if $pre(u) \leq pre(v) < pre(u) + size(u)$.

Suppose now that an edge $e = (u, v)$ is added into G. Then, since T is a spanning tree of G_s, e is a new non-tree edge in $\rho_{\mathcal{N}}(v)$. So, we need to find the tree edges $f \in T(e)$ such that $\widehat{LG}$ does not contain the edge (v, f). Equivalently, we need to find the vertices in $T(e)$ that are not already marked in b_v. Let x be the lowest common ancestor of u and v in T. (Note that we only use x for reference and do not need to find it explicitly.) To find the relevant unmarked vertices, we traverse the part of the cycle $T(e)$ from u to x as follows. Let $y = u$ be the current vertex. While y is not an ancestor of v, we check if $b_v[y] = 1$. If not, then we set $b_v[y] = 1$, add the edge from v to $(t(y), y)$ in $\widehat{LG}$, and set $y = t(y)$. Otherwise, we let $y = root_v$. This procedure stops as soon as y is an ancestor of v. At this point, if $b_v[y] = 0$, then we set $root_v = y$. We repeat the same procedure for $y = v$, where we stop when y becomes an ancestor of u.

▶ **Lemma 12.** *The above procedure correctly updates $\widehat{LG}$ in $O(n^2)$ total time for all edge insertions.*

Proof. It is easy to verify that the procedure maintains invariant (I) correctly, because of Lemma 11. Also, the algorithm inserts an edge (v, e) into $\widehat{LG}$ if and only if e is covered by v, which is in accordance to the labeling function $\widehat{\mathcal{L}}_c$. Now we bound the total running time for the construction of $\widehat{LG}$ after all insertions. The total running time is dominated by the time we need to locate tree edges that are just covered by each insertion. Let $e = (u, v)$ be a newly added edge to G. The above procedure visits at most two vertices that are already marked in b_v. For all other visited vertices u, we have $b_v[u] = 0$ before the visit and $b_v[u] = 1$ after the visit, excluding $root_v$, which also is visited at most twice per added edge. Hence, the total time throughout the whole sequence of insertions is bounded by $O(n^2)$. ◀

4.3 Incremental computation of the 2-edge-connected components

Let G be a strongly connected graph that undergoes edge insertions. We chose an arbitrary start vertex s, and compute a spanning trees T of G and a spanning T_R of G^R, rooted at s. We maintain two instances of the modified labeling graph of Section 4.1, $\widehat{LG}$ that represents the minimum 1-in sets of G, and $\widehat{LG}_R$ that represents the minimum 1-in sets of G^R, using

the two fixed spanning trees T and T_R. When an edge (u, v) is inserted into G, we execute the update operation of Section 4.2 for $\widehat{LG}$ and for $\widehat{LG}_R$. Note that for $\widehat{LG}_R$, we search for tree edges of T_R that are covered by u.

We maintain the SCCs of $\widehat{LG}$ and $\widehat{LG}_R$ incrementally, by running on each labeling graph the incremental SCCs algorithm of Bender et al. [3] for dense graphs. For a digraph with n vertices, the algorithm of [3] runs in $O(n^2 \log n)$ total time. The modified labeling graphs $\widehat{LG}$ and $\widehat{LG}_R$ have $O(n)$ vertices and $O(n^2)$ edges, since during their construction we never add duplicate edges. By Lemma 12, we can construct them incrementally in $O(n^2)$ time, and the total time for maintaining their SCCs is also $O(n^2 \log n)$.

We turn now to the query operations *query*(v, w) and *report*(). Each SCC of $\widehat{LG}$ (and similarly of $\widehat{LG}_R$) is represented by a canonical vertex, and the partition of the vertices into SCCs is maintained through a disjoint set union data (DSU) structure [36, 35]. The DSU data structure supports the operation *unite*(p, q), which, given canonical vertices p and q, merges the SCCs containing p and q into one new SCC and makes p the canonical vertex of the new SCC. It also supports the query *find*(v), which returns the canonical vertex of the SCC containing v. Since we aim at constant time queries, we use such a data structure that can support each *find* operation in worst-case $O(1)$ time and any sequence of *unite* operations in total time $O(n \log n)$ [36]. This way, we can identify the canonical vertex of the auxiliary component containing a query vertex in constant time. Hence, by Corollary 10, we can answer *query*(v, w) in G also in constant time, by testing if u and v are strongly connected in both $\widehat{LG}$ and $\widehat{LG}_R$.

To answer a *report*() query, we create, for each vertex $v \in V(G)$, a label *label*$(v) = \langle c_v, c_v^R, v \rangle$, where c_v and c_v^R are the canonical vertices in the SCCs of $\widehat{LG}$ and $\widehat{LG}_R$, respectively, that contain v. As above, each of these canonical vertices is available in $O(1)$ time. We form a list L consisting of *label*(v) for all $v \in V(G)$, and sort them lexicographically in $O(n)$ time using bucket sorting. Then, in the sorted list L, the vertices of the same 2-edge-connected component appear consecutively, since they have the same canonical vertices in their labels. Therefore, we can report the 2-edge-connected components of G in $O(n)$ time.

4.4 Extension to general digraphs

Now we extend our incremental algorithm to general (not strongly connected) digraphs. We note that Proposition 6 requires us to use labeling graphs that correspond to strongly connected flow graphs. To that end, we construct a two-level data structure, that uses various instances of the incremental SCCs algorithm of Bender et al. [3], as mentioned in Section 4.3.

Let G be the input digraph subject to edge insertions. The top level of our data structure, that we refer to as ISC(G), maintains the strongly connected components of G with the use of the incremental SCCs algorithm of [3]. More precisely, ISC(G) maintains the SCCs of G, represented with a DSU data structure, and the condensation of G, denoted by $\bar{G}$, which is the directed acyclic graph that results from G after we contract each SCC into its canonical vertex. We note that [3] also maintains a topological ordering of $\bar{G}$, and when a new SCC is formed, it removes loops and duplicate edges. The bottom level of our data structure maintains the information about the 1-in sets and 1-out sets within each SCC C of G, in a structure I2EC(C). Specifically, for each strongly connected component C of G, with a designated start vertex s, we store a spanning tree T of $G[C]$ and a spanning tree T_R of $G^R[C]$, rooted at s. Also, we maintain the data structures of Sections 4.2 and 4.3.

Now we describe how to handle an edge insertion. Suppose a new edge (x, y) is inserted into G. If x and y are located the same SCC C of G, then we execute the insertion procedure for I2EC(C), described in Sections 4.2 and 4.3. Otherwise, we execute the insertion in

the ISC(G) data structure. Note that this operation inserts the edge $(\mathit{find}(x), \mathit{find}(y))$ into the condensation $\bar{G}$ of G. If this insertion does not create a cycle in $\bar{G}$ then we are done. Otherwise, ISC(G) finds a new SCC of $\bar{G}$, corresponding to a new SCC of G, that is contracted into some canonical vertex. As a result, we need to update the bottom-level structure for the involved components of G.

Let C be the new SCC of G. The data structure ISC(G) identifies all components $C_1, C_2, \ldots, C_k$ of G that are merged into C after the insertion of (x, y), along with the edges $E(C_i, C_j)$ that connect distinct components. (Each such edge (u, v) satisfies $u \in C_i$, $v \in C_j$, where C_i precedes C_j in a topological ordering of the components.) Without loss of generality, assume that C_1 is the largest of these components. We choose the canonical (start) vertex s of C to be the start vertex of C_1. We refer to this component C_1 as the *principal component of* C. Also, we refer to the vertices of C_1 as the *principal vertices of* C. The remaining vertices in $C_2, \ldots, C_k$ are the *secondary vertices of* C.

Now we describe how to construct the data structure I2EC(C) for the new component C. We describe only the construction of the structures for $G[C]$. The structures for the reverse graph $G^R[C]$ are updated similarly. First, we extend the spanning tree T_1 of $G[C_1]$, which is rooted at s, to a spanning tree T of $G[C]$ rooted at s, so that $T_1 \subseteq T$. To achieve this, it suffices to traverse the edges $E(C_i, C_j)$ and the edges of the two spanning trees that we maintain for each component $C_2, \ldots, C_k$. This is enough to construct T, since the two spanning trees of each C_i form a sparse strongly connected subgraph of $G[C_i]$. (Note that we cannot afford to traverse all edges of $G[C_i]$.) Once T is constructed, we traverse it to recompute $\mathit{pre}(v)$ and $\mathit{size}(v)$ for all $v \in T$. This enables testing the ancestor-descendant relation in T in $O(1)$ time.

Next, we need to update the structures that keep track of the covered edges of T for each vertex $v \in C$. To initialize these structures for the new component C, we maintain the structures b_v and root_v, for all principal vertices v as they are. Then, we insert the nontree edges $e = (u, v)$ such that u is a secondary vertex in C. For each secondary vertex u, we compute b_u and root_u from scratch. Hence, in effect we construct I2EC(C) by inserting the secondary vertices and their adjacent edges in the data structure I2EC(C_1) of the principal component.

▶ **Lemma 13.** *The above procedure correctly updates* I2EC(C)*, for all SCCs C of G in $O(n^2 \log n)$ total time over all edge insertions.*

Proof. The correctness of the algorithm follows from Lemma 12, and the fact that the top structure ISC(G) maintains the SCCs of G. Next, we bound the total running time for any sequence of edge insertions.

First, we bound the total running time required to update the spanning tree T of $G[C]$, for each SCC formed in G. Let C be a new SCC of G that is formed by merging the components $C_1, \ldots, C_k$, where C_1 is the principal component. Let n_i be the number of vertices in each component C_i, and let $m_{ij} = |E(C_i, C_j)|$ be the number of edges connecting C_i and C_j. Then, the construction of T takes time proportional to $\sum_{i=1}^{k} n_i + \sum_{1 \le i,j \le k} m_{ij} = n_C + \sum_{1 \le i,j \le k} m_{ij}$, where $n_C = \sum_{i=1}^{k} n_i$ is the number of vertices in the new component C. Note that the second term, i.e., the sum $\sum_{1 \le i,j \le k} m_{ij}$ is charged only once during the construction of all spanning trees. Hence, the overall construction time for all spanning trees is $O(n^2 + m) = O(n^2)$.

Next, we consider the time required to maintain the data structures for the tree edges covered by each vertex $v \in V(G)$. Note that for as long as v is a principal vertex in a component C, the total time spent to process the edges in $\rho_{\mathcal{N}}(v)$ and update b_v and root_v is $O(n_C)$. Next, we consider the contribution of each vertex as a secondary vertex. Every time

we merge a sequence of secondary components $C_2, \ldots, C_k$ with a principal component C_1, we charge $O(n_C)$ time to each secondary vertex v, since we recompute b_v and $root_v$ from scratch. Let $C_v^1, \ldots, C_v^\xi$ be the sequence of SCCs that contain v as a secondary vertex. Let n_v^i be the number of vertices in C_v^i just before it gets merged into a larger component. Then, $n_v^i \leq n_v^{i+1}/2$, for $1 \leq i < \xi$, and $n_v^\xi \leq n$. The time required to maintain the data structures for the tree edges covered by v in C_v^i is proportional to n_v^i. Hence, the total time for the whole sequence of components $C_v^1, \ldots, C_v^\xi$ is bounded by $\sum_{i=1}^{\xi} n_v^i \leq \sum_{i=0}^{\log n} n/2^i < 2n$. This gives an $O(n^2)$ total bound for all vertices.

Finally, we consider the total time required to maintain the incremental SCCs data structures. Such a structure for the subgraph induced by a component C with n_C vertices requires $O(n_C^2 \log n_C)$ total time. We distribute this cost to the n_C vertices of the component, so each vertex is charged a cost of $O(n_C \log n_C)$. Hence, for as long as a vertex v is a principal vertex in its component C, it is charged a cost of $O(n_C \log n_C)$, where n_C is the number of vertices in C just before it is merged as a secondary component. If this never happens, then n_C is the final number of vertices in C. It remains to bound the contribution of v as a secondary vertex, which we can do as above. Let $C_v^1, \ldots, C_v^\xi$ be the sequence of SCCs that contain v as a secondary vertex, where each C_v^i has n_v^i vertices just before it gets merged into a larger component. As before, $n_v^i \leq n_v^{i+1}/2$, for $1 \leq i < \xi$, and $n_v^\xi \leq n$. Then, the cost charged to v for the whole sequence of components $C_v^1, \ldots, C_v^\xi$ is bounded by $\sum_{i=1}^{\xi} n_v^i \log n_v^i \leq \log n \sum_{i=0}^{\log n} n/2^i < 2n \log n$. This gives an $O(n^2 \log n)$ total bound for all vertices. ◀

5 Decremental algorithm

In this section, we present our decremental algorithm for maintaining the 2-edge-connected components of a digraph G. We can assume that G is strongly connected, as otherwise we can process each SCC separately. Let s be an arbitrarily chosen start vertex of G. Let T be a fixed spanning tree of flow graph G_s, and let T_R be a fixed spanning tree of flow graph G_s^R. (Both T and T_R are rooted at s.) The algorithm operates on the assumption that the edges of T and T_R are never deleted throughout the deletion sequence.

We describe how to efficiently update the edges of the modified labeling graph $\widehat{LG}$ of Section 4.1, as we delete edges in G. To achieve this, we need to maintain some additional information about the tree edges that are covered by each vertex $v \in V(G)$.

For a tree edge $e \in T$, we define $cover_v(e)$ to be the number of edges $f \in \rho_{\mathcal{N}}(v)$ that cover e, i.e., such that $e \in T(f)$. Then $(v, e) \in E(\widehat{LG})$ if and only if $cover_v(e) > 0$. Our approach is to maintain the $cover_v(e)$ values using a dynamic tree data structure [32]. This way, we can update $\widehat{LG}$ efficiently, and maintain its SCCs decrementally using the algorithm of [39].

A *dynamic tree* $\mathcal{T}$ is a data structure that efficiently maintains a collection of rooted trees, whose edges have real-valued costs, under dynamic operations such as linking and cutting edges, while supporting cost updates and queries on paths and subtrees. For our purposes, we will assume that each such data structure maintains a single tree that corresponds to the spanning tree T of G_s with root s. Recall that for any vertex $v \neq s$, $t(v)$ denotes the parent of v in T. Here, we also let $T[v, w]$ denote the tree path between two vertices v and w, ignoring edge directions.

We will use the following dynamic tree operations, which are supported in $O(\log n)$ time.

- *cost*(v): If $v \neq s$, then return the cost of the edge $(t(v), v)$.
- *update*(v, w, x): Add x to the cost of all edges on the tree path $T[v, w]$.
- *mincost*(v, w): Return a vertex $u \in T[v, w]$ such that the edge $(t(u), u)$ has minimum cost among the edges on the tree path $T[v, w]$.

We note that the original description of Sleator and Tarjan [32] has the operation $update(v, x)$, which adds the value x to the cost of the edges on the path from v to the root of $\mathcal{T}$, and the operation $mincost(v)$, which returns an edge of minimum cost on the path from v to the root of $\mathcal{T}$. We can implement our versions of *update* and *mincost*, by using the operation $evert(v)$ of [32], which makes v the root of $\mathcal{T}$, as follows. To implement $update(v, w, x)$, we do $evert(w)$, $update(v, x)$, and $evert(s)$. Similarly, to implement $mincost(v, w)$ we do $evert(w)$, $mincost(v)$, and $evert(s)$.

To initialize $\widehat{LG}$, we compute a spanning tree T of G rooted at s, and insert the edges of $\widehat{LG}$ as in Section 4.2. We also compute the $pre(v)$ and $size(v)$ values for all $v \in T$, so that we can test the ancestor-descendant relation in T in $O(1)$ time. Then, for each vertex $v \in V(G)$, we maintain a dynamic tree data structure $\mathcal{T}_v$, which implements the operations *update* and *mincost* on T, where each tree edge $e \in T$ has cost $cover_v(e)$. Initially, all edge costs in $\mathcal{T}_v$ are zero. Then, we process each edge $(u, v) \in \rho_{\mathcal{N}}(v)$, and execute $update(v, u, +1)$.

During the execution of the deletion sequence, we do the following. Let $e = (u, v)$ be the next edge of G to be deleted. By our assumption, $e \in \rho_{\mathcal{N}}(v)$, and we need to find the tree edges $f \in T(e)$ that are covered only by e. For each such tree edge f, we delete the corresponding edge (v, f) from $\widehat{LG}$.

To find these edges $f \in T(e)$, first we execute $update(v, u, -1)$. Then, we recursively search a tree path $T[x, y]$ for uncovered edges (that is, tree edges $f \in T[x, y]$ with $cover_v(f) = 0$), where initially $x = u$ and $y = v$, as follows. We compute $w = mincost(x, y)$, and check if $cost(w) \neq 0$. If this is the case, then all the edges on $T[x, y]$ are still covered by v, and we are done. Otherwise, we delete the edge $(v, (t(w), w))$ from $\widehat{LG}$, and repeat recursively this step for the two paths of $T[x, y] \setminus (t(w), w)$.

In more detail, if w is an ancestor of x, then we recursively search for uncovered edges on $T[x, w]$ and on $T[y, t(w)]$. Otherwise, if w is an ancestor of y, then we recursively search for uncovered edges on $T[y, w]$ and on $T[x, t(w)]$. Hence, we obtain the following bound.

▶ **Lemma 14.** *We can maintain the modified label graph $\widehat{LG}$ under a sequence of edge deletions of G in $O(n^2 \log n)$ total time.*

Proof. Consider the deletion of an edge $e \in \rho_{\mathcal{N}}(v)$. The above procedure finds a tree edge $f \in T(e)$ that becomes uncovered by v using a constant number of dynamic tree operations. For each edge f with $cover_v(f) = 0$, we delete the corresponding edge (v, f) in $\widehat{LG}$. Since $\widehat{LG}$ has $O(n^2)$ edges, and each dynamic tree operation takes $O(\log n)$ time, the bound follows. ◀

From Lemma 14 and the fact that we use the decremental algorithm of [39] to maintain the SCCs of $\widehat{LG}$, we obtain Theorem 2.

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443, 2014. doi:10.1109/FOCS.2014.53.

2 Stephen Alstrup, Dov Harel, Peter W. Lauridsen, and Mikkel Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–32, 1999. doi:10.1137/S0097539797317263.

3 Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Robert E. Tarjan. A new approach to incremental cycle detection and related problems. *ACM Trans. Algorithms*, 12(2), December 2015. doi:10.1145/2756553.

4 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 365–376, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316335.

5 Adam L. Buchsbaum, Loukas Georgiadis, Haim Kaplan, Anne Rogers, Robert E. Tarjan, and Jeffery R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008. doi:10.1137/070693217.

6 Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, s-t shortest path, and minimum-cost flow. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, pages 1165–1173, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3618260.3649745.

7 Harold N. Gabow. Path-based depth-first search for strong and biconnected components. *Information Processing Letters*, 74:107–114, 2000. doi:10.1016/S0020-0190(00)00051-X.

8 Harold N. Gabow. The minset-poset approach to representations of graph connectivity. *ACM Transactions on Algorithms*, 12(2):24:1–24:73, February 2016. doi:10.1145/2764909.

9 Harold N. Gabow and Robert E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–21, 1985. doi:10.1016/0022-0000(85)90014-5.

10 Zvi Galil and Giuseppe F. Italiano. Reducing edge connectivity to vertex connectivity. *SIGACT News*, 22(1):57–61, March 1991. doi:10.1145/122413.122416.

11 Loukas Georgiadis, Thomas Dueholm Hansen, Giuseppe F. Italiano, Sebastian Krinninger, and Nikos Parotsidis. Decremental data structures for connectivity and dominators in directed graphs. In *ICALP*, pages 42:1–42:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.42.

12 Loukas Georgiadis, Giuseppe F. Italiano, and Evangelos Kosinas. Computing the 4-edge-connected components of a graph in linear time. In *Proc. 29th European Symposium on Algorithms*, pages 47:1–47:17, 2021. doi:10.4230/LIPIcs.ESA.2021.47.

13 Loukas Georgiadis, Giuseppe F. Italiano, and Evangelos Kosinas. Computing the 3-edge-connected components of directed graphs in linear time. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 62–85, 2024. doi:10.1109/FOCS61266.2024.00015.

14 Loukas Georgiadis, Giuseppe F. Italiano, Luigi Laura, and Nikos Parotsidis. 2-edge connectivity in directed graphs. *ACM Transactions on Algorithms*, 13(1):9:1–9:24, 2016. doi:10.1145/2968448.

15 Loukas Georgiadis, Giuseppe F. Italiano, and Nikos Parotsidis. Incremental 2-edge-connectivity in directed graphs. In *ICALP*, pages 49:1–49:15, 2016. doi:10.4230/LIPIcs.ICALP.2016.49.

16 Loukas Georgiadis, Giuseppe F. Italiano, and Nikos Parotsidis. Strong connectivity in directed graphs under failures, with applications. *SIAM J. Comput.*, 49(5):865–926, 2020. doi:10.1137/19M1258530.

17 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30, 2015. doi:10.1145/2746539.2746609.

18 Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. *SIAM Journal on Computing*, 49(1):1–36, 2020. doi:10.1137/18M1180335.

19 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, July 2001. doi:10.1145/502090.502095.

20 John E. Hopcroft and Robert E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973. doi:10.1137/0202012.

21 Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, and Seth Pettie. Fully dynamic connectivity in $O(\log n(\log\log n)^2)$ amortized expected time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 510–520, USA, 2017. Society for Industrial and Applied Mathematics. doi:10.5555/3039686.3039718.

22 Wenyu Jin and Xiaorui Sun. Fully dynamic s-t edge connectivity in subpolynomial time (extended abstract). In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 861–872, 2022. doi:10.1109/FOCS52979.2021.00088.

23 Ken-Ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *Journal of the ACM*, 66(1), December 2018. doi:10.1145/3274663.

24 Tuukka Korhonen. Linear-time algorithms for k-edge-connected components, k-lean tree decompositions, and more. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, STOC '25, pages 111–119, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3717823.3718123.

25 Evangelos Kosinas. Computing the 5-edge-connected components in linear time. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1887–2119, 2024. doi:10.1137/1.9781611977912.76.

26 Jakub Łącki. Improved deterministic algorithms for decremental reachability and strongly connected components. *ACM Transactions on Algorithms*, 9(3):27, 2013. doi:10.1145/2483699.2483707.

27 Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.

28 Wojciech Nadara, Mateusz Radecki, Marcin Smulewicz, and Marek Sokołowski. Determining 4-edge-connected components in linear time. In *Proc. 29th European Symposium on Algorithms*, pages 71:1–71:15, 2021. doi:10.4230/LIPIcs.ESA.2021.71.

29 Hiroshi Nagamochi and Toshihide Ibaraki. A linear time algorithm for computing 3-edge-connected components in a multigraph. *Japan J. Indust. Appl. Math*, 9(163), 1992. doi:10.1007/BF03167564.

30 Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, 2008. 1st edition. doi:10.1017/CBO9780511721649.

31 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time . In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 950–961, Los Alamitos, CA, USA, October 2017. IEEE Computer Society. doi:10.1109/FOCS.2017.92.

32 Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26:362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

33 Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

34 Robert E. Tarjan. Finding dominators in directed graphs. *SIAM Journal on Computing*, 3(1):62–89, 1974. doi:10.1137/0203006.

35 Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975. doi:10.1145/321879.321884.

36 Robert E. Tarjan and Jan van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM*, 31(2):245–81, 1984. doi:10.1145/62.2160.

37 Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, February 2007. doi:10.1007/s00493-007-0045-2.

38 Yung H. Tsin. Yet another optimal algorithm for 3-edge-connectivity. *Journal of Discrete Algorithms*, 7(1):130–146, 2009. Selected papers from the 1st International Workshop on Similarity Search and Applications (SISAP). doi:10.1016/j.jda.2008.04.003.

39 Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva. Almost-Linear Time Algorithms for Decremental Graphs: Min-Cost Flow and More via Duality . In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2010–2032, Los Alamitos, CA, USA, October 2024. IEEE Computer Society. doi:10.1109/FOCS61266.2024.00120.

Online Makespan Scheduling Under Scenarios

Ekin Ergen
Technical University of Berlin, Germany

Abstract

We consider a natural extension of online makespan scheduling on identical parallel machines by introducing scenarios. A scenario is a subset of jobs, and the task of our problem is to find a global assignment of the jobs to machines so that the maximum makespan under a scenario, i.e., the maximum makespan of any schedule restricted to a scenario, is minimized.

For varying values of the number of scenarios and machines, we explore the competitiveness of online algorithms. We prove tight and near-tight bounds, several of which are achieved through novel constructions. In particular, we leverage the interplay between the unit processing time case of our problem and the hypergraph coloring problem both ways: We use hypergraph coloring techniques to steer an adversarial family of instances proving lower bounds for our problem, which in turn leads to lower bounds for several variants of online hypergraph coloring.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online scheduling, scenario-based model, online algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.27

Related Version *Full Version*: `https://arxiv.org/abs/2507.04016`

Funding The author was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689).

Acknowledgements I would like to thank Martin Skutella for inspiring me to consider this problem, for his invaluable suggestions and for the insightful discussions. I am also grateful to the anonymous reviewers for their careful reading and valuable feedback.

1 Introduction

We study a natural extension of *online makespan scheduling*, also known as *online load balancing*, one of the most well-known problems in the scope of online optimization. In online makespan scheduling, we know a number m of identical parallel machines in advance, in addition to which jobs j are revealed one at a time along with their processing times $p_j \geq 0$. Our task is to assign the jobs to a machine as soon as they are revealed, solely under the knowledge of the preceding jobs, so as to minimize the *makespan*, i.e., the maximum sum of processing times any machine is assigned to. This problem was initially studied by Graham almost 60 years ago, who proved that assigning every job to a currently least loaded machine is a $(2-\frac{1}{m})$-competitive algorithm [13]. It was observed that no better deterministic algorithm can exist for $m \in \{2,3\}$ [13, 9], while for larger values of m, such a tight ratio is still not known.

We extend online makespan scheduling by a *scenario-based model*: We are given a number m of machines. There is a known number K of job sets (called *scenarios*) $S_k \subseteq \{1,\dots,n\}$, $k \in [K]$, over a common ground set $\{1,\dots,n\}$. Although the number of scenarios is known, neither the number of jobs nor their processing times p_j ($j \in [n]$) are known in advance. The ground set $[n]$ is revealed in increasing order of jobs; more precisely, we are incrementally informed of $p_1,\dots,p_j$ as well as $S_k \cap \{1,\dots,j\}$ for increasing j. The task is to find a global assignment $\tau\colon \{1,\dots,n\} \to \{1,\dots,m\}$ of the jobs to machines that delivers a reasonable solution restricted to every scenario. To be more precise, the objective is to minimize the


© Ekin Ergen;
licensed under Creative Commons License CC-BY 4.0

makespan of the *worst-case scenario*, i.e., the scenario that yields the highest makespan. As in the classical online scheduling problem, a job j must be assigned irrevocably to a machine $\tau(j)$ as soon as it is revealed. In essence, we are solving a finite number of instances of Online Makespan Scheduling over the same ground set of jobs simultaneously.

The simplicity of the scenario model offers a variety of interpretations and applications. For instance, we can view the scenarios as distinct services that are provided in m (number of machines) sites to n (number of jobs) customers, where the customers are revealed one by one with their desired services and need to be assigned to one of the sites immediately. Another interpretation of our problem may concern a fair balancing of goods according to multiple agents in the following setting: Each of the K agents are interested in partitioning a subset S_k, $k \in \{1, \dots, K\}$ of n goods with weights p_j among m buckets, where the partition should be as balanced for all agents as possible (i.e., we would like to minimize the maximum bucket load for any agent). Here, the goods are revealed one by one together with the agents that are interested in them.

Moreover, even special cases of online makespan scheduling under scenarios correspond to interesting questions in their own regard. The special case $p_j \equiv 1$ of unit processing times is related to some cases of discrepancy minimization as well as hypergraph coloring.

In the large array of previous research on combinatorial optimization under scenarios, it was shown for several offline problems that optimizing under a number of scenarios is computationally harder than their underlying single-scenario counterparts. Accordingly, our goal in this work is to gain a better understanding of the intersection of scenario-based models and competitiveness, in particular how competitiveness diminishes under a growing number of scenarios.

1.1 Our Contribution

Implementing a miscellany of ideas to analyze competitiveness bounds for the varying numbers m of machines and K of scenarios, we achieve several tight bounds and further nontrivial bounds. An overview of our results, depending on the numbers m and K, can be seen in Table 1. Several of the entries are obtained via a generalization of Graham's List Scheduling Algorithm which we extend using the pigeonhole principle.

By a simple example, one can show that for any number $m \geq 2$ of machines and (at least) 3 scenarios, no algorithm with a competitive ratio better than 2 can exist, even for jobs with unit processing times. To underline the competitiveness gap between two and three scenarios, we contrast the aforementioned result with an algorithm for two machines and two scenarios, which is one of our main results.

▶ **Theorem 1.** *There exists a* $5/3$*-competitive algorithm for* Online Makespan Scheduling under Scenarios *for* $m = K = 2$.

This algorithm as well as its analysis combine techniques well-known from algorithms for online makespan scheduling with novel ideas that concern the scenario-based properties of a schedule. We first alter the definition of competitive ratio by allowing only specific lower bounds for an offlie optimum, which allows us to make structural modifications and restrictions on *worst-case* instances. What sets our analysis apart from previous work is that we alternate between forward and reverse engineering: Assuming that our algorithm obeys some fixed rules, we explore what we may assume about a well-structured worst-case instance. In spite of seemingly strong restrictions that we impose, the ratio of our algorithm is not far from the best achievable bound by any deterministic algorithm, which we show is at least $\frac{9+\sqrt{17}}{8} \approx 1.640$. Altogether, we obtain a near-complete picture of competitiveness for $m = 2$.

The remainder of the paper demonstrates a disparity between increasing the number m of machines and increasing the number K of scenarios. For a fixed but arbitrary m, we show that there exists a sufficiently large number $K(m)$ of scenarios and a family of instances with $K(m)$ scenarios, on which no deterministic online algorithm can achieve a ratio better than m. Furthermore, we can even choose all instances to have unit processing times $p_j \equiv 1$. This result is particularly noteworthy, considering that any online algorithm on m machines is trivially m-competitive: Indeed, the online output is at most the total load and an offline optimum is at least the average load, and these are by at most a factor of m apart.

To find the suitable number $K(m)$, we interpret the instances as hypergraphs, passing to a problem we call Online Makespan Hypergraph Coloring(m) instead (cf. Section 1.2). Strikingly, such a family can be created using hypertrees on which no deterministic algorithm can achieve a coloring without a monochromatic edge simultaneously.

▶ **Theorem 2.** *Let $m \in \mathbb{N}$. There exists a number $K(m)$ such that for any $r < m$, no deterministic algorithm for Online Makespan Hypergraph Coloring(m) is r-competitive, even when restricted to hypertrees with $K(m)$ hyperedges.*

The statement of Theorem 2 is best possible in the sense that the numbers $K(m)$ cannot be replaced by a global K for which the incompetitiveness holds for all m (cf. Theorem 5).

The proof of Theorem 2 can be found in the full version of the paper. To showcase some of our techniques, we sketch a proof for a similar statement for $K = 3$ in Section 4. We utilize a subhypergraph on 7 nodes as a gadget and obtain a relatively compact construction, both in terms of the hypergraph and the case distinction that we apply to analyze it.

An advantage of our approach while proving Theorem 2 is that it only features hyperforests. More precisely, we implicitly show that for every $m \in \mathbb{N}$, there is a number $n \leq (2m+1) \uparrow\uparrow 5$ of nodes[1] such that no deterministic online algorithm on n nodes that uses m colors can avoid a monochromatic hyperedge. In other words, in order to color a hyperforest with an arbitrary number n of nodes online without monochromatic edges, we need $\Omega(\text{slog}_5 n)$ colors, even if the restrictions $e \cap \{1, \dots, j\}$ of hyperedges e to revealed nodes are known upon the revelation of nodes $\{1, \dots, j\}$. Here, the emphasis is on the additional knowledge of partial hyperedges: This makes an adversarial revelation of hyperedges more challenging and to the best of our knowledge, none of the previous lower bound constructions [15, 14] are applicable under such additional information. This result addresses a completely open problem from [15] and sets a non-constant lower bound for Online Hypergraph Coloring for hypertrees under the knowledge of partial hyperedges.

▶ **Corollary 3.** *Let $n \in \mathbb{N}$. There exists no $o(\text{slog}_5 n)$-competitive deterministic algorithm for Online Hypergraph Coloring with nodes $v_1, \dots, v_n$ (revealed in this order) even if we restrict to hypertrees and additionally, the subsets $e \cap \{v_1, \dots, v_j\}$ of hyperedges $e \in E$ are known upon the assignment of node v_j.*

For unit processing times, $K = 3$ and varying m, we further propose a 2-competitive algorithm, which is best possible due to the lower bounds mentioned earlier. Although the special case of unit processing times is admittedly a restriction, the result still illustrates a contrast with the tight lower bound of m for a varying number of scenarios, which also holds already for instances with unit processing times.

[1] $F(x) \coloneqq x \uparrow\uparrow 5 \coloneqq x^{x^{x^{x^x}}}$ and $\text{slog}_5(x) \coloneqq F^{-1}(x)$.

Table 1 Overview of the competitive ratios obtained by our results. Recall that on m machines, any algorithm is trivially m-competitive.

m ↓ K →	2	3	4+
2	> 1.640, $\leq 5/3$	$= 2$	$= 2$
3	≤ 2	≥ 2, $= 2$ for $p_j \equiv 1$	≥ 2 $= 3$ for $K \geq 233$ and $p_j \equiv 1$
4+	≥ 3	≥ 2, < 4 $= 2$ for $p_j \equiv 1$	≥ 2, $< K+1$ $= m$ for K sufficiently large and $p_j \equiv 1$

1.2 Preliminaries

Throughout this paper, for $n \in \mathbb{N}$, we denote by $[n] \coloneqq \{1, \ldots, n\}$ the set of numbers 1 through n. An instance $I = (n, (p_j)_{j\in[n]}, \{S_k\}_{k\in[K]})$ of Online Makespan Scheduling under Scenarios(m, K) (OMSS(m, K)) is parametrized by a number $m \in \mathbb{N}$ of machines and a number $K \in \mathbb{N}$ of scenarios, and defined as a tuple consisting of a number $n \in \mathbb{N}$ of jobs, a processing time $p_j \geq 0$ for each job j and job subsets $S_1, \ldots, S_K \subseteq [n]$, called *scenarios*. As we usually work with a single instance at a time, we often suppress this notation in the sequel as well as the parameters m and K, which are often specified per section.

For a set $S \subseteq [n]$, we denote $p(S) \coloneqq \sum_{j\in S} p_j$. Our objective is to find a partitioning $J_1 \dot\cup \ldots \dot\cup J_m$ of the set of jobs that minimizes $\max_{k\in[K]} \max_{i\in[m]} p(J_i \cap S_k)$. In other words, we consider schedules that arise from restricting a partition to each of the scenarios, and our goal is then to minimize the makespan of the worst-case scenario, which we also call the *makespan of the schedule*. Furthermore, our instance is *online* in the sense that parameters m (number of machines) and K (the number of scenarios) are known in advance, but jobs $j \in [n]$ as well as their processing times p_j and the indices $k \in [K]$ such that $j \in S_k$ are revealed one job j at a time. Once the information of a job j is revealed in addition to already known information about the jobs in $[j-1]$, the job j must be assigned to a machine $i \in [m]$, which is an irrevocable decision.

In our analyses and arguments, we use the notion of a *completion time* of a job j. In classical scheduling, this is defined as the sum of processing times preceding (including) j that are on the same machine as j. In our setting, the restricted schedules are evaluated without idle times, meaning that such completion times might be different for different scenarios. Accordingly, we define the *completion time of job $j \in J_i \cap S_k$ in scenario S_k* as $C_j^k \coloneqq \sum_{j'\in J_i\cap S_k\cap[j]} p_{j'}$ and the *completion time* as $C_j \coloneqq \max_{k\in[K], j\in S_k} C_j^k$.

Online algorithms are often analyzed by their performance compared to the best possible outcome under full information. In existing literature, this comparison is quantified by the *competitive ratio*. An algorithm is said to have competitive ratio ρ if for every instance, it is guaranteed to output a solution that has an objective value which deviates from the value of an optimal solution under full information by at most a factor of ρ.

We also consider the special case of unit processing times $p_j \equiv 1$. In this special case, we may view our jobs as nodes, our scenarios as hyperedges containing a subset of the nodes and the partition $J_1 \dot\cup \ldots \dot\cup J_m$ as a coloring $c\colon [n] \to [m]$ of the nodes. Then, our problem can be formulated as an online hypergraph coloring problem, in which we color a hypergraph $H = (V, E)$ with a fixed number of colors $1, \ldots, m$ and seek to minimize $\max_{i\in[m], e\in E} |\{c^{-1}(i)\} \cap e|$. The nodes $j \in [n]$ are revealed one at a time in increasing order along with partial hyperedges $e \cap [j]$ $(e \in E, j \in [n])$.[2] We shall name this problem Online Makespan Hypergraph Coloring(m) (OMHC(m)) and address it in Section 4.

[2] According to this definition, hyperedges can have size 1, although such hyperedges can and will be omitted as they pose no difference with respect to the objective function.

1.3 Related Work

The problem that we introduce in this paper lies in the intersection of several well-known problems, which we would like to mention briefly.

Optimization under Scenarios. In recent years, scenario-based models have received significant attention. An instance of a scenario model over a discrete optimization problem is often given as an instance of the underlying problem; and in addition, subsets of the underlying instance (called *scenarios*) are specified. The objective function involves the objective values that are attained restricted to each scenario; e.g., the maximum of these values over scenarios or their average.

Although a significant portion of the literature is rather recent, some well-known problems have been scenario-based problems in disguise. For instance, the famous exact matching problem ([16]), which asks whether a given graph whose edges are colored blue or red has a perfect matching with exactly k blue edges for a given k, is equivalent to the matching problem under two scenarios, whereas a stochastic scenario-based model for matchings has recently been studied in [4]. Other examples include bin packing ([7, 5]), metric spanning tree ([3]) and the traveling salesperson problem ([24]).

Among the scenario-based problems with the widest array of studies is machine scheduling with its many variations, see [21] for an extensive overview. The closest to our problem is [10], in which the worst-case makespan is being minimized for parallel machine scheduling on two machines and a variable number of scenarios. Although an instance under a variable number of scenarios is not approximable with a ratio better than the trivial bound of 2, a constant number of scenarios allows a PTAS.

In some variants of scenario scheduling, e.g., in [6], a complexity gap is observed between two and three scenarios. Interestingly, we obtain a distinction between two and three scenarios as well, at least when we restrict to the special case of two machines.

Online Scheduling. As it is impossible to capture the vast amount of literature on all online scheduling problems, let us restrict to online parallel machine scheduling with the objective of minimizing the makespan. This is precisely the special case $K = 1$ of our problem.

The problem was first addressed in Graham's seminal work [13] from which we also draw inspiration in multiple ways. The competitive ratio of $2 - 1/m$ for m machines was not beaten for three decades, until it was improved slightly in [12]. Graham's List Scheduling Algorithm is evidently best possible for $m = 2$, and it was observed to be best possible for $m = 3$ as well [9]. In the following years, a stream of research narrowed the gap between upper and lower bounds for competitive ratios depending on m ([2, 17, 1]), although for $m \geq 4$, still no tight bound is known. The state-of-the-art results are Fleischer and Wahl's 1.9201-competitive algorithm ([11]) and an asymptotic lower bound of 1.88 due to Rudin III ([18]). For $m = 4$, the best known lower bound is $\sqrt{3} \approx 1.732$, due to Rudin III and Chandrasekaran ([19]).

Discrepancy Minimization, Hypergraph Coloring. A problem similar to Online Makespan Hypergraph Coloring, often called Online Hypergraph Coloring, has been studied extensively over the years. In this problem, the objective is to use the smallest number of colors while avoiding a *monochromatic hyperedge*, i.e., a hyperedge whose nodes all attain the same color. For hypertrees with n nodes, an $O(\log(n))$-competitive algorithm is known to be best possible ([14]), while for general graphs with n nodes, no $o(n)$-competitive algorithm can exist ([15]). However, this standard model differs from ours, as a hyperedge is revealed only once all of its nodes have been revealed. This nuance deems it much easier

to construct families of instances to achieve lower bounds. Moreover, due to differences in their respective objective functions, this problem does not seem to have a direct relation to Online Makespan Hypergraph Coloring. However, our approach has implications for a variant of Online Hypergraph Coloring in which partial hyperedges are known (see Section 1.1).

The reader may also notice similarities to the well-known discrepancy minimization problem (see e.g. [23, 8]): Indeed, in the usual setting of discrepancy minimization, we have $m = 2$ and the objective is to minimize the difference $\max_{e \in E} ||\{c^{-1}(1)\} \cap e| - |\{c^{-1}(2)\} \cap e||$. If all hyperedges $e \in E$ have the same size, this objective function is even equivalent to ours. In this regard, the problem that we study generalizes the ℓ_∞ norm online discrepancy minimization [22] in uniform set systems as well.

2 Extending Graham's List Scheduling

Our first observation is a simple algorithm that generalizes Graham's List Scheduling Algorithm for OMSS(m, K) and performs well when there are too many machines to force the schedule to create significant discrepancy. Our algorithm beats the trivial bound of m if $m > K + 1$.

We first observe that if the number m of machines is large enough, we can apply the pigeonhole principle to find a machine that is not among the most loaded machines of any scenario. For $s \in [m-1]$ and $k \in [K]$, we call a machine i *s-favorable with respect to the k-th scenario* if there are at least s distinct machines $i' \neq i$ that each have at least the load of machine i in the k-th scenario, i.e., $p(J_{i'} \cap S_k) \geq p(J_i \cap S_k)$.

▶ **Observation 4.** *There exists a machine i that is $\left(\lceil \frac{m}{K} \rceil - 1\right)$-favorable with respect to every scenario $k \in [K]$.*

Proof. There are at most s machines per scenario that are not s-favorable. Let $s = \lceil \frac{m}{K} \rceil - 1$. Since $m > s \cdot K$, there must be a machine i which is not non-s-favorable with respect to any scenario. ◀

In light of this observation, we propose the following algorithm: Upon the assignment of each job j, we find a machine i that is $(\lceil m/K \rceil - 1)$-favorable with respect to all scenarios k with $j \in S_k$ and assign the job j to the machine i.

▶ **Theorem 5.** *The above algorithm is $\left(\frac{m-1}{\lceil \frac{m}{K} \rceil} + 1\right)$-competitive for $m > K$.*

Notice that the aforementioned competitive ratio is strictly smaller than $K + 1$.

This algorithm is also applicable in the special case where every job appears in at most k scenarios for a fixed $k \in \mathbb{N}$, yielding a ratio of $(m-1)/\lceil \frac{m}{k} \rceil + 1$. Furthermore, plugging in $K = 1$ yields the well-known competitive ratio of $2 - 1/m$ of Graham's List Scheduling Algorithm.

3 Near-Tight Bounds for $m = 2$ Machines

When the partitioning is among two machines, a clear line can be drawn: Any algorithm is trivially 2-competitive, and if we have at least 3 scenarios, this is best possible.

▶ **Observation 6.** *For $m = 2$ and $K \geq 3$, there exists no r-competitive algorithm for any $r < 2$, even for the special case of unit processing times.*

Proof. For all jobs j below, we assume $p_j = 1$. The first instance $\mathcal{I}_1$ is given as follows: $n = m + 1$, $S_1 = \{1, 3, \ldots, m + 1\}$, $S_2 = \{2, 3, \ldots, m + 1\}$.

We consider an arbitrary algorithm that places the first jobs 1 and 2 to two distinct machines, without loss of generality, to the first and second, respectively. In this case, the makespan at the end must be 2, as two jobs $(j, j') \neq (1, 2)$ must be assigned to the same machine. However, the following assignment, which places the first two jobs together would have yielded a completion time of 1: $J_1 = \{1, 2\}, J_i = \{i + 1\}$ for $2 \leq i \leq m$. Hence, no algorithm that places two unit weight jobs in $S_1 \setminus S_2$ resp. $S_2 \setminus S_1$ to two separate machines can attain a competitive ratio better than 2.

In light of this, we consider another instance $\mathcal{I}_2$, on which assigning the first two jobs to the same machine leads to a failure. It is given by $n = m + 2$, $S_1 = \{1, 3\}$, $S_2 = \{2, 4, 5, 6, \ldots, m + 2\}$, $S_3 = \{3, 4, 5, \ldots, m + 2\}$. The first two jobs are indeed identical to those of $\mathcal{I}_1$.

By our considerations about the instance $\mathcal{I}_1$, we may assume that the first two jobs are assigned to the same machine, the first one without loss of generality. However, any assignment that places the first two jobs to the same machine admits makespan 2: Otherwise, due to the third scenario, the jobs $\{3, \ldots, m + 2\}$ must occupy one machine each. The job j that is placed to the first machine has a common scenario with either job 1 (if $j = 3$), or with job 2 (otherwise). ◀

On the other hand, this statement cannot be extended to $K = 2$. To show this, we present a minimalistic yet insightful algorithm (Algorithm 1) for makespan scheduling under $K = 2$ scenarios and $m = 2$ machines. Our general idea is to fix a simple rule for assigning *double-scenario jobs*, i.e., jobs in $S_1 \cap S_2$, and partition the *single-scenario jobs* $j \in S_1 \triangle S_2$ so that the machines are reasonably balanced even after upcoming hypothetical double-scenario jobs.

▶ **Rule 7.** *If $j \in S_1 \cap S_2$, then assign the job j to a machine i such that the makespan of the schedule was previously attained by the other machine $3 - i$. In case of a tie, select the machine that received the job $j - 1$ ($j = 1$ is assigned to the first machine if applicable).*

Throughout our analysis, we only use two types of lower bounds for the offline optimum: Processing times p_j of selected jobs j whose completion times equal the makespan, and the average load of a selected scenario S_k. Hence, while searching for the worst-case sequence of the upcoming double-scenario jobs, we may replace the actual definition of competitiveness with that of a proxy one:

▶ **Definition 8.** *We define the* proxy competitive ratio *of a schedule as*

$$\rho = \frac{\max_{i,k\in\{1,2\}} p(S_k \cap J_i)}{\max\left\{\max_{k\in\{1,2\}}\left\{\frac{p(S_k)}{2}\right\}, \max_{j\in[n]\cap \mathrm{argmax}_{\mathrm{j}\in[\mathrm{n}]}\{\mathrm{C_j}\}}\{p_j\}\right\}},$$

where C_j denotes the completion time of the job j.

The proxy competitive ratio is an upper bound on the competitive ratio. The lower bounds that we use are already well-known; e.g., Graham's List Scheduling Algorithm is analyzed by bounding the proxy competitive ratio from above. We take this approach, however, one step further and restrict ourselves to analyzing only some instances that attain the largest possible proxy competitive ratio (see next Observation). This is significantly more difficult for the competitive ratio in the usual sense, as a witness to having good or bad offline optima highly depends on how the processing times of jobs fit together numerically.

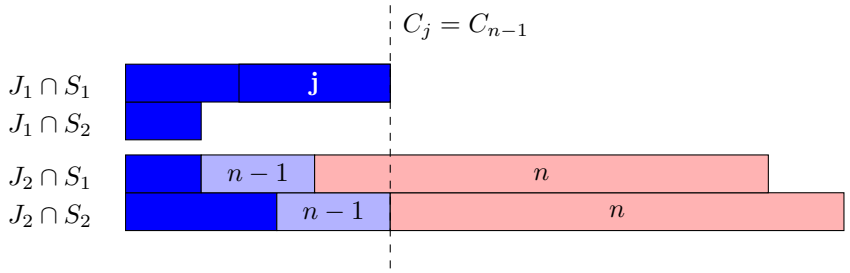


Figure 1 Illustrative description of Observation 9. Processing times p_j are given by the widths. Given the dark blue schedule, a worst outcome turns out to be the light blue job $n-1$ with a processing time p_{n-1} such that $C_{n-1} = C_j$, followed by the light red job n with $p_n = p(S_1 \cap [n-1])$.

▶ **Observation 9** (cf. Figure 1). *Let $\mathcal{A}$ be an online algorithm for OMSS$(2,2)$ obeying Rule 7. Assume that $\mathcal{A}$ has a proxy competitive ratio of at most $\frac{5}{3}$ on instances where*

(i) *$n \in S_1 \cap S_2$, and*

(ii) *$\mathcal{A}$ outputs a schedule with the following property: If $j \in [n]$ is the largest index with $j \in S_1 \triangle S_2$, then $C_j = C_{n-1}$.*

Then, $\mathcal{A}$ is $\frac{5}{3}$-competitive.

The above observation motivates the following parameter of significant value.

▶ **Definition 10.** *Consider a schedule where, without loss of generality, the makespan is attained at $S_1 \cap J_1$. The* anticipation α *of this schedule is given by the ratio*

$$\alpha := \begin{cases} \frac{p(J_1 \cap S_1)}{\max\{p(J_1 \cap S_2),\, p(J_2 \cap S_1) + p(J_1 \cap S_1) - p(J_2 \cap S_2)\}} & \text{if } p(J_2 \cap S_2) > p(J_2 \cap S_1), \\ 0 & \text{else.} \end{cases}$$

By convention, we assume $\frac{0}{0} = 1$.

In Figure 1, we have $p(J_2 \cap S_2 \cap [n-2]) > p(J_2 \cap S_1 \cap [n-2])$ and the second term of the denominator corresponds to the load $p(J_2 \cap S_1 \cap [n-1])$, i.e., the load of $J_2 \cap S_1$ after the $(n-1)$-st job is assigned. In general, anticipation measures the minimum discrepancy of a machine with respect to different scenarios after machine 2 is loaded with double-scenario jobs just enough to attain the makespan of the schedule together with machine 1. In our algorithm, one of the goals is to keep $\alpha \leq 2$ in order to brace ourselves for upcoming double-scenario jobs. The invariant which we maintain to this end is the following.

▶ **Invariant 11.** *The schedule satisfies the following assertions:*

(i) *Its proxy competitive ratio is bounded by $\rho \leq 5/3$.*

(ii) *The schedule has anticipation bounded by $\alpha \leq 2$.*

(iii) *Let k, i be given so that $J_i \cap S_k$ admits the makespan in the schedule. If the schedule is* dominated by the i-th machine, *i.e., $p(J_i \cap S_{k'}) > p(J_{3-i} \cap S_{k''})$ for all $k', k'' \in \{1,2\}$, then we have $p(J_i \cap S_k) \leq 2p(J_i \cap S_{3-k})$.*

The last property might appear somewhat unnatural at the first glance, though it is crucial for bounding the anticipation $\alpha \leq 2$ in the subsequent iterations.

Algorithm 1 shows how a single job j is assigned given a schedule of the first $j-1$ jobs where Invariant 11 is maintained. To analyze the algorithm, we prove that instances satisfying the properties in Observation 9 are always assigned so that the first $n-1$ jobs maintain Invariant 11 and the last job results in a proxy competitive ratio of at most $5/3$.

Algorithm 1 The assignment of a single job j given a $\frac{5}{3}$-competitive schedule that satisfies Invariant 11.

```
1: Rename machines and scenarios so that (J_1 ∩ S_1) is maximal
2: if j ∈ S_1 \ S_2 then
3:     return 2
4: end if
5: if j ∈ S_2 \ S_1 then
6:     if assigning j to the first machine satisfies Invariant 11 then
7:         return 1
8:     else
9:         return 2
10:    end if
11: end if
12: if j ∈ S_1 ∩ S_2 then
13:    return argmin_{i=1,2} max_{k=1,2} {p(J_i ∩ S_k)}
14: end if
```

A significant portion of the analysis is focused on proving that if line 9 is executed, Invariant 11 is indeed maintained. We refer the reader to the full version of the paper for further details.

In spite of the restrictions that arise from both the fixed double-scenario rule and how we evaluate lower bounds, we find out that our algorithm in the previous subsection is not far from being best possible.

▶ **Theorem 12.** *For $m = 2$ and $K = 2$, there exists no deterministic algorithm with competitive ratio strictly smaller than $\frac{9+\sqrt{17}}{8} \approx 1.640$.*

Proof. To ease the notation once again, let $X := 3 + \sqrt{17}$.

The first four jobs are given by $1, 2 \in S_1 \setminus S_2$, $3, 4 \in S_2 \setminus S_1$, $p_1 = p_2 = p_3 = p_4$. Any algorithm with competitive ratio better than 2 must assign these jobs so that the makespan equals 1 at the end; otherwise, we may stop revealing further jobs so that the offline optimum equals 1, while the makespan equals 2.

The fifth job fulfills $5 \in S_1 \setminus S_2$ and $p_5 = X$. Without loss of generality, it is assigned to the first machine.

The sixth job is given by $6 \in S_2 \setminus S_1$, $p_6 = \frac{X}{2}$. Now there are two options:

(i) If the sixth job is also assigned to the first machine, we reveal the seventh job with $7 \in S_2 \setminus S_1$ and $p_7 = X$. This job cannot be assigned to the first machine, as the makespan would then be $1 + \frac{3X}{2}$, while the offline optimum is X. Indeed, this would imply a ratio of at least

$$\frac{1 + \frac{9+3\sqrt{17}}{2}}{3 + \sqrt{17}} = \frac{9 + \sqrt{17}}{8}.$$

In particular, we obtain $p(S_1 \cap J_1 \cap [7]) = p(S_2 \cap J_2 \cap [7]) = 1 + X$.

(ii) If the sixth job is assigned to the second machine, we reveal the seventh job as $7 \in S_1 \cap S_2$ and $p_7 = \frac{X}{2}$. Due to similar reasoning to the other case, the seventh job must be assigned to the second machine and we again have $p(S_1 \cap J_1 \cap [7]) = p(S_2 \cap J_2 \cap [7]) = 1 + X$.

See Figure 2 for the first seven jobs in each of the cases.

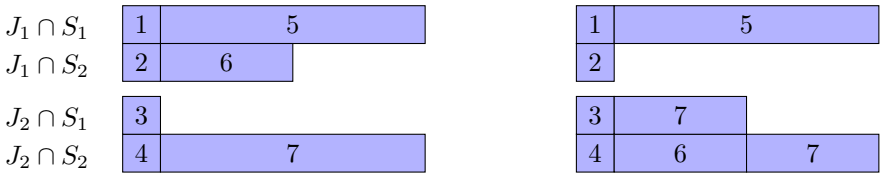


Figure 2 The first seven jobs in cases (i) and (ii) of the proof of Theorem 12, respectively. In both cases, an eighth job $8 \in S_1 \cap S_2$ forces the schedules to the desired lower bound of competitiveness.

In both of these cases, we reveal the eighth job with $8 \in S_1 \cap S_2$ and $p_8 = 2 + \frac{3X}{2}$. No matter where this job is assigned, the makespan is $3 + \frac{5X}{2}$, whereas the optimal solution $\{1, \dots, 7\} \dot{\cup} \{8\}$ achieves a makespan of $2 + \frac{3X}{2}$. Therefore, the competitive ratio is bound to be at least

$$\frac{3 + \frac{5X}{2}}{2 + \frac{3X}{2}} = \frac{6 + 5(3 + \sqrt{17})}{4 + 3(3 + \sqrt{17})} = \frac{9 + \sqrt{17}}{8},$$

as desired. ◀

Notice that the aforementioned family of instances fails to achieve small competitive ratios for reasons similar to those that require Property (iii) in Invariant 11.

It can also be shown through a simple array of instances that, no algorithm that satisfies Rule 7 can beat the bound of $5/3$, for which we refer the reader to the full version of the paper.

4 Tight Bounds for Sufficiently Many Scenarios

One of our main results (Theorem 2) states that for any number $m \in \mathbb{N}$ of machines, there exists a number K of scenarios such that no deterministic algorithm for OMSS(m, K) can have a competitive ratio less than the trivial upper bound m, even for unit processing times.

Due to space constraints, we merely sketch a proof for the special case of $m = 3$. While doing so, we showcase key ideas of the general argument, although we forfeit the restriction onto hypertrees for the sake of a more clearly structured description. In the context of scheduling, this is indeed not a significant change.

▶ **Theorem 13.** *For OMHC(3), there exists no r-competitive online algorithm for any $r < 3$, even when restricted to instances with at most 233 hyperedges of size at most 3 each.*

The problem OMHC(m), which we formulated in Section 1.2, is equivalent to the unit processing time case of OMSS(m, K) with K part of the input, so that the lower bounds are immediately implied for the latter as well.

To prove Theorem 13, we provide a family of instances on which any online algorithm is forced to output a monochromatic hyperedge of size 3. For each of the instances, we also maintain (offline) colorings $c\colon [j] \to \{\text{blue}, \text{red}, \text{yellow}\}$, with the property that if two nodes v, v' are contained in a common hyperedge, we have $c(v) \neq c(v')$. The latter coloring suggests an offline optimal value of 1 for each of the instances, implying the competitive ratio of 3 immediately. The main difference between the online and offline colorings is, as expected, that we may change the offline coloring completely at all times as long as its makespan equals 1. Among other modifications, we often employ a permutation $\pi\colon \{\text{red}, \text{blue}, \text{yellow}\} \to \{\text{red}, \text{blue}, \text{yellow}\}$ on connected components to realize this possibility. We must trace the offline coloring simultaneously because the hyperedges are often defined adversarially according to both colorings of the current hypergraph.

To differentiate between the online and offline colorings, we denote by $J_i \subseteq V$ the subset of nodes assigned to the online color i for $i \in \{1, 2, 3\}$. Right before the revelation of a node j, we implicitly mean $J_i \cap [j-1]$ whenever we mention J_i for some $i \in \{1, 2, 3\}$. Inspired by our original scheduling problem, we call the objective value $\max_{e \in E} \max_{i \in \{1,2,3\}} |e \cap J_i|$ of an instance its *makespan*. Without loss of generality, we truncate an instance as soon as the online solution is forced to obtain a makespan of 3.

Our gadget: Seven-node subhypergraph $\mathcal{S}$. Initially, we study a subhypergraph $\mathcal{S}$ that we use as a building block throughout our construction. The hypergraph $\mathcal{S}$ is given by the nodes $V(\mathcal{S}) = \{1, \dots, 7\}$ and hyperedges

$$E(\mathcal{S}) = \big\{\{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}, \{2,4\}, \{4,5\}, \{2,5\}, \{2,4,5\}, \\ \{4,6\}, \{6,7\}, \{4,7\}, \{4,6,7\}, \{3,6\}, \{3,7\}\big\}.$$

The hyperedges have size at most 3 and the set $E(\mathcal{S})$ is closed under taking subsets of size at least 2, i.e., if $e \in E(\mathcal{S})$, $|e| = 3$ and $e' \subseteq e$ with $|e'| = 2$, then $e' \in E(\mathcal{S})$. The hypergraph $\mathcal{S}$ together with the maximal hyperedges $e_1, \dots, e_5$ is given in Figure 3.

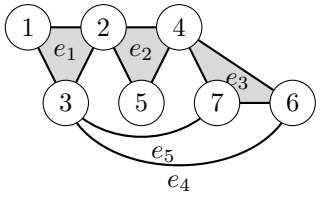


Figure 3 The hypergraph $\mathcal{S}$ with the maximal hyperedges $e_1, \dots, e_5$.

We first assume that the seven nodes in $V(\mathcal{S})$ are assigned to exactly two out of the three online colors. Since the online colors are initially symmetric, let us assume that we use the online colors 1 and 2. By a careful case distinction, we may observe:

▶ **Proposition 14.** *For any partitioning $V(\mathcal{S}) = J_1 \dot\cup J_2$ of the nodes that admits a makespan of at most 2, there exists a node coloring $c\colon V(\mathcal{S}) \to \{\text{blue}, \text{red}, \text{yellow}\}$ such that there are two hyperedges $e \in E(\mathcal{S}[J_1]), e' \in E(\mathcal{S}[J_2])$ with $c(v) \in \{\text{red}, \text{blue}\}$ (or another subset of $\{\text{blue}, \text{red}, \text{yellow}\}$ of size 2) for all $v \in e \cup e'$.*

In other words, we have induced monochromatic hyperedges of size 2 with two different colors, both with respect to the online coloring. Applying this repeatedly, we can further force the algorithm to a monochromatic hyperedge of size 3, which we show next.

Pigeonholing the two-color case. In our construction, we would like to create *copies* of the graph $\mathcal{S}$ successively, which we denote by $\mathcal{S}(t)$ for incrementing $t \in \mathbb{N}$.

A straightforward application of the pigeonhole principle yields that, if we have seven subhypergraphs $\mathcal{S}(1), \dots, \mathcal{S}(7)$ that are each partitioned onto two online colors, at least three of them (without loss of generality $\mathcal{S}(1)$, $\mathcal{S}(2)$, $\mathcal{S}(3)$) must be partitioned onto the same two online colors. Again without loss of generality, these two online colors are 1 and 2. Considering Proposition 14, we can permute the colorings of the three copies $\mathcal{S}(1)$, $\mathcal{S}(2)$ and $\mathcal{S}(3)$ each offline so that for any offline color $u \in \{\text{blue}, \text{red}, \text{yellow}\}$, there exists a copy $\mathcal{S}(t)$ with two (online) monochromatic hyperedges $e_i^t \in E(\mathcal{S}(t))$ $(i \in \{1, 2\})$ of size 2 avoiding the color u. The hypergraph in Figure 4 restricted to nodes of the form j_s^i illustrate these edges.

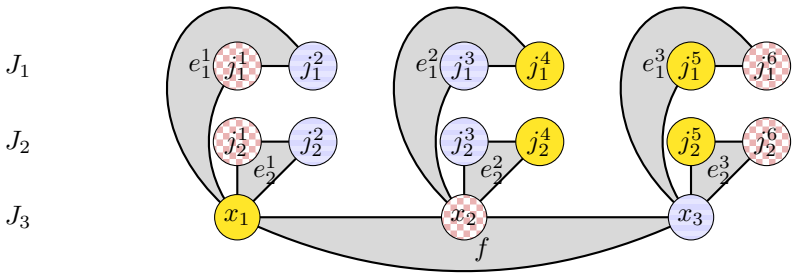


Figure 4 All fifteen relevant nodes together with the hyperedges within them. The heights of the vertices encode their online color (J_1, J_2, J_3), while their printed colors denote their offline color. Any assignment of the node x_3, given the other nodes and induced hyperedges, leads to a makespan of 3.

Once the six edges in question are found, one can insert nodes x_1, x_2, x_3 in this order, revealing hyperedges as depicted in Figure 4. To avoid a monochromatic edge, both x_1 and x_2 must be assigned the third online color. Keeping this in mind, it follows that x_3 cannot be assigned an online color without creating a monochromatic edge with respect to the online coloring. Together with the fact that the offline coloring indeed has makespan 1, we obtain:

▶ **Lemma 15.** *Any algorithm that assigns at least 7 copies of $\mathcal{S}$ to at most two online colors each is no better than 3-competitive.*

Subhypergraph $\mathcal{S}$ on Three Online Colors: The Palettes. It remains to consider the case where copies of the subhypergraph $\mathcal{S}$ are partitioned among the three colors, i.e., its nodes $\{1, \dots, 7\}$ are partitioned in nonempty sets $J_1 \dot\cup J_2 \dot\cup J_3$.

▶ **Lemma 16.** *For every online coloring $J_1 \dot\cup J_2 \dot\cup J_3$ of the nodes $V(\mathcal{S}) = \{v_1, \dots, v_7\}$ for nonempty sets $J_1 \dot\cup J_2 \dot\cup J_3$, there is an offline coloring $c\colon V(\mathcal{S}) \to \{\text{red}, \text{blue}, \text{yellow}\}$ such that two vertices exist with different online colors but the same offline color.*

In our construction, we exploit this property as follows: Let us consider a copy $\mathcal{S}(t)$ and fix two of its nodes $v_i(t) \in J_i$, $v_{i'}(t) \in J_{i'}$ provided by Lemma 16 as well as a third node $v_{i''}(t) \in J_{i''}$, where $i'' \notin \{i, i'\}$. By their choice, the nodes $v_i(t)$ ($i \in \{1, 2, 3\}$) admit at most two offline colors and avoid a third offline color $C \in \{\text{blue}, \text{red}, \text{yellow}\}$.

In this case, the subhypergraph $\mathcal{S}(t)$ is called a C *palette* (i.e., *yellow palette*, *blue palette* or *red palette*) and the corresponding nodes $v_i(t)$ are called *palette nodes*. In particular, $c(v_i(t)) \neq C$ holds for the palette nodes of a C palette. The choice of the name is motivated by the fact that, if we connect a new node w to e.g. yellow palette nodes, we are still allowed to extend the offline coloring by $c(w) = \text{yellow}$, while maintaining that the makespan of the offline coloring is 1.

The Construction. Below is a summary of our construction for the case where Lemma 15 is not applicable. Essentially, we create copies of the subhypergraph $\mathcal{S}$ which we connect with the existing hypergraph. The online color of the node v is denoted by $\varphi(v)$ and its (current) offline color by $c(v)$.

(i) We create up to 13 copies of the subhypergraph $\mathcal{S}$. By Lemma 15, we may assume that at least 7 of the copies are assigned to all three online colors (we truncate this step as soon as 7 such copies can be found). We handpick 7 such copies of $\mathcal{S}$, calling them $\mathcal{S}(1), \dots, \mathcal{S}(7)$.

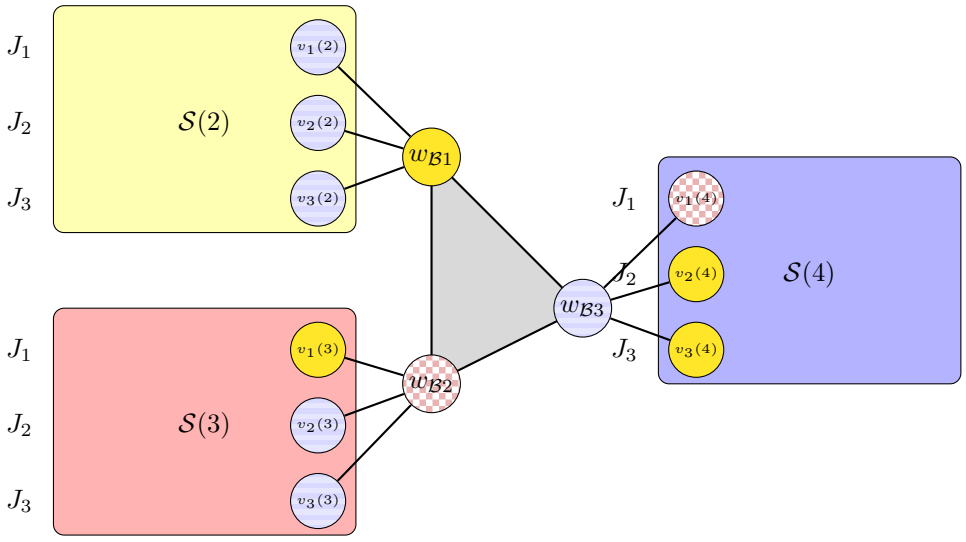


Figure 5 An example constellation of the three palettes together with the nodes $w_{\mathcal{B}t}$. The y-axis in the palettes denote the online colorings of the palette nodes respectively. The offline colors of the palette nodes might look differently, but they are never the same color as the palette, which is depicted as the color of the rectangles. The online coloring of the nodes $w_{\mathcal{B}i}$ are not yet known.

(ii) We permute the offline coloring on each copy of $\mathcal{S}$ so that $\mathcal{S}(1)$, $\mathcal{S}(4)$ and $\mathcal{S}(7)$ are yellow palettes, $\mathcal{S}(2)$ and $\mathcal{S}(5)$ are blue palettes, and $\mathcal{S}(3)$ and $\mathcal{S}(6)$ are red palettes.

(iii) We reveal a node $w_{\mathcal{A}}$ which is connected to palette nodes $v_i(1) \in V(\mathcal{S}(1)) \cap J_i$ for $i \in \{1,2,3\}$. The assignment to the online color $\varphi(w_{\mathcal{A}})$ leads to a monochromatic hyperedge $\{v_{\varphi(w_{\mathcal{A}})}(1), w_{\mathcal{A}}\}$ of online color $\varphi(w_{\mathcal{A}})$. We define $v_{\mathcal{A}} \coloneqq v_{\varphi(w_{\mathcal{A}})}(1)$.

(iv) We reveal three nodes $w_{\mathcal{B}1}$, $w_{\mathcal{B}2}$ and $w_{\mathcal{B}3}$ that are connected by the hyperedge $\{w_{\mathcal{B}1}, w_{\mathcal{B}2}, w_{\mathcal{B}3}\}$ to each other. Moreover, for each $s \in \{1,2,3\}$, the node $w_{\mathcal{B}s}$ is connected to the palette nodes $v_i(s+1)$ for $i \in \{1,2,3\}$. There is an $i \neq \varphi(w_{\mathcal{A}})$ such that $\{v_i(s+1), w_{\mathcal{B}s}\}$ is a monochromatic edge of online color i for a suitable copy $\mathcal{S}(s)$, $s \in \{1,2,3\}$. For this s, we define $v_{\mathcal{B}} \coloneqq v_i(s+1)$ and $w_{\mathcal{B}} \coloneqq w_{\mathcal{B}s}$.

(v) We reveal further copies of $\mathcal{S}$ together with edges described below, until a copy $\mathcal{C}$ is assigned to all three online colors. In the copy $\mathcal{C}$, we denote the nodes $v_i(t)$ by $w_{\mathcal{C}t}$ for clarity. The nodes $w_{\mathcal{C}t}$ of the copy $\mathcal{C}$ are connected to the palettes $\mathcal{S}(5)$, $\mathcal{S}(6)$, $\mathcal{S}(7)$ by edges depending on their offline colors, in that nodes $w_{\mathcal{C}t}$ with $c(w_{\mathcal{C}t}) = C$ are connected to the palette nodes $v_i(s)$ for all $i \in \{1,2,3\}$ and $s \in \{5,6,7\}$ such that $\mathcal{S}(s)$ is a C palette, see Figure 6. As the nodes $w_{\mathcal{C}t}$ are assigned to all three online colors, a makespan of 2 is also achieved in the online color $i \notin \{\varphi(w_{\mathcal{A}}), \varphi(w_{\mathcal{B}})\}$. More precisely, there exists a monochromatic edge $\{v_i(s), w_{\mathcal{C}t}\}$ of the online color i. We rename $v_{\mathcal{C}} \coloneqq v_i(s)$ and $w_{\mathcal{C}} \coloneqq w_{\mathcal{C}t}$.

(vi) We permute the offline coloring in each of the connected components created in (iii), (iv) and (v) again, so that $c(v_j) =$ blue and $c(w_j) =$ yellow for $j \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$.

(vii) We add a final node v_n together with hyperedges $\{v_n, v_j, w_j\}$ for each $j \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$. Now, the online color $\varphi(v_n)$ that this node is assigned to admits a makespan of 3, while extending the offline coloring by $c(v_n) =$ red maintains an offline coloring with makespan 1.

Notice that we have maintained an offline coloring of the nodes with makespan 1 so far up to the very last node, and the last node n neighbors only blue and yellow nodes by construction and therefore the offline assignment $c(n) =$ red does not increase the makespan.

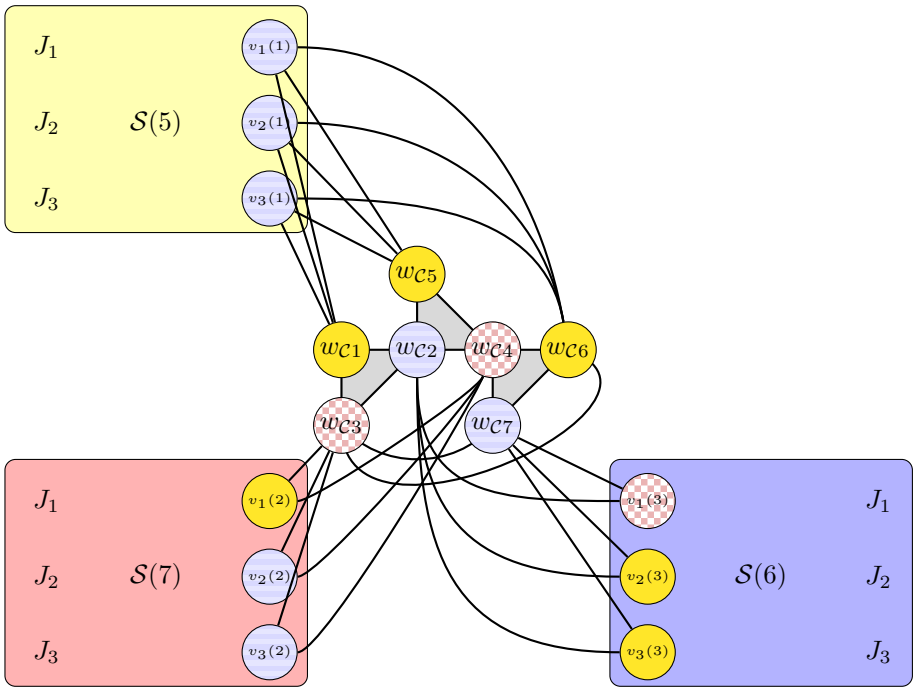


Figure 6 Partial sketch of subinstance $\mathcal{C}$.

In total, this yields that the offline optimal value is 1. A careful enumeration of introduced hyperedges reveals that 233 hyperedges suffice to create a monochromatic hyperedge for any of the cases demonstrated, which proves Theorem 13.

A Note on Theorem 2. Here, we abstain from presenting a proof of Theorem 2. Our construction in this general case is also based on incrementally creating monochromatic edges of each online color. However, we are faced with two fundamental challenges:

- **(i)** The monochromatic hyperedges increase in size rather slowly, yet unless they have size of $m-1$, they are not strong enough to act as a palette for other hyperedges in the same sense as here.
- **(ii)** To use our construction for Corollary 3, we restrict to hypertrees. This is a significant constraint, as it roughly implies that every palette may be used at most once.

Therefore, we include a more intricate recursive construction with much larger hypergraphs in the full version of the paper.

5 Tight Bounds For $K = 3$ and Unit Processing Times

In the previous section, we have established that for large enough K, there is no better-than-3-competitive algorithm for OMSS$(3, K)$, even for unit processing times. Let us reverse the parameters and ask: How competitively can we solve OMSS$(m, 3)$? This is not an easy question, indeed, even for OMSS$(m, 1)$, $m \geq 4$, a tight bound is not known. Restricting to the unit processing time case ($p_j \equiv 1$), we again obtain a tight bound, which is the smallest possible ratio (due to Theorem 5):

▶ **Theorem 17.** *For all $m \in \mathbb{N}$, there is a 2-competitive algorithm for the special case of OMSS$(m, 3)$ where the jobs have unit processing times.*

The algorithm in question employs an adapted round-robin method which assigns jobs to machines according to the scenarios they appear in. Essentially, we try to assign jobs $j \in S_1 \setminus (S_2 \cup S_3)$ and $j' \in (S_2 \cup S_3) \setminus S_1$ in a similar manner, and analogously for cyclic permutations of the indices $i \in \{1, 2, 3\}$. This gives a partition of jobs $j \in [n] \setminus (S_1 \cap S_2 \cap S_3)$ in three categories. The round robin assignments for each category are translated by roughly $m/3$, and jobs $j \in S_1 \cap S_2 \cap S_3$ are handled separately.

The algorithm is best visualized by drawing an analogue to a bingo card, for which we refer the reader to the full version.

6 Conclusion and Outlook

Throughout, we have established several competitiveness results for Online Makespan Scheduling under Scenarios. The first main takeaway from our work is a competitiveness gap between two and at least three scenarios when we consider $m = 2$ machines. This result draws a parallel to the known tractability results of several other problems, in which the line between easy and hard was drawn between two and three scenarios. Surely, the fact that the subsets of $\{1, 2, 3\}$ are not laminar, has been a deciding factor in our simple non-competitiveness result, as has been in several of the well-known results. Interestingly, restricting to the proxy competitive ratio as well as fixing the method of assigning double-scenario jobs essentially reduced the existence problem to the description of a polyhedron. We believe that similar techniques might prove useful for other cases and related problems as well.

We have further compared the behavior of competitiveness for increasing number K of scenarios and increasing number m of machines. In the special setting of unit processing times and $m = 3$ machines, the contrast is evident: On one hand, there is a 2-competitive algorithm for three scenarios and m machines for all $m \in \mathbb{N}$, which matches the lower bound obtained readily for $m = 2$. On the other hand, for K sufficiently large, the trivial upper bound of 3 cannot be beaten. That being said, $K + 1$ is a strict upper bound as well for a problem on K scenarios. To summarize, for large values of m and small values of K, we would expect the possibilities for competitive ratios to be dominated by K.

In light of our results, we may raise several open questions: In addition to the obvious task of finding tight bounds for all possibilities (m, K) of the numbers of machines and scenarios, exploring the competitiveness as m approaches to infinity in the weighted processing time case remains interesting. However, these problems have not been resolved even for $K = 1$, whence it is fair to assume challenges in further progress. Another interesting research direction is to introduce *migration*, as seen in e.g. [20]. In this model, we are allowed to revoke a bounded size of decisions in hindsight. Similarly, instead of minimizing the worst-case scenario, one could look into minimizing the average scenario or maximum regret.

References

1 Susanne Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999. doi:10.1137/S0097539797324874.

2 Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51(3):359–366, 1995. doi:10.1006/jcss.1995.1074.

3 Dimitris Bertsimas, Patrick Jaillet, and Amedeo R. Odoni. A priori optimization. *Operations Research*, 38(6):1019–1033, 1990. doi:10.1287/OPRE.38.6.1019.

4 Danny Blom, Dylan Hyatt-Denesik, Afrouz Jabal Amelia, and Bart Smeulders. Approximation algorithms for k-scenario matching. In Marcin Bieńkowski and Matthias Englert, editors, *Approximation and Online Algorithms*, pages 89–103, Cham, 2025. Springer Nature Switzerland.

5 Yulle Borges, Vinicius Loti de Lima, Flávio Miyazawa, Lehilton Pedrosa, Thiago Queiroz, and Rafael Schouery. Algorithms for the bin packing problem with scenarios. *CoRR*, May 2023. doi:10.48550/arXiv.2305.15351.

6 Thomas Bosman, Martijn van Ee, Ekin Ergen, Csanád Imreh, Alberto Marchetti-Spaccamela, Martin Skutella, and Leen Stougie. Total completion time scheduling under scenarios. In *Proceedings of the 21st International Workshop on Approximation and Online Algorithms*, pages 104–118. Springer, 2023. doi:10.1007/978-3-031-49815-2_8.

7 Attila Bódis and János Balogh. Bin packing problem with scenarios. *Central European Journal of Operations Research*, 27(2):377–395, June 2019. doi:10.1007/s10100-018-0574-3.

8 Moses Charikar, Alantha Newman, and Aleksandar Nikolov. Tight hardness results for minimizing discrepancy. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1607–1614, January 2011. doi:10.1137/1.9781611973082.124.

9 Ulrich Faigle, Walter Kern, and Gyorgy Turan. On the performance of on-line algorithms for partition problems. *Acta Cybern.*, 9:107–119, January 1989. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3359.

10 Esteban Feuerstein, Alberto Marchetti-Spaccamela, Frans Schalekamp, René Sitters, Suzanne van der Ster, Leen Stougie, and Anke van Zuylen. Minimizing worst-case and average-case makespan over scenarios. *Journal of Scheduling*, pages 1–11, 2016.

11 Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In Mike S. Paterson, editor, *Algorithms - ESA 2000*, pages 202–210, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:10.1007/3-540-45253-2_19.

12 Gábor Galambos and Gerhard J. Woeginger. An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22(2):349–355, 1993. doi:10.1137/0222026.

13 R. L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 45(9):1563–1581, 1966. doi:10.1002/j.1538-7305.1966.tb01709.x.

14 Yaqiao Li and Denis Pankratov. Online vector bin packing and hypergraph coloring illuminated: Simpler proofs and new connections. *CoRR*, June 2023. doi:10.48550/arXiv.2306.11241.

15 J. Nagy-György and Cs. Imreh. Online hypergraph coloring. *Information Processing Letters*, 109(1):23–26, 2008. doi:10.1016/j.ipl.2008.08.009.

16 Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *J. ACM*, 29(2):285–309, April 1982. doi:10.1145/322307.322309.

17 Karger D. R., Phillips S. J., and Torng E. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 1996.

18 John F. Rudin. *Improved bounds for the on-line scheduling problem.* PhD thesis, The University of Texas at Dallas, 2001.

19 John F. Rudin and R. Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM Journal on Computing*, 32(3):717–735, 2003. doi:10.1137/S0097539702403438.

20 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming*, pages 1111–1122, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-27836-8_92.

21 Dvir Shabtay and Miri Gilenson. A state-of-the-art survey on multi-scenario scheduling. *European Journal of Operational Research*, 2022.

22 Joel Spencer. Balancing games. *Journal of Combinatorial Theory, Series B*, 23(1):68–74, August 1977. doi:10.1016/0095-8956(77)90057-0.

23 Joel Spencer. Six standard deviations suffice. *Transactions of the American Mathematical Society*, 289(2):679–706, June 1985. Copyright: Copyright 2016 Elsevier B.V., All rights reserved. doi:10.1090/S0002-9947-1985-0784009-0.

24 Martijn van Ee, Leo van Iersel, Teun Janssen, and René Sitters. A priori TSP in the scenario model. *Discrete Applied Mathematics*, 250:331–341, 2018.

Sliding Squares in Parallel

Hugo A. Akitaya
Miner School of Computer and Information Sciences, University of Massachusetts Lowell, MA, USA

Sándor P. Fekete
Department of Computer Science, TU Braunschweig, Germany

Peter Kramer
Department of Computer Science, TU Braunschweig, Germany

Saba Molaei
Sharif University of Technology, Teheran, Iran

Christian Rieck
Department of Discrete Mathematics, University of Kassel, Germany

Frederick Stock
Miner School of Computer and Information Sciences, University of Massachusetts Lowell, MA, USA

Tobias Wallner
Department of Computer Science, TU Braunschweig, Germany

Abstract

We consider algorithmic problems motivated by modular robotic reconfiguration in the *sliding square model*, in which we are given n square-shaped modules in a (labeled or unlabeled) start configuration and need to find a schedule of sliding moves to transform it into a desired goal configuration, maintaining connectivity of the configuration at all times. Recent work has aimed at minimizing the total number of moves, resulting in fully sequential schedules that can perform reconfiguration in $\mathcal{O}(n^2)$ moves, or $\mathcal{O}(nP)$ for arrangements of bounding box perimeter size P.

We provide first results in the sliding square model that exploit parallel motion, performing reconfiguration in worst-case optimal makespan of $\mathcal{O}(P)$. We also provide tight bounds on the complexity of the problem by showing that even deciding the possibility of reconfiguration within makespan 1 is NP-complete in the unlabeled case. In the labeled variant, we note that deciding the same for makespan 2 is NP-complete, while makespan 1 is straightforward.

2012 ACM Subject Classification Theory of computation → Computational geometry; Computing methodologies → Motion path planning

Keywords and phrases Sliding squares, parallel motion, reconfigurability, motion planning, multi-agent path finding, makespan, swarm robotics, computational geometry

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.28

Related Version *Full Version*: https://arxiv.org/abs/2412.05523 [5]

Funding This work was partially supported by the German Research Foundation (DFG), project "Space Ants" (FE 407/22-1) and grant 522790373, by the National Science Foundation (NSF), grant CCF-2348067, and by a fellowship of the German Academic Exchange Service (DAAD).

1 Introduction

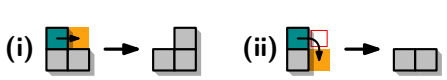
Reconfiguring an arrangement of geometric objects is a fundamental task in a wide range of areas, both in theory and practice. A typical task arises from relocating a (potentially large) collection of agents from a given start into a desired goal configuration in an efficient manner, while avoiding collisions between objects or violating other constraints, such as maintaining connectivity of the overall arrangement. In recent years, the problem of modular robot reconfiguration [9, 28, 30] has enjoyed particular attention [1, 2, 3, 4, 6] in the context



© Hugo A. Akitaya, Sándor P. Fekete, Peter Kramer, Saba Molaei, Christian Rieck, Frederick Stock, and Tobias Wallner;
licensed under Creative Commons License CC-BY 4.0



of Computational Geometry: In the *sliding square model* introduced by Fitch, Butler, and Rus [19], a given start configuration of n identical modules, each occupying a square grid cell, must be transformed by a sequence of atomic, sequential moves (shown in Figure 1(a)) into a target arrangement, without losing connectivity of the underlying grid graph.



(a) Two types of move can be performed by individual modules: **(i)** Slides and **(ii)** convex transitions.



(b) Some moves can be performed in parallel transformations, as shown here.

Figure 1 Our model allows for two types of moves to occur in parallel, collision-free transformations. In this paper, we show the symmetric difference of a transformation using turquoise and yellow.

Aiming at minimizing the total number of moves, previous research has resulted in considerable progress, recently establishing [2] universal configuration in $\mathcal{O}(nP)$ moves for a 2-dimensional arrangement of n modules with bounding box perimeter size P, and $\mathcal{O}(n^2)$ in three dimensions [1, 22]. However, the resulting schedules are purely sequential, not optimally minimizing the overall time until completion, called *makespan*. With parallel motion, much lower makespan can be achieved – which is also a more challenging objective, as it requires coordinating the overall motion plan, not just at the atomic level (see Figure 1(b)), but also at the global level to maintain connectivity and avoid collisions.

1.1 Our contributions

We provide first results for *parallel* reconfiguration in the sliding squares model with no restriction on the input. We achieve tight outcomes, both on the negative and the positive side.

1. We prove that even deciding the existence of a schedule with the smallest possible makespan of 1 is NP-complete in the parallel reconfiguration model, as well as APX-hard.
2. We give an in-place algorithm that achieves makespan $\mathcal{O}(P)$, where P is the perimeter of the union of the bounding boxes of start and target configurations. We show that this is asymptotically worst-case optimal.

Due to limited space, we only provide high-level descriptions here and refer to the full version [5] for the majority of our proofs and full technical details. Additionally, we extend our results to the *labeled* variant: When individual robots are distinguishable, deciding makespan 2 is NP-complete, while makespan 1 is easy. Moreover, asymptotically worst-case optimal schedules can still be computed in polynomial time with a relaxation of the in-place requirement. For precise definitions and the problem statement, see Section 2.

1.2 Related work

On the theoretical side, algorithmic methods for coordination the motion of many robots can be traced back to the classical work of Schwartz and Sharir [29] from the 1980s. On the practical side, [20] presented an architecture for modular robots, followed by a wide spectrum of work that often used cuboids as elementary building blocks; see [32] for a survey.

Of particular interest for the algorithmic side is the *sliding cube model* (or *square*) by Fitch, Butler, and Rus [19], introduced in the context of a modular robotic hardware [9, 28, 30]. Recent work [1, 4, 13, 22] has studied algorithmic methods for *sequentially* sliding squares and cubes, with typical schedules requiring a quadratic number of moves. Also related are models with slightly different types of moves, such as “pivoting”, see [2, 3, 10, 24].

In [24] the authors claim an algorithm for connectivity-preserving reconfiguration of sliding squares in $\mathcal{O}(n)$ parallel steps. Unfortunately, this is incorrect for general input. Nevertheless, if the input configurations are guaranteed to not contain a specific "forbidden" local pattern (a 2×2 arrangement with only two diagonally adjacent squares), then their algorithm is correct. Similar models of parallel sliding squares have been investigated experimentally [31]. However, this still leaves the existence of linear-time parallel protocols unresolved.

While the previous work on sliding squares focuses on minimizing the number of moves, a different line of research aims at minimizing the total time until completion. In [21], the authors study parallel distributed algorithms for a less restrictive model of sliding squares, obtaining $\mathcal{O}(n)$ makespan. They show how to obtain the same result in the standard sliding square model using *meta-modules*, small groups of cooperating modules that behave like a single entity. This, however, imposes extra constraints in the input. Algorithmic research on meta-modules was considered in [6, 27]. In [7, 12], the authors considered coordinated motion planning in a less constrained model (see discussion in Section 2) for labeled robots to minimize the makespan, aiming for *constant stretch*, i.e., a makespan within a constant factor of the maximum distance for a single robot. This was extended to preserve connectivity for unlabeled [8, 16] and labeled robots [18], and between obstacles [17].

Scaffolding and Meta-modules. Computing reconfiguration paths is not a simple task and, for many models, it is computationally intractable (even PSPACE-complete [3]). *Scaffolding* (proposed independently by [23] and [26]) aims at making reconfiguration simpler by restricting the scope to configurations containing a regular substructure (the *scaffold*) that remains connected between moves (thus removing complexity from the connectivity constraint) while having enough empty positions to allow modules to move through the scaffold (thus removing complexity from the free space constraints). They obtain a scaffold by using *meta-modules*, groups of modules that cooperate to move as a higher-level functional unit. Previous work has used these to translate algorithms for a particular model to another (see, e.g., [21, 27, 28]).

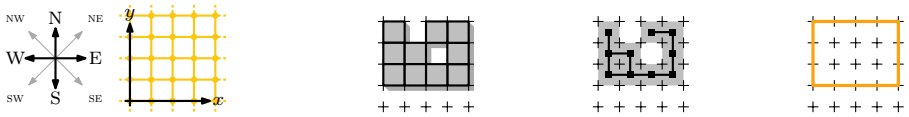
The drawback of these techniques is that their input must belong to the subset of configurations where modules are already organized into meta-modules (the scaffold exists in both the initial and target configuration). The authors of [8, 16] proposed an algorithm that first builds the scaffold from an arbitrary scaled configuration. They achieve *constant stretch*, i.e., a makespan within a constant factor of the maximum minimum distance for a single robot (based on previous work [7, 12]). This was extended for labeled modules [18], and between obstacles [17]. Although their techniques apply to a wider range of configurations, there are still restrictions on the input. The algorithm presented in this paper also makes use of scaffolding and meta-module techniques. To the best of our knowledge, our algorithm is the first universal one to build the scaffold and meta-modules (i.e., from arbitrary input).

Model comparison. Our model is more constrained than existing models for parallel reconfiguration under connectivity constraints [15, 16, 18, 21, 24]. Michail, Skretas, and Spirakis [24] do not explicitly require the connected backbone constraint, and do not explicitly define the collision model. The models in [16, 18] do not require the connected backbone constraint, and allow modules to enter and leave cells simultaneously, even in orthogonal directions. Similarly, [21] relaxes the free-space constraint of the convex transition move, allowing a module to move through two diagonally adjacent static modules. However, the authors disallow chain moves as shown in Figure 1(b), so that the cells involved in individual slide moves in the same transformation are disjoint. The same is true for [14, 15] where the paths of transformations at any given time stamp must be disjoint.

2 Preliminaries

We study the connected reconfiguration of squares in the infinite, two-dimensional integer grid by sliding moves in a parallel variant of the sequential *sliding squares* model, as follows.

Configurations. Each simple 4-cycle of edges in the infinite integer grid $\mathbb{Z}^2$ bounds a unit *cell* c, which is uniquely identified by its minimal integer coordinates $x(c)$ and $y(c)$. Two such cells are *edge-adjacent* (resp., *vertex-adjacent*) if their boundary cycles share an edge (resp., vertex). A *configuration* C of n square *modules* is defined by the set of occupied cells, each containing a single module. We say that C is valid if and only if its dual graph, defined by edge-adjacency, is connected, and denote by B the unique axis-aligned bounding box of minimal perimeter P that contains C; see Figure 2 for illustrations.



(a) The integer grid $\mathbb{Z}^2$. **(b)** A configuration C of 10 modules, its dual graph, and bounding box B.

Figure 2 We illustrate the relation of the integer grid and configurations.

Moves. Individual modules can perform two types of *move* into an unoccupied edge-adjacent or vertex-adjacent cell, *slides* and *convex transitions*, as illustrated in Figure 1(a). We denote a move by its start and target cells, e.g., $u \rightsquigarrow v$. In our model, both moves take an identical (unit) duration to complete, which allows us to easily extend the existing notion of move counting in sequential models [4, 13, 15, 25] to the parallel setting. Note that in our figures, the path denoting a convex transition is shown containing a circular arc for clarity. Such a path would be accurate if the corners of modules were rounded.

Transformations. The parallel execution of moves constitutes a *transformation*, which we denote by the initial and resulting configurations, e.g., $C_1 \to C_2$. A transformation is valid only if it (i) preserves connectivity and (ii) does not cause collisions.

As in a number of existing models for both sequential and parallel reconfiguration of squares in the grid [1, 4, 22, 31], connectivity is determined by the existence of a *connected backbone*: Given a configuration C, we call a subset M of modules *free* if $C \setminus M'$ is a valid configuration for any $M' \subseteq M$. Let $M \subset C_1$ refer to all moving modules during $C_1 \to C_2$. Then there exists a connectivity-preserving backbone if and only if M is free both in C_1 and in C_2. A module m is *free* if $\{m\}$ is free. In Figure 3, both transformations are illegal as they attempt to move modules that are not free (conflicts are marked in red).

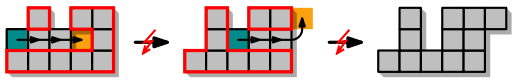


Figure 3 Moving non-free modules causes disconnections and makes transformations illegal.

Secondly, a transformation is collision-free exactly if all modules can move along their designated path at a constant rate, completing it in unit time without overlapping with any other module in the process. We illustrate a partial selection of collisions in Figure 4. Similar to the sequential variant, the defined moves and transformations are reversible, i.e., if for any transformation $C_1 \to C_2$, there exists an inverse transformation $C_2 \to C_1$.

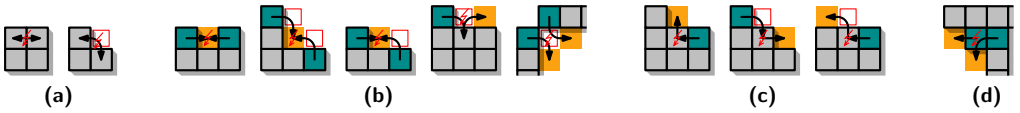


Figure 4 Collisions: (a) Modules cannot swap places or (b) enter the same cell simultaneously, (c) moves cannot meet at endpoints orthogonally, and (d) speed mismatches must be avoided.

Problem statement. The Parallel Sliding Squares problem, asks for the connected reconfiguration of modules in the infinite integer grid by legal transformations, with an *instance* $\mathcal{I}$ being composed of two configurations (C_1, C_2) of n modules. A schedule is *feasible* for $\mathcal{I}$ if and only if it transforms C_1 into C_2. The *makespan* of a schedule is the number of transformations in it. The decision problem for Parallel Sliding Squares asks, given an instance $\mathcal{I}$ and $k \in \mathbb{N}^+$, whether there is a feasible schedule for $\mathcal{I}$ with makespan at most k.

Labeled problem variant. In the case of distinguishable (or *labeled*) modules, we speak of the Labeled Parallel Sliding Squares problem. This variant uses the exact same move set and collision constraints as the unlabeled variant, but configurations are now mappings between $[n]$ and $\mathbb{Z}^2$ rather than subsets thereof. This allows us to specify individual target positions and track the movement of individual modules across transformations.

Useful notation. Throughout this paper, we make use of cardinal and ordinal directions. In particular, the unit vector $(1, 0)$ points *east*, $(0, 1)$ points *north*, and their opposite vectors point *west* and *south*, respectively. For an illustration, see Figure 2(a). Throughout this paper, we use B_1 and B_2 to denote the bounding boxes of C_1 and C_2, respectively. We assume, as in the literature [25], that B_1 and B_2 share a south-west corner at $(0, 0)$.

We define the (open) neighborhood $N(c)$ of a cell c as the set of cells that are edge-adjacent to c, and the closed neighborhood $N[c]$ as $N(c) \cup \{c\}$. We abuse notation to express neighborhoods of sets of cells as $N[S] = \bigcup_{c \in S} N[c]$ and $N(S) = N[S] \setminus S$. Analogously, define $N^*(S)$ (resp., $N^*[S]$) as the open (resp., closed) neighborhood using vertex-adjacency.

A feasible schedule for $\mathcal{I}$ is (*strictly*) *in-place* (as defined in [4, 25]) if and only if no intermediate configuration exceeds the union $B_1 \cup B_2$ by more than one module. We relax this constraint slightly and say that a schedule is *weakly in-place* if no intermediate configuration exceeds the union $B_1 \cup B_2$ by more than a constant number of units. Note that we do not restrict the number of modules located outside the union, only their distance to it. This corresponds to a slightly more restricted variant of the definition used in [1].

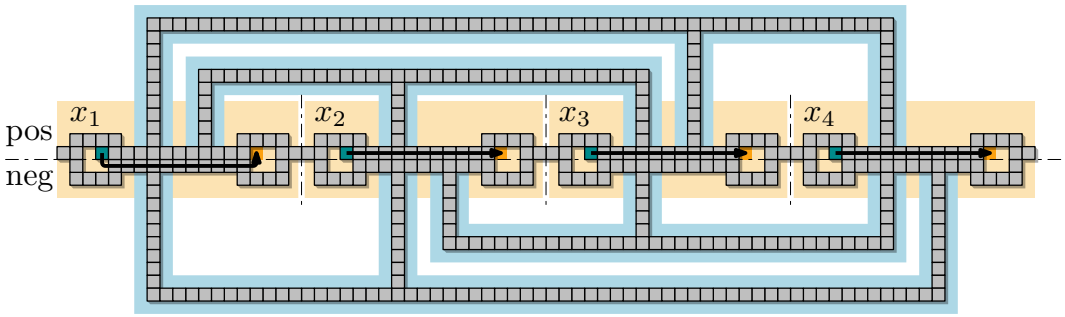
3 Computational complexity

We provide a number of complexity results for Parallel Sliding Squares, which are complementary to the NP-completeness result obtained by the authors of [4] for the sequential variant of this problem. This highlights a previously unrecognized gap in complexity, compared to closely related models for parallel, connected reconfiguration without the connected backbone constraint. In particular, in the related model studied in [16, 18], deciding the existence of schedules of makespan 2 is NP-complete, while makespan 1 is in P. We give a high-level description of our construction for the unlabeled case and refer to the full version [5] for technical details, as well as the labeled problem variant.

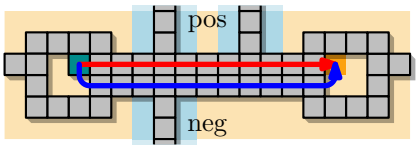
▶ **Theorem 1.** *Let $\mathcal{I}$ be an instance of Parallel Sliding Squares. It is NP-complete to decide whether there exists a feasible schedule of makespan 1 for $\mathcal{I}$.*

We reduce from the NP-complete problem Planar Monotone 3Sat [11], which asks whether a Boolean formula in conjunctive normal form is satisfiable. Each clause consists of at most 3 literals, all either positive or negative, and the clause-variable incidence graph must admit a plane drawing where variables are mapped to the x-axis, positive (resp., negative) clauses are mapped to the upper (resp., lower) half-plane, and edges do not cross the x-axis.

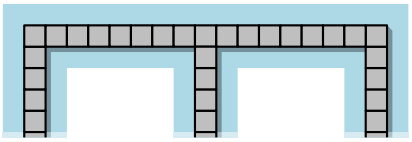
Our reduction starts with a rectilinear embedding of the clause-variable incidence graph of an instance φ of Planar Monotone 3Sat and constructs an instance $\mathcal{I}_\varphi$ of Parallel Sliding Squares using *variable* and *clause gadgets*. We then argue that φ can be satisfied if and only if there is a single parallel transformation that solves $\mathcal{I}_\varphi$. For a formula φ over m variables with k clauses, we create m copies of the variable gadget in a horizontal line, directly adjacent to one another. The literals in each monotone positive (negative) clause are then connected to by a clause gadget placed above (below) the horizontal line of variables. A simple example of our construction is depicted in Figure 5(a).



(a) Our construction for $\varphi = (x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$. The depicted transformation represents the satisfying assignment $\alpha(\varphi) =$ (`true`, `false`, `false`, `false`).



(b) Variable gadget (blue/red ≡ `true`/`false`).



(c) Clause gadget.

Figure 5 An overview of our hardness reduction and the two types of gadget used.

Variable gadget. The variable gadget consists of two cycles of 12 modules each, the left containing an additional module in its interior, see Figure 5(b). These circles are connected by a solid, horizontal *assignment strip* of height 2, to which individual clause gadgets (blue) are connected. In the target configuration, the extra module is located in the other circle. There are two feasible transformations, each representing either a `true` or `false` value assignment.

Clause gadget. The clause gadget consists of a thin horizontal strip that spans the gadgets of variables contained in the clause, and (up to) three vertical prongs that connect to the assignment strips of the incident variable gadgets. No squares move here, see Figure 5(c).

Due to the way in which both configurations are connected, the schedules depicted in Figure 5(b) locally sever the direct connection between the variable gadget and either all its positive or negative clauses. Thus, a satisfying assignment maps to a feasible transformation.

▶ **Corollary 2.** *Parallel Sliding Squares is* APX*-hard and cannot be approximated within a factor better than* 2 *in polynomial time unless* P = NP.

4 A worst-case optimal algorithm

In this section, we introduce a three-phase algorithm for efficiently reconfiguring two given configurations into one another. Our approach computes reconfiguration schedules linear in the perimeter P_1 and P_2 of the bounding boxes B_1 and B_2 of C_1 and C_2, respectively.

▶ **Theorem 3.** *For any instance $\mathcal{I}$ of Parallel Sliding Squares, we can compute a feasible, weakly in-place schedule of $\mathcal{O}(P_1 + P_2)$ transformations in polynomial time.*

Our algorithm consists of three phases as depicted in Figure 6: In **Phase (I)**, we identify a subconfiguration used as a "backbone" (similar to [21]), and gather $\Theta(P_1)$ many modules around a piece of this backbone, thereby enhancing its connectivity (as in [4]). We use the flexibility of this piece to construct a sweep-line structure out of meta-modules in **Phase (II)**, that is then used in **Phase (III)** to efficiently compact and transform the remaining modules into grid-aligned 3×3 squares, creating a 3-scaled configuration. In general, a configuration is *c-scaled* exactly if it is composed of $c \times c$ squares that are aligned with a corresponding $c \times c$ integer grid. By generalizing techniques from [16], we can reconfigure these efficiently:

▶ **Theorem 4.** *For any 3-scaled instance $\mathcal{I}$ of Parallel Sliding Squares, we can compute a feasible, in-place schedule of $\mathcal{O}(P_1 + P_2)$ transformations in polynomial time.*

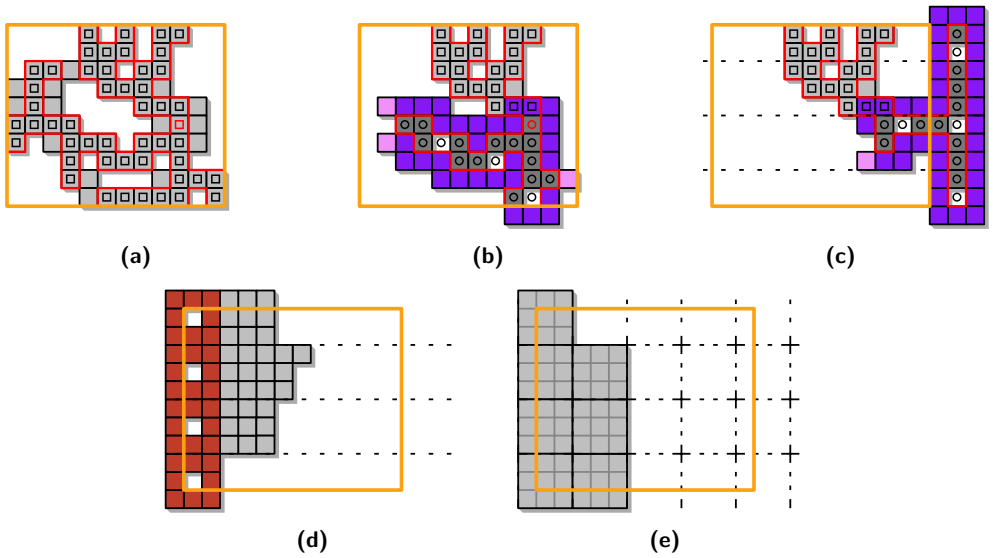


Figure 6 The high-level overview of our approach. (a) Initial configuration and (b), (c) and (e) are the result of Phases **(I–III)**, respectively. (d) shows an intermediate configuration in Phase **(III)**.

To reach our target configuration, we then simply apply **Phases (I-III)** in reverse. Although several of our techniques are inspired by previous work, they differ substantially in our context of parallel reconfiguration and considerable changes were needed.

▶ **Lemma 5.** *The bounds achieved in Theorem 3 are asymptotically worst-case optimal.*

Proof. Assume that n is even and consider two configurations C_1, C_2 contained in an $n/2 \times n/2$ bounding box B. Define C_1 to contain the $n-1$ modules adjacent to the left and bottom edges of B and a module at $(1,1)$. Define C_2 to contain the $n-1$ modules adjacent to the top and right edges of B and a module at $(n/2-1, n/2-1)$. Then, the minimum bottleneck matching between full cells in C_1 and C_2 has bottleneck $\Omega(n)$, implying that a module must make $\Omega(n)$ moves. The makespan is then $\Omega(n) = \Omega(P)$, where P is the perimeter of B. ◀

4.1 Phase (I): Gathering squares

In **Phases (I)** and **(II)**, we use an underlying connected substructure (called *skeleton*) of the initial configuration to guide the reconfiguration. Intuitively (formal definitions below), this tree-like skeleton functions as a backbone around which we move modules toward a "root" module, making a subtree "thick" (where the cells in the neighborhood of part of the skeleton are all full). This "thick" subskeleton is more "manipulable", making it easier to mold it into a sweep line in **Phase (II)**. Moving modules around a tree is an idea also used by the authors of [21]. This type of strategy becomes complicated when the tree creates bottlenecks where potential collisions happen, which they solve by strengthening the model and allowing modules to "squeeze through" bottlenecks. We achieve a stronger result in the classic sliding model via careful definition of the tree-like structure and movement schedules.

Skeleton. We define the *skeleton* of a configuration C as a valid subconfiguration S such that: $C \subset N[S]$ (every module of C is either contained in S or edge-adjacent to a module of S); and cycles in the dual graph of S are pairwise disjoint with length at most 4. We call a module in S (resp., not in S) a skeleton (resp., nonskeleton) module. In Figure 6(a), skeleton modules are highlighted with an internal square and the perimeter of the skeleton is shown in red. Note that any set of nonskeleton modules is free.

On a high level, we can compute a skeleton S of a given configuration C as follows. Think of S as a subset of C, initialized as the empty set. Modules of C are then algorithmically added to S until it satisfies the definition of a skeleton. First, all modules with even x-coordinate are added to S, then all modules with odd x-coordinate with no east and west neighbors. Next modules are added to S until it is a connected subconfiguration of C (see magenta squares in Figure 6(a)). This might introduce large cycles (greater than length 4) to S. These cycles are broken via removal of modules from S or exchanging adjacent modules. We show that careful execution of these steps will build S into a skeleton of C.

Equipped with a skeleton S, we now consider its dual graph. Contracting every cycle in the dual of S to a single vertex renders a max-degree-4 tree where each node is either a cell or a 4-cycle. We refer to the nodes of this tree as *nodes of* S and abuse notation by referring to S as a tree. Let r be the *root* node of S, arbitrarily chosen. For every nonskeleton module, we assign an adjacent module of S as its *support*. We define the *subskeleton rooted at* c (denoted S_c) as the subconfiguration containing c and its descendant nodes. The set S_c^* contains the modules in S_c and the nonskeleton modules supported by modules in S_c. The *weight* of a subskeleton S_c is defined as $|S_c^*|$. By definition, $S_r^* = C$ and $|S_r^*| = n$.

▶ **Lemma 6.** *Given a rooted skeleton S and an integer $1 < w \leq n$, there exists a node c of S such that $w \leq |S_c^*| \leq 3w$.*

Using this, we locate a module $h \in S$ (highlighted with a red marker in Figures 6(a) and 6(b)) such that $|S_h^*| \in \Theta(P)$. Afterwards, we "thicken" S_h^* into a structure called the *exoskeleton*.

Exoskeleton. An exoskeleton is made from three types of modules: *core*, *shell*, and *tail* modules; depicted respectively in dark gray, purple and pink in the figures. Recall that we wish to make the neighborhood of S_h is full. The core modules occupy the positions originally occupied by the skeleton S_h, the shell modules are the ones "coating" the skeleton, and the tail modules are "leftover" modules in the neighborhood of a leaf. We recursively thicken S_h by applying this process to its smaller subtrees, effectively converting tail or core modules at the leaves into a shell module near the root. If the core modules cover the entire subtree of S_h, it may take a linear number of transformations to move a module from a leaf

to the root. Instead, we allow empty positions inside the exoskeleton which permits us to "teleport" a module from a leaf to the root in $\mathcal{O}(1)$ transformations. Although some of the positions are empty, the structure remains connected due to its shell and the fact that these empty positions are well separated (in tree metric along S_h, see Property 4 below).

Formally, we define an *exoskeleton* X_c as follows. The *core* $\overline{X_c}$ of X_c are positions in the lattice (not necessarily full) that form a subtree of S_c containing c (since we recursively "shave off" leaves to make material for the shell). Note that $\overline{X_c}$, unlike S_c, is not a subgraph of the configuration's dual graph because of the empty positions. In our figures, we highlight core cells with an internal circle. Let $\mathcal{L}$ be the set of positions corresponding to the leaves of $\overline{X_c}$. We call $N^*(\overline{X_c} \setminus \mathcal{L}) \setminus \mathcal{L}$ the *shell* of X_c (recall $N^*(S)$ is the open neighborhood under vertex-adjacency). The following must hold:

1. All modules in X_c are in the neighborhood of its core. ($|\overline{X_c}| \geq 2$, and $X_c \subset N^*\left[\overline{X_c}\right]$.)
2. The leaves are occupied. ($\mathcal{L} \subset X_c$.)
3. All cells in the shell are occupied. ($N^*(\overline{X_c} \setminus \mathcal{L}) \setminus \mathcal{L} \subset X_c$.)
4. The depth (in $\overline{X_c}$) of every empty cell is congruent to k (mod 4) for a fixed $k \in \{0, 1, 2, 3\}$.



Figure 7 Visualizing an exoskeleton. (a) Skeleton and core cells are highlighted with an inner square and circle, respectively. (b) Highlights the core; the root is highlighted with a red circle.

The modules that are in $N^*(\ell)$, for a leaf $\ell \in \mathcal{L}$, and not in the shell or core of X_c are called the *tail* modules of ℓ. By definition, a module in X_c is either in the shell, in the core, or is a tail module. Shell modules protect the connectivity of X_c, allowing us to place empty positions in the core wherever necessary to expand the exoskeleton constant time.

We show that S_h can be reconfigured into an exoskeleton X_h in $\mathcal{O}(P)$ transformations, thereby inductively establishing Lemma 9. To this end, we first prove Lemma 7, demonstrating how small skeletons (lighter than 9) are turned into exoskeletons. In doing so, we obtain Corollary 8, which settles the base case of our induction. Lemma 7 will also be pivotal in the inductive step. All that this lemma does is take a subskeleton of weight ≤ 9 and transform it into a 3×3 square, a minimal instance of an exoskeleton ($|X_c| = 2$).

▶ **Lemma 7.** *Let C be a configuration with skeleton S in which potentially some of its subskeletons were reconfigured into exoskeletons. Let c be a skeleton node, with parent node d, such that S_c is not yet part of an exoskeleton and $|S_c^*| \leq 9$. Let M be the set of modules in S_c^* that are not contained in exoskeletons. Then, in $\mathcal{O}(1)$ transformations, M can be reconfigured so that:*

- *if $\deg(d) = 2$, either $N^*[d]$ is full or $M \cap (N^*[c] \setminus N^*[d])$ is empty (i.e., the modules in M are all contained in the neighborhood of d if this region is not full);*
- *if $\deg(d) > 2$, either $N^*[d] \cap N^*[c]$ is full or $M \cap (N^*[c] \setminus N^*[d])$ is empty (i.e., the modules in M are all contained in the intersections of the neighborhood of d and c if this region is not full).*

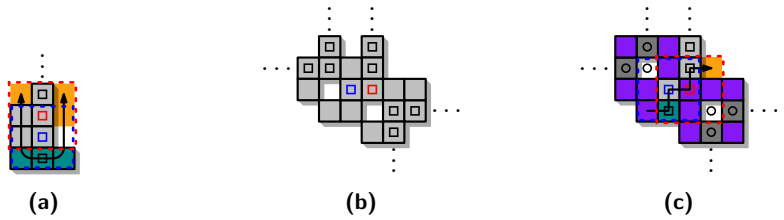


Figure 8 Illustration of Lemma 7. Modules d and c are highlighted with inner red and blue squares. The boundaries of $N^*[d]$ and $N^*[c]$ are shown with dashed red and blue lines.

At first glance, the lemma might seem overly complicated, since without any obstacle, modules can freely move along the perimeter of S (see Figure 8(a)). However, nonlocal pieces of the skeleton (connected subsets S_1 and S_2 of S are *local* if $S \cap N^*(S_1 \cup S_2)$ is connected) previously converted to exoskeletons can interfere when processing later subskeletons. Figures 8(b) and 8(c) show a configuration before and after two nonlocal pieces of the skeleton were converted into exoskeletons. Modules in the working subskeleton S_c might be part of the shell of another exoskeleton, making their movement dangerous (potentially disconnecting the nonlocal exoskeletons). We try to leave such modules in their place, changing their membership from S_c^* to the shell of the exoskeleton.

By applying Lemma 7 a constant number of times in appropriate subskeletons, we eventually obtain a solid 3×3 square from which we can construct an exoskeleton.

▶ **Corollary 8.** *Given a subskeleton S_c with $9 \leq |S_c^*|$, we can reconfigure S_c^* transforming one of its subskeletons into an exoskeleton in $\mathcal{O}(1)$ many transformations, without changing other exoskeletons or modules not in S_c^*.*

Corollary 8 provides the base case of our induction for Lemma 9. We now establish some routines for the inductive step. We assume there is a module $d \in S_h$ so that for every child c of d, S_c^* was reconfigured to an exoskeleton X_c satisfying Property 4 with $k = 0$. If $N^*[d]$ is full, we can finish by making d the root of the exoskeleton, effectively merging the children exoskeletons. Note that this will necessarily happen if d had degree 4 and is not part of a 4-cycle. Thus, assume that there is an empty position $e \in N^*[d]$. We apply a routine Inchworm-Push that fills e with local moves, using a module initially in the core of the exoskeleton X_c (making that cell empty). The routine Inchworm-Pull then moves core modules higher (in tree metric) pushing empty positions deeper until they "exit" the exoskeleton at leaves. Again, ignoring nonlocal interactions, these operations are straightforward. However, in the general case, these interactions mandate extra care.

- Inchworm-Push: Let c be a child of d, and p be the parent of d. Require that $N^*[c]$ is full. We branch into cases. (i) If c, d, and p are collinear, there is a path π in $N^*(d)$ from c to e going through only occupied cells, see Figure 9(a); (ii) If c, d and p form a bend, there is a path π in $N^*[d]$ from c to e going through d and at most one nonskeleton module, see Figures 9(b) and 9(c); (iii) In analogous cases to (i–ii), if a node in $\{c, d, p\}$ is a 4-cycle, there is a path π in $N^*(d)$ from a module in c to e going through one module in the shell of X_d, depicted in Figure 9(d). For all cases, move the modules along π making e occupied and c empty. This requires at most two transformations since there is at most one collision along π and we can decompose π into two paths with no collisions.
- Inchworm-Pull: The operation consists of four stages, each involving at most two transformations. For every empty cell p in the core, select one of its children q arbitrarily, and let $x(q)$ be its x-coordinate. Move the module from q to p in stage $j \in \{1, 2, 3, 4\}$ if

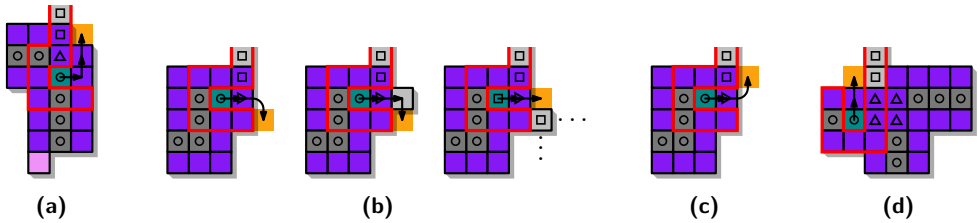


Figure 9 Inchworm-Push. Module c is colored turquoise, cell e is yellow, and d is highlighted with an inner triangle. The red curve encloses a subconfiguration around the moves that remains connected guaranteeing that the transformations do not disconnect the configuration.

$x(q) \equiv j \pmod 4$. Figure 10(a) gives two examples, in one of which p is a 4-cycle (that takes two transformations). If q is a leaf, if there is a tail module of q that is originally in S_c^* and not in another exoskeleton, move such a module to q, visualized in Figures 10(b) and 10(c). If there are no more tail modules we can update $\overline{X_c}$ by deleting q which causes the tail module that just moved to become a shell module, see Figure 10(b), or causes two shell modules to become tail modules as in Figure 10(c).

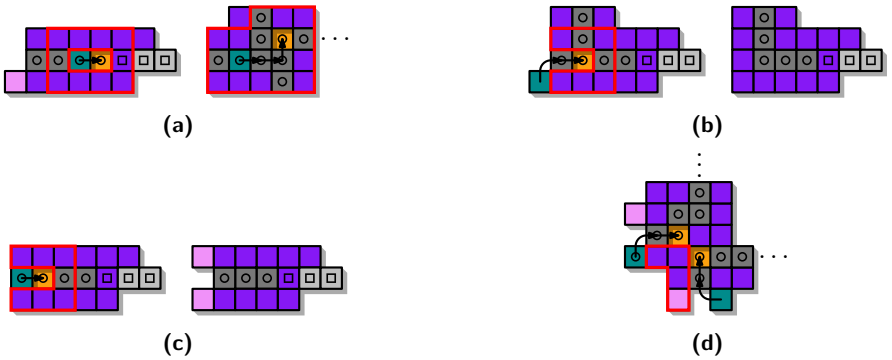


Figure 10 Inchworm-Pull. Module q is colored turquoise and empty cell p is yellow.

The reason we stagger the moves into four stages in Inchworm-Pull is to guarantee that the configuration contains enough static modules around a move. Otherwise, performing all moves in a single transformation might disconnect the configuration, as shown in Figure 10(d). In the full version [5], we argue that Inchworm-Push and Inchworm-Pull preserve connectivity and require only $\mathcal{O}(1)$ transformations. To establish local connectivity of the static modules in the neighborhood of each move, we rely on Properties 3–4.

▶ **Lemma 9.** *Given a configuration C with skeleton S, and a skeleton node d with $|S_d^*| \geq 9$, $\mathcal{O}(|S_d^*|)$ transformations are sufficient to reconfigure S_d into an exoskeleton X_d.*

Proof. If a subskeleton of S_d does not contain an exoskeleton, we can apply Corollary 8. If, after that, S_d is not the core of an exoskeleton X_d, we apply induction. Let Q be the set containing the children of d. For every child $c \in Q$ with $|S_c^*| \geq 9$ we can get an exoskeleton X_c in $\mathcal{O}(|S_c^*|)$ transformations by inductive hypothesis. For every child $c \in Q$ with $|S_c^*| < 9$, we apply Lemma 7 that either results in an exoskeleton X_c, or results in deleting c from S "compacting" all its modules in $N^*[c] \cap N^*[d]$. If after that $N^*[d]$ is not full, $N^*[d]$ contains at most three empty cells if d is a degree-2 bend in S, at most two empty cells if d is a "straight" degree-2 in S, or at most 1 empty cell if d has degree 3. (Note that if d has degree 4

and is not a 4-cycle, then $N^*[d]$ is full.) For each of these empty cells, we can fill them by applying Inchworm-Push followed by at most four applications of Inchworm-Pull to ensure that $N^*[c]$ is full for all children c of d while maintaining Property 4. This takes at most $\mathcal{O}(1)$ transformations. When $N^*[d]$ is full, the configuration contains an exoskeleton X_d except for the "4-arity" of the empty positions (Property 4). That can be resolved by applying Inchworm-Pull to X_c at most three times for each child c of d. ◀

By Lemma 6, we can choose an appropriately heavy node d of S to obtain:

▶ **Corollary 10.** *Let C be a configuration with bounding box perimeter P. We can reconfigure C so that it contains an exoskeleton X_h with $\geq 36P$ modules in $\mathcal{O}(P)$ transformations. All modules stay within $\mathcal{O}(1)$ distance from the bounding box of C.*

4.2 Phase (II): Scaffolding

Phase (I) of our algorithm recursively constructed a sufficiently large exoskeleton X_h that is now reconfigured in **Phase (II)**. The goal of this reconfiguration is to create a $\dashv$ -shaped exoskeleton, with its vertical segment precisely aligned along the right boundary of the bounding box. This vertical edge will serve as the sweep line in **Phase (III)**.

Phase (II) of our approach consists of the following three steps:

1. First "compact" X_h so that its core contains no empty cells.
2. We then choose a rightmost node c in the core of X_h and use Inchworm-Push and -Pull to grow a horizontal path from c rightward, until a 3×3 square is formed outside the bounding box, see Figures 11(a)–11(f).
3. Use Inchworm-Push and -Pull to grow the path upwards until it hits the top of the bounding box, then grow a path downwards until it hits the bottom of the bounding box, see Figures 11(f)–11(h).

We make sure that X_c always contains h so that it maintains connectivity with the subset of S that was not transformed into an exoskeleton. We can show that the connectivity properties of exoskeletons allow us to "go through" obstacles (sections of S that remained untouched in **Phase (I)**), see Figures 11(a)–11(f). That may require some extra moves as shown in Figure 11(b), where an Inchworm-Push would move the cut vertex marked with a red $\times$. In such cases, we sequentially fill in up to two cells in the neighborhood of X_c (yellow in Figure 11(b)), making all modules in π (from the definition of Inchworm-Push) noncut.

4.3 Phase (III): Creating scale

Once **Phase (II)** concludes, we are left with an intermediate configuration that contains a scaffold structure along its east boundary, recall Figure 11(h). In the remaining phase, we use this scaffold to obtain a 3-scaled configuration that can be arbitrarily reconfigured due to Theorem 4. To build squares in an efficient manner, we modify part of the exoskeleton structure created during **Phase (II)**, creating a vertical line of 3×3 meta-modules.

Sweep line. We define a scaffold structure composed of *meta-modules*, each consisting of eight modules occupying the open vertex-neighborhood of a common center cell v, i.e., one in every cell of $N^*(v)$ as shown in Figure 12(a). A *sweep line* $\ell \subset C$ is then a valid subconfiguration formed by the union of h disjoint meta-modules $M_1, \ldots, M_h$ with identical x-coordinates such that ℓ spans the full height of the bounding box of $C \setminus \ell$. Let v_i now refer to the center cell of M_i, such that $y(v_i) < y(v_j)$ exactly if $i < j$, see Figures 12(b) and 12(c).

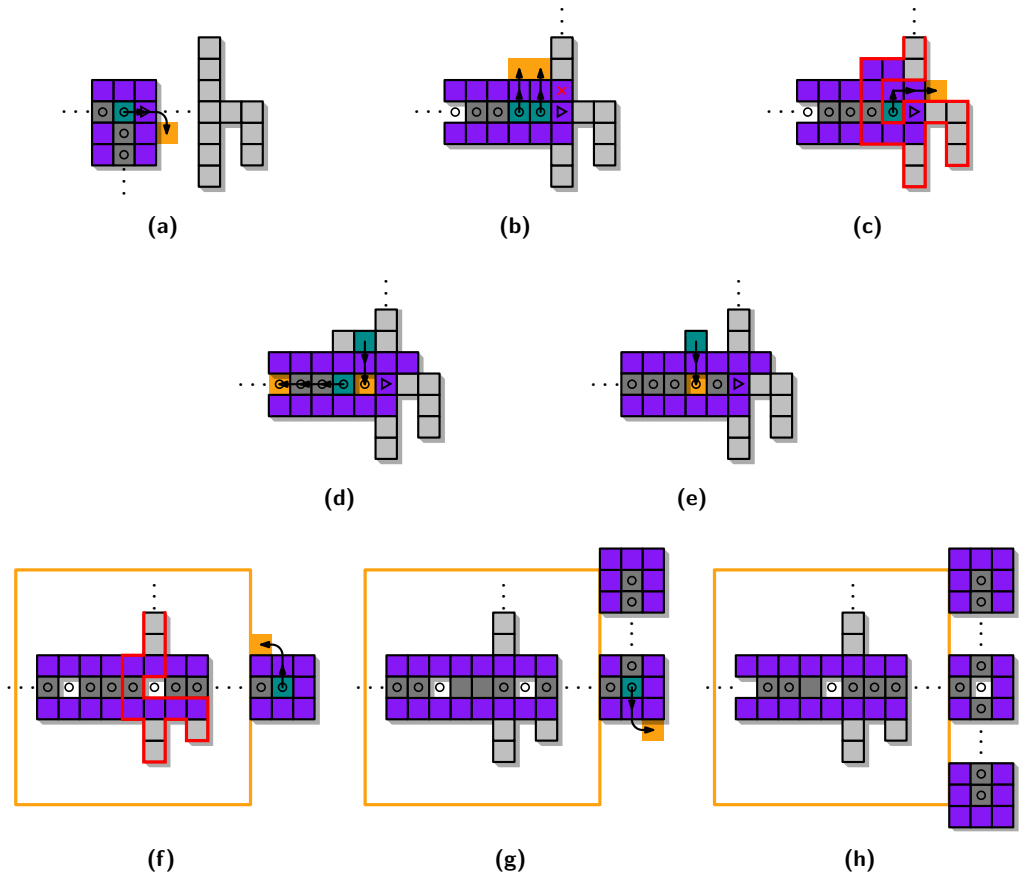


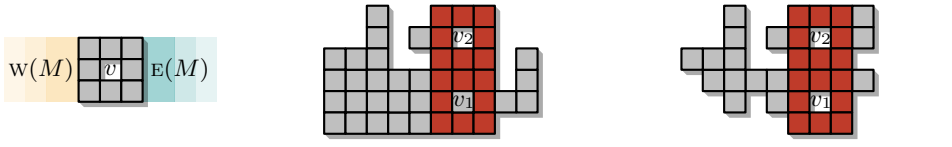
Figure 11 Building the scaffolding; bounding box of the initial configuration is shown in yellow.

For each meta-module M in a sweep line $\ell \subseteq C$, we define as its east and west strips as the cells which share a y-coordinate with one of its modules and are located within the bounding box of the full configuration C. We denote these by $\textsc{e}(M)$ and $\textsc{w}(M)$, respectively, and refer to their union for ℓ, e.g., $\textsc{w}(\ell)$. We use these to define the following characteristics.

A sweep line ℓ is *clean* exactly if every meta-module M_i either has an empty center cell v_i, or its respective west strip is fully occupied, i.e, $\textsc{w}(M_i) \subseteq C$. See Figure 12(b) for an illustration. Analogously, we call ℓ *solid* if all center cells of its meta-modules are occupied.

Furthermore, we say that is ℓ a *separator* if for every meta-module M_i in ℓ:

1. The east strip $\textsc{e}(M_i)$ does not contain modules unless all cells in $\textsc{w}(M_i)$ are full.
2. The modules in $\textsc{e}(M_i)$ are located in its x-minimal cells, thus forming a 3 cell tall rectangle with at most two additional "loose" modules.



(a) A meta-module M. **(b)** A clean sweep line (red). **(c)** A separator sweep line (red).

Figure 12 Sweep lines are formed by vertical stacks of meta-modules.

We start by transforming part of the exoskeleton from **Phase (II)** into meta-modules that form a sweep line at the eastern boundary of B.

▶ **Lemma 11.** *The vertical portion of the "⊣-shaped" exoskeleton in C can be made into a separator sweep line in $\mathcal{O}(1)$ transformations.*

This gives us a separator sweep line ℓ that is not necessarily clean or solid, located at the east edge of the configuration's bounding box B. Our goal is to move ℓ towards the west edge of B while maintaining its separator property, compacting all encountered modules into 3-tall strips in the process. We define two procedures that suffice to reach this goal.

Intuitively, the Advance operation provides a schedule that translates a sweep line by one unit to the west, and the Clean operation replaces a given sweep line with a clean one. To efficiently parallelize these operations, we split the sweep line into *leading* and *trailing* meta-modules such that M_i is trailing exactly if i is odd, and leading otherwise. At any given time, either the leading or the trailing meta-modules are active, but never both.

By repeatedly alternating between the Advance and Clean operations, we create a configuration with a clean separator sweep line that has an empty west half-space, see Figure 13.

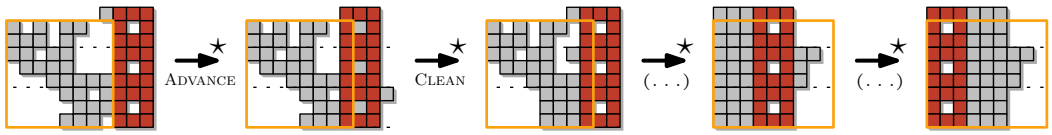


Figure 13 We alternate between the Advance and Clean operations until $\mathrm{w}(\ell) \cap C = \varnothing$.

We first argue the Clean operation can be performed in constantly many transformations. Let M_i be an active meta-module with an occupied center cell. On a high-level, we consider a rectangle induced by occupied cells in west direction within its strip. It is easy to see that we can fill an empty position immediately adjacent to this rectangle, while also cleaning the center cell of M_i by at most two subsequent transformations. By doing this for leading and trailing meta-modules separately, we conclude the following lemma.

▶ **Lemma 12.** *A configuration C with a sweep line ℓ can be cleaned by 4 strictly in-place transformations, without moving any modules east, or exceeding the bounding box B of C.*

It remains to argue that the same is true of the Advance operation. For this, we consider the three positions immediately west of an active meta-module, and distinguish eight cases based on occupied and empty cells. For all of them, we define local transformation schedules that enable us to move a meta-module one step to the east. Thus, we obtain:

▶ **Lemma 13.** *A configuration C with a clean separator sweep line ℓ can be advanced into a clean separator sweep line by $\mathcal{O}(1)$ strictly in-place transformations if $C \cap \mathrm{w}(\ell) \neq \{\varnothing\}$.*

After no more than $\mathcal{O}(P)$ many iterations consisting of Clean+Advance, we obtain a state in which $\mathrm{w}(\ell) \cap C = \varnothing$. The separator property implies that the modules in $\mathrm{e}(\ell)$ form 3-wide, horizontal stacks. It remains to show that we can make this configuration 3-scaled.

To simplify our arguments, assume that n is a multiple of nine. In the general case, n can exceed a multiple of nine by at most 8 modules. We can place these at the south-west corner of the extended bounding box B' and locally ensure the existence of a connected backbone during each transformation.

▶ **Lemma 14.** *Let C contain a clean separator sweep line ℓ with an empty west half-space. Then C can be transformed into a 3-scaled configuration in $\mathcal{O}(P)$ transformations.*

On a high level, we achieve this by filling the center cells of all meta-modules and balancing the remaining modules between their east strips such that each contains a multiple of nine. Having constructed a 3-scaled configuration, we can arbitrarily reconfigure due to Theorem 4 before applying **Phases (I-III)** in reverse to obtain our target configuration.

5 Conclusions and future work

We have presented several new results for reconfiguration in the sliding squares model, making full use of parallel motion. In particular, we showed that deciding whether there exists a single transformation that reconfigures one configuration into another is already NP-complete. Additionally, we provided algorithmic results for the unlabeled variant, including a polynomial-time algorithm that performs in-place reconfiguration in $\mathcal{O}(P)$ transformations, where P denotes the perimeter of the union of the configurations' bounding boxes.

We note that this algorithm can easily be adapted to the labeled setting by "sorting" the modules in the xy-monotone configuration using $\mathcal{O}(P)$ transformations. However, this requires a relaxation of the in-place requirement. Furthermore, deciding whether a schedule of makespan 1 exists in the labeled setting is straightforward, while the problem becomes NP-complete for a makespan of 2. Details on these results are provided in the full version [5].

Our algorithmic results are only worst-case optimal, and there are a number of possible generalizations and extensions of the setting. Previous work in the sequential setting has progressed from two dimensions to three. Can our approach be extended to higher dimensions? We are optimistic that significant speedup can be achieved, but the intricacies of three-dimensional topology may require additional tools.

Another aspect is a refinement of the involved parameters, along with fixed parameter tractability. Our hardness proofs show that the investigated problems are not FPT when parameterized by the number of transformations in the output. The number of modules in the input configuration is not a suitable parameter since it is the input size. Are there other parameters for which these problems are FPT? An interesting candidate would be the size of the symmetric difference between the two input configurations. In our hardness proofs, the size of the symmetric difference is linear on the size of the SAT instance.

References

1 Zachary Abel, Hugo A. Akitaya, Scott Duke Kominers, Matias Korman, and Frederick Stock. A universal in-place reconfiguration algorithm for sliding cube-shaped robots in a quadratic number of moves. In *Symposium on Computational Geometry (SoCG)*, pages 1:1–1:14, 2024. doi:10.4230/LIPIcs.SoCG.2024.1.

2 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmovic, Robin Y. Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $\mathcal{O}(1)$ musketeers. *Algorithmica*, 83(5):1316–1351, 2021. doi:10.1007/s00453-020-00784-6.

3 Hugo A. Akitaya, Erik D. Demaine, Andrei Gonczi, Dylan H. Hendrickson, Adam Hesterberg, Matias Korman, Oliver Korten, Jayson Lynch, Irene Parada, and Vera Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In *Symposium on Computational Geometry (SoCG)*, pages 10:1–10:20, 2021. doi:10.4230/LIPIcs.SoCG.2021.10.

4 Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. In *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 4:1–4:19, 2022. doi:10.4230/LIPIcs.SWAT.2022.4.

5 Hugo A. Akitaya, Sándor P. Fekete, Peter Kramer, Saba Molaei, Christian Rieck, Frederick Stock, and Tobias Wallner. Sliding squares in parallel, 2025. doi:10.48550/arXiv.2412.05523.

6 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Robin Flatland, Stefan Langerman, Joseph O'Rourke, Suneeta Ramaswami, Vera Sacristán, and Stefanie Wuhrer. Linear reconfiguration of cube-style modular robots. *Computational Geometry*, 42(6-7):652–663, 2009. doi:10.1016/J.COMGEO.2008.11.003.

7 Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Matthias Konitzny, Lillian Lin, and Christian Scheffer. Coordinated motion planning: The video. In *Symposium on Computational Geometry (SoCG)*, pages 74:1–74:6, 2018. doi:10.4230/LIPICS.SOCG.2018.74.

8 Julien Bourgeois, Sándor P. Fekete, Ramin Kosfeld, Peter Kramer, Benoît Piranda, Christian Rieck, and Christian Scheffer. Space ants: Episode II - Coordinating connected catoms. In *Symposium on Computational Geometry (SoCG)*, pages 65:1–65:6, 2022. doi:10.4230/LIPIcs.SoCG.2022.65.

9 Zack Butler and Daniela Rus. Distributed planning and control for modular robots with unit-compressible modules. *The International Journal of Robotics Research*, 22(9):699–715, 2003. doi:10.1177/02783649030229002.

10 Matthew Connor, Othon Michail, and George Skretas. All for one and one for all: An $\mathcal{O}(1)$-musketeers universal transformation for rotating robots. In *Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, pages 9:1–9:20, 2024. doi:10.4230/LIPIcs.SAND.2024.9.

11 Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *International Journal on Computational Geometry and Applications*, 22(3):187–206, 2012. doi:10.1142/S0218195912500045.

12 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019. doi:10.1137/18M1194341.

13 Adrian Dumitrescu and János Pach. Pushing squares around. *Graphs and Combinatorics*, 22(1):37–50, 2006. doi:10.1007/s00373-005-0640-1.

14 Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Formations for fast locomotion of metamorphic robotic systems. *International Journal of Robotics Research*, 23(6):583–593, 2004. doi:10.1177/0278364904039652.

15 Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Motion planning for metamorphic systems: feasibility, decidability, and distributed reconfiguration. *Transactions on Robotics*, 20(3):409–418, 2004. doi:10.1109/TRA.2004.824936.

16 Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected coordinated motion planning with bounded stretch. *Autonomous Agents and Multi-Agent Systems*, 37(2):43, 2023. doi:10.1007/S10458-023-09626-5.

17 Sándor P. Fekete, Ramin Kosfeld, Peter Kramer, Jonas Neutzner, Christian Rieck, and Christian Scheffer. Coordinated motion planning: Multi-agent path finding in a densely packed, bounded domain. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 29:1–29:15, 2024. doi:10.4230/LIPIcs.ISAAC.2024.29.

18 Sándor P. Fekete, Peter Kramer, Christian Rieck, Christian Scheffer, and Arne Schmidt. Efficiently reconfiguring a connected swarm of labeled robots. *Autonomous Agents and Multi-Agent Systems*, 38(2):39, 2024. doi:10.1007/S10458-024-09668-3.

19 Robert Fitch, Zack J. Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2460–2467, 2003. doi:10.1109/IROS.2003.1249239.

20 Toshio Fukuda, Seiya Nakagawa, Yoshio Kawauchi, and Martin Buss. Structure decision method for self organising robots based on cell structures-cebot. In *International Conference on Robotics and Automation (ICRA)*, pages 695–696, 1989. doi:10.1109/ROBOT.1989.100066.

21 Ferran Hurtado, Enrique Molina, Suneeta Ramaswami, and Vera Sacristán Adinolfi. Distributed reconfiguration of 2d lattice-based modular robotic systems. *Autonomous Robots*, 38(4):383–413, 2015. doi:10.1007/S10514-015-9421-8.

22 Irina Kostitsyna, Tim Ophelders, Irene Parada, Tom Peters, Willem Sonke, and Bettina Speckmann. Optimal in-place compaction of sliding cubes. In *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 31:1–31:14, 2024. doi:10.4230/LIPICS.SWAT.2024.31.

23 Keith D. Kotay and Daniela L. Rus. Algorithms for self-reconfiguring molecule motion planning. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2184–2193, 2000. doi:10.1109/IROS.2000.895294.

24 Othon Michail, George Skretas, and Paul G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 102:18–39, 2019. doi:10.1016/j.jcss.2018.12.001.

25 Joel Moreno and Vera Sacristán. Reconfiguring sliding squares in-place by flooding. In *European Workshop on Computational Geometry (EuroCG)*, pages 32:1–32:7, 2020. URL: https://www1.pub.informatik.uni-wuerzburg.de/eurocg2020/data/uploads/papers/eurocg20_paper_32.pdf.

26 An Nguyen, Leonidas J. Guibas, and Mark Yim. Controlled module density helps reconfiguration planning. *New Directions in Algorithmic and Computational Robotics*, pages 23–36, 2001.

27 Irene Parada, Vera Sacristán, and Rodrigo I. Silveira. A new meta-module design for efficient reconfiguration of modular robots. *Autonomous Robots*, 45(4):457–472, 2021. doi:10.1007/S10514-021-09977-6.

28 Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10:107–124, 2001. doi:10.1023/A:1026504804984.

29 Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: III. Coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983. doi:10.1177/027836498300200304.

30 Serguei Vassilvitskii, Mark Yim, and John Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *International Conference on Robotics and Automation (ICRA)*, pages 117–122, 2002. doi:10.1109/ROBOT.2002.1013348.

31 Matthijs Wolters. Parallel algorithms for sliding squares. Master's thesis, Utrecht University, 2024.

32 Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *Robotics and Automation Magazine*, 14(1):43–52, 2007. doi:10.1109/MRA.2007.339623.

The Tape Reconfiguration Problem and Its Consequences for Dominating Set Reconfiguration

Nicolas Bousquet
CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69622 Villeurbanne, France
CNRS - Université de Montréal CRM – CNRS, Canada

Quentin Deschamps
CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69622 Villeurbanne, France

Arnaud Mary
Université de Lyon, Université Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive UMR 5558, 69622 Villeurbanne, France
ERABLE team, Inria Lyon, Villeurbanne, France

Amer E. Mouawad
Department of Computer Science, American University of Beirut, Lebanon

Théo Pierron
CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69622 Villeurbanne, France

Abstract

A dominating set of a graph $G = (V, E)$ is a set of vertices $D \subseteq V$ whose closed neighborhood is V, i.e., $N[D] = V$. We view a dominating set as a collection of tokens placed on the vertices of D. In the token sliding variant of the Dominating Set Reconfiguration problem (TS-DSR), we seek to transform a source dominating set into a target dominating set in G by sliding tokens along edges, and while maintaining a dominating set all along the transformation.

TS-DSR is known to be PSPACE-complete even restricted to graphs of pathwidth w, for some non-explicit constant w and to be XL-complete parameterized by the size k of the solution. The first contribution of this article consists in using a novel approach to provide the first explicit constant for which the TS-DSR problem is PSPACE-complete, a question that was left open in the literature.

From a parameterized complexity perspective, the token jumping variant of DSR, i.e., where tokens can jump to arbitrary vertices, is known to be FPT when parameterized by the size of the dominating sets on nowhere dense classes of graphs. But, in contrast, no non-trivial result was known about TS-DSR. We prove that DSR is actually much harder in the sliding model since it is XL-complete when restricted to bounded pathwidth graphs and even when parameterized by k plus the feedback vertex set number of the graph. This gives, for the first time, a difference of behavior between the complexity under token sliding and token jumping for some problem on graphs of bounded treewidth. All our results are obtained using a brand new method, based on the hardness of the so-called Tape Reconfiguration problem, a problem we believe to be of independent interest.

We complement these hardness results with a positive result showing that DSR (parameterized by k) in the sliding model is FPT on planar graphs, also answering an open problem from the literature.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Combinatorics

Keywords and phrases combinatorial reconfiguration, parameterized complexity, structural graph parameters, treewidth, dominating set

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.29

Related Version *Full Version*: https://arxiv.org/abs/2505.00988



© Nicolas Bousquet, Quentin Deschamps, Arnaud Mary, Amer E. Mouawad, and Théo Pierron;
licensed under Creative Commons License CC-BY 4.0



Funding *Nicolas Bousquet*: The author has been partly supported by ANR project ENEDISC (ANR-24-CE48-7768-01) and by Campus-France PHC Cedre (ProLo).
Théo Pierron: The author has been partly supported by ANR projects ENEDISC (ANR-24-CE48-7768-01) and P-GASE (ANR-21-CE48-0001-01).

1 Introduction

The *combinatorial reconfiguration* framework aims to investigate algorithmic and structural aspects of the solution space of an underlying *base problem*. For instance, given an instance of some base problem along with two feasible solutions, i.e., the source and target feasible solutions, the goal is to determine if (and in how many steps) one can transform the source to the target via a sequence of adjacent feasible solutions. Such a sequence is called a *reconfiguration sequence* and every step in the sequence (going from one solution to an adjacent one) is called a *reconfiguration step*. Reconfiguration problems arise in various fields such as combinatorial games, motion of robots, random sampling, and enumeration. It has been extensively studied for various rules and types of (base) problems such as satisfiability [25, 37], graph colorings [11, 19], vertex covers and independent sets [29, 30, 34] and matchings [12]. The reader is referred to the surveys [39, 40, 18] for a more complete overview of the field.

An alternative view of reconfiguration problems is via the notion of so-called (re)configuration graphs. Let Π be a base problem and let $\mathcal{I}$ be an instance of Π. The *configuration graph* $\mathcal{R}_\mathcal{I}$ is a graph whose nodes correspond to feasible solutions of $\mathcal{I}$ and in which two solutions are adjacent if and only if we can transform the first into the second in a single reconfiguration step. In this paper, we focus on the reachability question that asks, given two solutions S, S' of $\mathcal{I}$, whether there exists a reconfiguration sequence from S to S'. In other words, does there exist a path between S and S' in the configuration graph. Other works have focused on different problems such as the connectivity of the configuration graph or the diameter of its connected components; see, e.g. [15, 25].

A *dominating set* of a graph $G = (V, E)$ is a set of vertices $D \subseteq V$ such that $N[D] = V$. That is, the union of the closed neighborhoods of vertices of D *spans* V. The Dominating Set (DS) problem, i.e., deciding whether a graph contains a dominating set of size at most k, is one of the fundamental NP-complete problems [33]. By now, both the classical and parameterized complexity of the problem are very well understood especially when restricted to sparse graph classes and even when considering various possible parameterizations [22, 20, 2, 3]. In particular, and relevant to this work, Dominating Set is fixed-parameter tractable (FPT) parameterized by the size of the solution in nowhere-dense classes of graphs [20].

In the rest of this section, we informally state our results and put them into context. In Section 2.1, we introduce the main problem from which we will derive our results and state the hardness result we obtain for it. In Section 2.2, we explain the consequences of this result for Dominating Set Reconfiguration (DSR) and in Section 2.3, we state our main positive results.

Token jumping vs. token sliding. The main focus of this paper is to study the parameterized complexity of reconfiguration problems restricted to sparse classes of graphs and in particular the case of Dominating Set Reconfiguration, or DSR for short. There are three different natural adjacency relations one can consider between dominating sets, which gave rise to

three different models in the literature; namely the token jumping (TJ) model, the token sliding (TS) model, and the token addition/removal (TAR) model. Since the TAR model is known to be polynomially equivalent to the TJ model [13], we only discuss the other two.

In the *token jumping* model, we say that two solutions S, S' are *adjacent* if both $S \setminus S'$ and $S' \setminus S$ have size at most one. In other words, we can remove a vertex of S and add a vertex of S' to transform S into S'. In the token view of the problem, we say a token *jumps* from some vertex u to another (possibly already occupied) vertex v. In the *token sliding* model, we additionally require that $uv \in E$, i.e., the vertices u and v must be adjacent, and the token is said to *slide* on the edge uv from u to v. We shall use TS-Π and TJ-Π to denote the reconfiguration variant of problem Π under the token sliding and token jumping model, respectively. For instance, TS-DSR corresponds to DSR in the token sliding model and TJ-DSR corresponds to DSR in the jumping model.

TJ-DSR is known to be PSPACE-complete on split graphs, bipartite graphs [13, 27, 32], and planar graphs of maximum degree 3 [29] and bounded-bandwidth graphs [42][1]. On the positive side, linear-time algorithms are known for trees, interval graphs, and cographs. As for TS-DSR, the problem is known to be PSPACE-complete on circle graphs [16], split graphs [13], bipartite graphs [13], and planar bounded-bandwidth graphs of maximum degree three [42]. Polynomial-time algorithms for TS-DSR are known for circular-arc graphs, dually chordal graphs, and cographs [28, 13, 16, 4].

Even though both TS-DSR and TJ-DSR are known to be PSPACE-complete restricted to instances of constant bandwidth/pathwidth/treewidth, an exact constant above which the problems become hard is not explicit in the hardness proofs of Wrochna [42]. Determining an explicit upper bound for which the problems are hard has been left open for almost a decade now, see e.g. [8, 16, 5].[2] Our first main result consists in giving an explicit upper bound on the treewidth and even the pathwidth above which the TS-DSR problem becomes PSPACE-complete. Our proof uses a completely different approach from the one of Wrochna [42] and is based on a new problem, namely Tape Reconfiguration (Tape-Rec), we introduce and prove to also be hard (formal definitions in Section 2.1).

▶ **Theorem 1.** *TS-DSR is PSPACE-complete even when restricted to graphs of treewidth (resp. pathwidth) at most* 12 *(resp.* 18*).*

The existence of explicit constants b and w for which (i) TS-DSR is PSPACE-complete in graphs of bandwidth b, and (ii) TJ-DSR is PSPACE-complete in graphs of pathwidth or treewidth at most w is also open. Unfortunately, our proof technique does not yield such explicit constants. Moreover, it is very unlikely that a small variation of our proof technique can provide such constants since this would almost automatically imply that the problems (parameterized by dominating set size k) are also XL-complete when restricted to the aforementioned graph classes, which is in contradiction with known results [18]. This follows, in part, from the fact that our reductions construct instances in which the dominating set size is negligible with respect to n.

We note that the constants 12 and 18 are probably not optimal. In particular, the complexity status of both TJ-DSR and TS-DSR in outerplanar graphs (which are graphs of treewidth at most 2) is still open. We will discuss in more detail the proof technique in

[1] Bandwidth is a very restricted subclass of pathwidth which, in turn, is a restriction of treewidth. That is $\text{bandwidth}(G) \geq \text{pathwidth}(G) \geq \text{treewidth}(G)$.

[2] The question is open for a wealth of reconfiguration problems including Independent Set Reconfiguration, Vertex Cover Reconfiguration, Shortest Path Reconfiguration, and Dominating Set Reconfiguration.

Section 3, but note that the proof technique of Theorem 1 is *drastically* different from the one of Wrochna [42] since the size of the dominating set is linear in the size of the graph in [42] while it is independent from $n = |V(G)|$ in our proof. This, in turn, allows us to derive the non-existence of parameterized algorithms which was completely out of reach with the techniques of [42].

Parameterized complexity and the mysterious case of token sliding. A systematic study of the parameterized complexity of reconfiguration problems was initiated by Mouawad et al. [38]. This was followed by a long sequence of results trying to push the tractability boundary of many reconfiguration problems (see [18] for a survey which mainly focuses on dominating set and independent set reconfiguration results).

On general graphs, TJ-DSR and TS-DSR are W[2]-complete parameterized by the dominating sets size k plus the length of a reconfiguration sequence ℓ [38, 10]. For the parameter k alone, it was shown by Bodlaender et al. [10] that both problems are XL-complete[3] if ℓ is not given, XNL-complete if ℓ is given in binary, and XNLP-complete if ℓ is given in unary as part of the input. The constructions of Bodlaender et al. [10] result in heavily dense graphs that contain all possible subgraphs H as subgraphs. In particular, it was left as an open problem whether the token sliding and token jumping variants of the problems behave the same on H-free graphs[4].

When parameterized by ℓ alone, both TJ-DSR and TS-DSR are FPT on any class of graphs where first-order model-checking is FPT, e.g., nowhere dense classes [26] and classes of bounded twinwidth [14] (assuming a contraction sequence is given as part of the input). Indeed, for the parameter ℓ, we can find a fixed sentence expressing the existence of a reconfiguration sequence of length ℓ.

Unfortunately, the approach via first-order model-checking does not help us deal with the parameterization by k since reconfiguration sequences might be arbitrarily long compared to k. It is not possible to state the existence of such a sequence in first-order logic with a bounded length sentence. Instead, the key tool to tackle TJ-DSR parameterized by k alone is based on the notion of domination cores [20]. In fact, the complexity of TJ-DSR parameterized by k is quite well understood for sparse classes of graphs. The problem is known to be FPT parameterized by k for biclique-free graphs and semi-ladder-free graphs [35, 24] (classes encompassing nowhere dense and degenerate graphs). In particular, it restricts quite a lot the type of graph on which TJ-DSR might be XL-complete.

On the other hand, the parameterized complexity of TS-DSR remains open even when restricted to very simple graph classes such as bounded pathwidth or treewidth graphs[5] or planar graphs. This difference of knowledge between the sliding and jumping variants is also observed for other problems such as Independent Set Reconfiguration (ISR) where the token jumping version is known to be FPT (parameterized by k) on $K_{\ell,\ell}$-free graphs while its sliding counterpart is only known to be FPT on planar graphs and is open beyond [18]. Bodlaender et al. [10] mentioned that "*it would be interesting to investigate*

[3] XL is a complexity class that contains the W-hierarchy and which naturally contains most of the reconfiguration problems. In particular, when a problem is XL-complete it is very unlikely that it is FPT.

[4] A graph G is *H-free* if it does not contain H as an *induced subgraph*. That is, there is no subset of vertices in G whose induced subgraph is isomorphic to H. A graph G is *H-minor-free* if it does not contain H as a *minor*. That is, H cannot be formed from G by deleting vertices and edges and by contracting edges.

[5] Note that the parameterized complexity status of TS-DSR on bounded bandwidth graphs is trivially FPT since the number of vertices is upper bounded by a function of the bandwidth and the domination number.

for which graph classes switching between token jumping and token sliding does affect the parameterized complexities". The second goal of this paper is to study this question via the lens of Dominating Set Reconfiguration.

In the token sliding case, apart from the few polynomial results on very restricted classes, no FPT algorithm parameterized by k is known. The existence of such an algorithm is conjectured in several papers including [18, 36, 8] or the recent survey [18]. The second main contribution of this article makes a step towards closing this gap by proving the following theorem:

▶ **Theorem 2.** *TS-DSR parameterized by k is XL-complete when restricted to graphs of treewidth at most* 12 *and pathwidth at most* 18*. Moreover, it remains XL-hard when parameterized by k plus the feedback vertex set number.*

A *feedback vertex* is a subset of vertices whose deletion leaves an acyclic graph. The *feedback vertex set number* is the minimum size of a feedback vertex set. The first hardness result of Theorem 2 (for pathwidth and treewidth) is again a consequence of the hardness of the Tape-Rec problem. The second hardness result is also based on the hardness of Tape-Rec but the proof is more technical since we need to maintain a small feedback vertex set number. One can wonder if we can strengthen the result of Theorem 2 by replacing feedback vertex set number by vertex cover number. The answer is negative. A *vertex cover* is a subset of vertices whose deletion leaves an independent set. Recall that given a simple undirected graph G, a set of vertices $I \subseteq V$ is an *independent set* if the vertices of I are pairwise non-adjacent. The *vertex cover number* is the minimum size of a vertex cover. We note that TS-DSR parameterized by k plus the vertex cover number is trivially FPT. This follows from the fact that given a vertex cover C, we can partition $R = V \setminus C$ into classes such that two vertices belong to the same class if and only if they have the same neighbors in C. All classes consist of independent sets (since C is a vertex cover) and any class containing more than k vertices can be reduced since no more than k vertices can be used from a class (and all vertices of a class are "equivalent"). Hence, we obtain an instance with at most $|C| + 2^{|C|}k$ vertices (assuming no isolated vertices).

Theorem 2 answers, as mentioned before, a problem that has been left open in several previous articles [18, 36, 8, 18]. The result ensures that, even in very restricted sparse graph classes, TS-DSR and TJ-DSR behave completely differently. To our knowledge, it is the first time the token sliding and token jumping models behave differently for a reconfiguration problem on a class of bounded treewidth (and even nowhere dense)[6]. Actually, it is, as far as we know, the first reconfiguration problem that is not FPT on bounded treewidth graphs. It also ensures that TS-DSR and TS-ISR behave very differently on nowhere dense classes of graphs (since an easy applications of the lemmas of [8] ensures that TS-DSR is FPT parameterized by k plus the feedback vertex set number). Moreover, our results provide the first (and infinite) collection of graphs (any supergraph of a large enough biclique) for which TS-DSR is XL-complete parameterized by k while TJ-DSR is FPT, which partially answers the question of [10].

To sum up, our results make progress in three directions by showing that:

1. TS and TJ do not necessarily behave the same on graphs of bounded treewidth (and below);
2. ISR and DSR do not behave the same on nowhere dense graph classes; and
3. TJ-DSR and TS-DSR behave differently on infinitely many H-free graphs.

[6] For sparse graphs, such a difference has already been observed for bounded degeneracy graphs for instance [18]

We complement our negative results by positive results proving that TS-DSR is FPT on planar graphs (and actually beyond), answering a problem left open in [18].

▶ **Theorem 3.** *TS-DSR parameterized by k is FPT on planar graphs.*

To obtain the positive result, we adapt the strategy used in the token jumping model via domination cores to handle the case where the size of a forbidden minor is not too large (namely $K_{4,\ell}$-minor-free graphs). However, while the proof is almost direct in the token jumping variant whenever domination cores exist, the proof gets more technical here as connectivity matters. Note that this method cannot be generalized much further since we prove that TS-DSR is XL-complete for graphs of treewidth at most 12, and when parameterized by k plus feedback vertex set number. However, it would be interesting to understand the limit between tractability and intractability on H-minor free graphs.

Reconfiguration of connected dominating sets. Our proof techniques are actually strong enough to be generalized further to the Connected Dominating Set Reconfiguration problem (CDSR). Recall that a dominating set D is a *connected dominating set* if the graph induced by D is connected. Deciding whether a given graph contains a connected dominating set of size at most k is known to be NP-complete [33] and W[2]-complete parameterized by k [21].

TS-CDSR and TJ-CDSR, the corresponding reconfiguration variants, are known to be PSPACE-complete (even on graphs of bounded pathwidth) and W[1]-hard parameterized by $k + \ell$ [36, 10]. Moreover, TJ-CDSR parameterized by $k + \ell$ remains W[1]-hard even when restricted to 5-degenerate graphs. On the positive side, TJ-CDSR parameterized by k is known to be FPT on planar graphs [36].

TJ-CDSR parameterized by k has been conjectured to be FPT on every nowhere dense class of graphs in several papers, see e.g. [18, 36]. In [36], the authors ask whether TJ-CDSR and TS-CDSR parameterized by k are FPT on graphs of bounded pathwidth. We provide a negative answer to all of the aforementioned questions. Our proof techniques can also be adapted for other dominating set variants such as distance d dominating sets and total dominating sets which have been studied in the literature.

Related work. Deciding whether a given graph contains an independent set of size at least k is known to be NP-complete [33] and W[1]-complete parameterized by (solution size) k [21].

The Independent Set Reconfiguration (ISR) problem is another central problem that has been very widely studied under the combinatorial reconfiguration framework. Both TJ-ISR and TS-ISR are known to be PSPACE-complete restricted to planar graphs of bounded bandwidth [42, 41] and XL-complete when parameterized by the size of independent sets k [10].

Similarly to TJ-DSR, the complexity of the token jumping variant, i.e., TJ-ISR, is rather well understood. Indeed, Ito et al. [31] proved that TJ-ISR parameterized by k is FPT on planar graphs. This result has been generalized to nowhere dense and degenerate classes in [35, 1] and even to $K_{d,d}$-free graphs in [17]. The idea of these proofs is to follow a two-step strategy. First, an "important" subset X of vertices of bounded size is identified. Second, the remaining vertices are classified according to their neighborhood in X, and it is shown that if some class is too large then it can be reduced.

Again, the situation becomes a lot less clear when we consider the token sliding model, i.e., TS-ISR. For instance, while TJ-ISR is trivial on chordal graphs [32], TS-ISR is PSPACE-complete on split graphs [9]. Lokshtanov and Mouawad [34] showed that, in bipartite graphs, TJ-ISR is NP-complete while TS-ISR remains PSPACE-complete.

Just like DSR, even after a decade of research, very little is known about the parameterized complexity of ISR under the token sliding model. It was proved in [6] that TS-ISR is FPT on bipartite C_4-free graphs. The result was later generalized by Bartier et al. [8] on planar graphs and graphs of bounded maximum degree and graphs of girth at least 5 [7]. One can easily use the reduction rules introduced in [8] in order to prove that TS-ISR is FPT parameterized by k plus the feedback vertex set number of the graph, which shows that the behavior of TS-DSR and TS-ISR is very different on sparse graph classes.

2 Technical overview of our results

2.1 Tape Reconfiguration

Let Σ be an alphabet. The elements of Σ are called *letters*. A Σ-*tape* (or tape for short when Σ is clear from context) is a graph where each vertex, called a *cell*, is labeled with a subset of Σ, called its *content*. Moreover, each tape comes with two distinguished vertices, called the *start* and *end* cell, respectively. We will move a token, called *(read) head*, on each tape.

Let $\mathcal{T} = \{T_1, \ldots, T_p\}$ be a set of Σ-tapes. For $i \in [1, p]$,[7] let c_i be a cell of T_i, and $start_i$ (resp. end_i) be the start (resp. end) cell of T_i. We say that $(c_1, \ldots, c_p)$ forms a *valid configuration* if the union of their contents is Σ. Let $C = (c_1, \ldots, c_p)$ and $C' = (c'_1, \ldots, c'_p)$ be two valid configurations. We say that there is an *elementary transformation* (or a *(reconfiguration) step*) between C and C' if they differ on exactly one element, say the j-th, and $c_j c'_j$ is an edge in T_j.

Tape Reconfiguration (Tape-Rec)
Input: An alphabet Σ and a set $\mathcal{T}$ of Σ-tapes, an initial and a final configuration C_s, C_t.
Parameter: Size of the alphabet Σ plus the number of Σ-tapes.
Output: Yes, if it is possible to transform the start configuration (i.e., $(start_1, \ldots, start_p)$) into the end configuration (i.e., $(end_1, \ldots, end_p)$) via a sequence of elementary transformations while keeping a valid configuration all along.

We argue in the full version of the paper that one can always assume the number of Σ-tapes is bounded by the size of the alphabet $|\Sigma|$. Consequently, one can assume $|\Sigma|$ to be the parameter. The problem can be equivalently defined as follows. We are given a set of p graphs, each with a read head positioned on one vertex. In one step, we are permitted to slide a read head along an edge within any one of these graphs. Note that since we only allow the movement of tokens along edges we can assume that these graphs are connected. We will mostly show hardness results for this problem, and a natural question is about how the structure of the input graphs affects its complexity. As it turns out, even the simplest case – captured by the variant Path-Tape-Rec, which requires all tapes to be paths whose endpoints are the start and end cells – is already hard. In particular, we show the following:

▶ **Theorem 4.** *Tape-Rec is XL-complete.*

2.2 Consequences for TS-DSR (and TS-CDSR)

We prove that, for DSR, the results for TJ have not been generalized to TS for a good reason; TS-DSR is hard even on very restricted graph classes. Namely, we show the following:

[7] $[1, p]$ denotes the set of integers between 1 and p.

▶ **Theorem 5.** *TS-DSR and TS-CDSR are PSPACE-complete and XL-complete parameterized by k plus the feedback vertex set number, even restricted to 7-degenerate graphs of treewidth at most 12 and pathwidth at most 18.*

Note that, as far as we know, this is the first "natural" reconfiguration problem which becomes hard parameterized by k plus the feedback vertex set number of the graph. Very few problems are hard for that parameter since allowing the feedback vertex set number to be part of the parameter enables the use of very powerful algorithmic techniques (given the simple structure of graphs having a feedback vertex set of bounded size).

With minor modifications to the reduction, we obtain the following slightly weaker result for TJ-CDSR:

▶ **Theorem 6.** *TJ-CDSR is PSPACE-complete and XL-complete when parameterized by k plus the feedback vertex set number, even restricted to 5-degenerate graphs of treewidth at most 13 and pathwidth at most 19.*

Theorems 5 and 6 imply that both TS-DSR and TJ-CDSR are very hard problems, even though the connectivity constraints are quite different in both; the latter is a global condition on the dominating set while the former only wants to ensure local connectivity when moving along edges. In addition, both problems are much harder than TJ-DSR which is known to be FPT parameterized by k on nowhere dense, degenerate, biclique-free, and semi-ladder-free graphs.

2.3 Positive results for TS-DSR

We complement our hardness results with positive ones. In particular, we show that TS-DSR is FPT on planar graphs. We first prove that the following holds:

▶ **Theorem 7.** *TS-DSR parameterized by k is FPT on $K_{3,d}$-free graphs.*

As a by-product of Theorem 7, TS-DSR (parameterized by k) is FPT on planar graphs. The proof technique is inspired from [31, 35] and consists in looking at neighborhood classes in a domination core and proving that all these classes can be reduced to have bounded size. It was proven in [23] that a domination core of size $\mathcal{O}(k^{d+1})$ exists (and can be efficiently computed) even in $K_{d,d}$-free graphs (having a dominating set of size k).

▶ **Theorem 8.** *TS-DSR parameterized by k is FPT on $K_{4,d}$-minor-free graphs.*

To prove Theorem 8, we start as in the proof of Theorem 7. Although it can be seen as rather incremental compared to the previous result, reducing neighborhood classes is actually much trickier and requires more general arguments. In some steps of the proof we have to use the fact that dominating the domination core ensures that the whole graph is dominated to show that some subsets of vertices have to intersect with the dominating set (which in turn permits reducing some large enough parts of the graph).

3 Hardness results roadmap

3.1 Synchronized version and width of tape reconfiguration problems

The goal of this section is to state our main hardness results and give outlines of their proofs. In order to do so, we will need to define some auxiliary problems along the way, which we also believe to be of independent interest.

The first auxiliary problems we will need are synchronized versions of Tape-Rec and Path-Tape-Rec where we assume that all the read heads move at the same speed. We say that a tape is r-*numbered* if each cell is labeled with some integer of $[1, r]$ (the same integer possibly occurring on several cells), where the number of adjacent cells differs by at most 1 modulo r. A configuration is numbered with j if all the read heads are on a cell numbered with j. The read heads are *synchronized* if, for every pair of tapes, the numbers of the cells under their read heads differ by at most 1 modulo r. In other words, if the read head in the tape T is at a cell numbered q, then the read head in T' is on a cell numbered with q or $q \pm 1$ mod r. A transformation is synchronized if the read heads are synchronized at each step. A configuration of a synchronized instance is *valid* if read heads on the synchronized tapes are synchronized and the union of the contents of the reading heads is the whole alphabet.

The synchronized version of Tape-Rec is defined as follows:

Sync-Tape-Rec
Input: An alphabet Σ, a set $\mathcal{T}$ of r-numbered tapes and two numbered configurations C_s, C_t.
Parameter: $|\Sigma| + |\mathcal{T}|$.
Output: Yes if and only if there exists a synchronized transformation from C_s to C_t.

We similarly define the restriction Path-Tape-Rec when tapes are paths whose endpoints are the start and end cells, and its synchronized version Sync-Path-Tape-Rec where, for technical reasons, we assume that the start cells are all numbered with 1 and the numbering is non-decreasing along each tape.

Observe that the synchronization makes the problem much simpler to solve in some configurations. For instance, consider instances of Path-Tape-Rec where cells of paths are labeled with their position in the path. Then, one can easily upper bound the number of valid configurations by $2^{|\mathcal{T}|} \cdot n$ valid configurations (where n is the size of the longest tape). In particular, the following holds:

▶ Remark 9. In the particular setting where cells of paths are labeled with their position in the path, Sync-Path-Tape-Rec is FPT.

Note that we do not know if Sync-Path-Tape-Rec lies in P in that setting.

A more general problem. We will consider a harder problem whose flavor is close to Path-Tape-Rec, denoted by Multi-Tape-Rec, and prove that this problem is W[∗]-hard. The idea consists in allowing to make choices to construct a valid instance of Path-Tape-Rec by selecting tapes among some tuples. More formally, we define the problem Multi-Tape-Rec as follows:

Multi-Tape-Rec
Input: An alphabet Σ, k tuples $\mathcal{T}^1, \dots, \mathcal{T}^k$ of path Σ-tapes.
Parameter: $k + |\Sigma|$
Output: Yes if and only if there exists $i_1, \dots, i_k$ such that $\{\mathcal{T}^j_{i_j} \mid j \in [1, k]\}$ is a positive instance of Path-Tape-Rec.

There is a trivial reduction from Path-Tape-Rec to Multi-Tape-Rec; by considering an instance of Multi-Tape-Rec where each tuple has size 1. More formally, $\{T_1, \dots, T_k\}$ is a positive instance of Path-Tape-Rec if and only if the k tuples $(T_1), \dots, (T_k)$ of size 1 form a positive instance of Multi-Tape-Rec. In other words, we have no choice on the tape to choose in each tuple so we simply have an instance of Path-Tape-Rec. We will later provide

a reduction in the converse direction, proving that Path-Tape-Rec and Multi-Tape-Rec have the same complexity. We define Sync-Multi-Tape-Rec as the synchronized version of Multi-Tape-Rec as we defined Sync-Tape-Rec from Tape-Rec earlier.

Before stating our main results, let us first prove that Sync-Multi-Tape-Rec is hard. We obtain much stronger hardness results later but this proof is interesting since it illustrates the utility of having synchronized tapes to design hardness reductions with the following very simple statement.

▶ **Theorem 10.** *Sync-Multi-Tape-Rec is W[2]-hard even when* $|\Sigma| = 1$.

3.2 Hardness results for tape reconfiguration

Width of tape reconfiguration instances. In the rest of this section we will need some notions of width of a tape reconfiguration instance. Let $I = (\Sigma, (T_i)_i, C_s, C_t)$ be an instance of Tape-Rec (or any other tape problem defined above such as Multi-Tape-Rec or Sync-Tape-Rec), where Σ is the alphabet and $(T_i)_i$ is a collection of tapes. The *extended graph* of I is the graph with vertex set $\bigcup_i V(T_i) \cup \Sigma$, and where xy is an edge if either it is an edge of some T_i, or x is a cell of some T_i which contains the letter y.

We say that I is *d-degenerate* whenever its extended graph is d-degenerate. We similarly define the notion of pathwidth, treewidth, and minors of an instance I; the *pathwidth (resp. treewidth)* of I is the pathwidth (resp. treewidth) of its extended graph.

We say that a path (resp. tree) decomposition of the extended graph of I is *s-structured* if each of its bags contains vertices of at most s tapes. The utility of structured decompositions will appear naturally in future reductions. The minimum width of a s-structured path (resp. tree) decomposition is called the *s-structured pathwidth (resp. treewidth)* of I.

Hardness results. The goal of this section is to give the main hardness results for Tape-Rec and its variants. The first and one of our main results is the following:

▶ **Theorem 11.** *Tape-Rec is PSPACE-complete and XL-complete even restricted to* 5*-degenerate instances of* 2*-structured treewidth at most* 9 *and* 3*-structured pathwidth at most* 14*.*

The proof of Theorem 11 is divided into two steps. The first one consists in proving that the synchronized version of Tape-Rec, Sync-Tape-Rec, is PSPACE-complete and XL-complete. The reduction is from TJ-DSR. The proof generalizes the proof of Theorem 10. Indeed, Theorem 10 shows first how to encode a choice of vertices by a choice of tapes, and then translates checking that these vertices dominate the graph into moving read heads on these tapes. We reuse these "checking paths" to prove Theorem 11, but now we have to encode a way to reconfigure dominating sets. This can be done by arranging the checking paths between some cells that correspond to vertices of G. By doing so, our tapes are not paths anymore; they become subdivided stars. Moreover, in order to guarantee that we reconfigure dominating sets one vertex at a time, we also need to add another tape. This construction leads to the following statement which we prove in the full version of the paper:

▶ **Theorem 12.** *Sync-Tape-Rec is PSPACE-complete and XL-complete parameterized by* k *plus the feedback vertex set number, even restricted to* 3*-degenerate instances of* 1*-structured treewidth at most* 3*,* 2*-structured pathwidth at most* 5*, and whose tapes are subdivided stars.*

The second step in the proof of Theorem 11 consists in providing a very generic reduction from synchronized problems to their unsynchronized versions. This reduction allows us to prove that synchronized and unsynchronized versions of each problem are actually equivalent (up to small changes in the parameters and the structured width).

To prove this, we add a new tape and new characters to Σ that force the read heads to move at the same "speed" in all the tapes. The reduction also ensures that all the cells are useful to cover the whole alphabet Σ. Namely, we say that an instance $(\Sigma, \mathcal{T}, C_s, C_t)$ of Tape-Rec is *irreducible* if Σ cannot be written as the union of the contents of less than $|\mathcal{T}|$ cells.

Formally, we show that the following holds, which directly implies Theorem 11, when combined with Theorem 12.

▶ **Theorem 13.** *There is an FPT-reduction (and PL-reduction) Φ from instances of Sync-Tape-Rec to irreducible instances of Tape-Rec. Moreover, if I is an instance of Sync-Tape-Rec of s-structured treewidth (resp. pathwidth) w, then $\Phi(I)$ has $(s+1)$-structured treewidth (resp. pathwidth) at most $w + 3s + 3$. Furthermore, Φ increases the degeneracy by at most 2 and the feedback vertex set number by at most $3|\Sigma| + 3$.*

Theorem 11 is a direct corollary of Theorems 12 and 13. Theorem 13 is proved in the full version of the paper. We give two proofs of Theorem 13; one where one additional tape is a long path and one where it is a triangle. The spirit of the two proofs is similar but each construction allows us to reach a specific conclusion. For long paths, we guarantee that all tapes are paths and thus obtain hardness results for Path-Tape-Rec. However this path reduction comes at a cost; the size of clique-minors and treewidth increase by a function of the size of the alphabet, hence they become unbounded. We thus provide another reduction where the additional tape is a triangle that permits to control widths and degeneracy. In the end, the first reduction yields hardness results for TS-DSR parameterized by feedback vertex set number, while the second handles the bounded widths and degeneracy cases.

3.3 Consequences for (C)DSR

Let us finally explain the consequences of the results from the previous subsections on TS-DSR, TJ-CDSR, and TS-CDSR.

▶ **Theorem 5.** *TS-DSR and TS-CDSR are PSPACE-complete and XL-complete parameterized by k plus the feedback vertex set number, even restricted to 7-degenerate graphs of treewidth at most 12 and pathwidth at most 18.*

The proof of Theorem 5 follows from the next lemma:

▶ **Lemma 14.** *There is an FPT-reduction (and PL-reduction) from Tape-Rec on irreducible d-degenerate instances of s-structured treewidth (resp. pathwidth) at most w with k tapes and of feedback vertex set number f to TS-DSR and TS-CDSR with $k+1$ tokens on graphs:*

- *of treewidth (resp. pathwidth) at most $s + w + 1$ and degeneracy at most $d + 2$, or*
- *of feedback vertex set number $f + k + 1$.*

By Theorem 11, Tape-Rec is PSPACE-complete and XL-complete on 5-degenerate instances of 2-structures treewidth at most 9 and 3-structured pathwidth at most 14. Moreover, by Theorem 13, this remains true when the instances are additionally irreducible. Therefore, applying Lemma 14, we can conclude the proof of Theorem 5.

Note that considering irreducible instances in the above proof allowed us to increase the widths by at most $w + 1$. Without this assumption, a similar reduction works, but we need to introduce a twin of each x_i. This yields a reduction to possibly non-irreducible instances of Tape-Rec with slightly worse bounds (adding $2w + 1$ to the widths instead).

The reduction can also be modified to get hardness results for the parameterized (and classical) complexity of TJ-CDSR, which solves open problems from the literature. Namely, we adapt our reduction to show that TJ-CDSR is PSPACE-complete and XL-complete.

▶ **Theorem 6.** *TJ-CDSR is PSPACE-complete and XL-complete when parameterized by k plus the feedback vertex set number, even restricted to* 5*-degenerate graphs of treewidth at most* 13 *and pathwidth at most* 19*.*

3.4 Hardness of tape reconfiguration on paths

Tape-Rec is XL-complete even restricted to instances where tapes are trees (and even subdivided stars). One can then naturally wonder if the same holds when tapes are simpler graphs, for example paths. We did not succeed to obtain exactly the same hardness result but nevertheless we prove that Path-Tape-Rec is W[$*$]-hard. The W[2]-hardness is a consequence of the synchronization result used to prove Theorem 13. This result is actually quite generic and can be applied to all the problems defined before. In particular, we will get the following for free from Theorem 10:

▶ Remark 15. Multi-Tape-Rec is W[2]-hard.

Recall that there is a trivial reduction from Path-Tape-Rec to Multi-Tape-Rec. Using another idea, called the *selector gadgets*, we can provide a reduction in the converse direction, proving that Path-Tape-Rec and Multi-Tape-Rec have the same complexity. More formally, we will prove that the following holds:

▶ **Theorem 16.** *Multi-Tape-Rec and Path-Tape-Rec are equivalent under FPT-reductions.*

As a byproduct, Theorems 10 and 16 ensure that Path-Tape-Rec is W[2]-hard. We will actually prove the following more general result (as sketched above):

▶ **Theorem 17.** *Multi-Tape-Rec (and consequently Path-Tape-Rec) is W[$*$]-hard.*

To prove Theorem 17, we will generalize the ideas of Theorem 10. In the proof of Theorem 10, we gave a reduction from Dominating Set, which is a problem in W[2]. In other words, it can be expressed with a logical sentence, whose variables correspond to the vertices chosen in the dominating set, with the following shape:

$$\bigwedge_{u \in V} \bigvee_{v \in N[u]} x_v,$$

that is a conjunction of disjunctions of literals. The reduction of Theorem 10 allowed us to make a selection (by the definition of Multi-Tape-Rec) and then check the conjunction of conditions (one condition per cell), each of them corresponding to a disjunction (at least one cell has to contain ✓).

The W-hierarchy can be generalized to higher levels; W[h] problems can be expressed in the same way except that up to h nested boolean operators are allowed. Therefore, in order to prove that Multi-Tape-Rec is W[h]-hard, we build on this W[2]-hardness result and design OR gadgets and AND gadgets allowing us to lift W[$h+1$]-hardness from W[h]-hardness. Note that this has to be done without adding too many characters nor additional tapes. The details of the proof can be found in the full version of the paper.

We were not able to determine if the problem is XL-complete or not and leave as an open problem the exact complexity of Path-Tape-Rec.

References

1 Akanksha Agrawal, Soumita Hait, and Amer E. Mouawad. On finding short reconfiguration sequences between independent sets. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 39:1–39:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.39.

2 Jochen Alber, Hans L. Bodlaender, Henning Fernau, and Rolf Niedermeier. Fixed parameter algorithms for PLANAR DOMINATING SET and related problems. In Magnús M. Halldórsson, editor, *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, volume 1851 of *Lecture Notes in Computer Science*, pages 97–110. Springer, 2000. doi:10.1007/3-540-44985-X_10.

3 Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009. doi:10.1007/S00453-008-9204-0.

4 Niranka Banerjee and Duc A. Hoang. The complexity of distance-r dominating set reconfiguration. *CoRR*, abs/2310.00241, 2023. doi:10.48550/arXiv.2310.00241.

5 Valentin Bartier. *Combinatorial and Algorithmic aspects of Reconfiguration*. PhD thesis, Université Grenoble Alpes, 2021.

6 Valentin Bartier, Nicolas Bousquet, Clément Dallard, Kyle Lomer, and Amer E. Mouawad. On girth and the parameterized complexity of token sliding and token jumping. *Algorithmica*, 83(9):2914–2951, 2021. doi:10.1007/S00453-021-00848-1.

7 Valentin Bartier, Nicolas Bousquet, Jihad Hanna, Amer E. Mouawad, and Sebastian Siebertz. Token sliding on graphs of girth five. *Algorithmica*, 86(2):638–655, 2024. doi:10.1007/S00453-023-01181-5.

8 Valentin Bartier, Nicolas Bousquet, and Amer E. Mouawad. Galactic token sliding. *J. Comput. Syst. Sci.*, 136:220–248, 2023. doi:10.1016/J.JCSS.2023.03.008.

9 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory Comput. Syst.*, 65(4):662–686, 2021. doi:10.1007/S00224-020-09967-8.

10 Hans L. Bodlaender, Carla Groenland, and Céline M. F. Swennenhuis. Parameterized complexities of dominating and independent set reconfiguration. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 9:1–9:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.IPEC.2021.9.

11 Marthe Bonamy, Nicolas Bousquet, Carl Feghali, and Matthew Johnson. On a conjecture of Mohar concerning Kempe equivalence of regular graphs. *Journal of Combinatorial Theory, Series B*, 135:179–199, 2019. doi:10.1016/J.JCTB.2018.08.002.

12 Marthe Bonamy, Nicolas Bousquet, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Arnaud Mary, Moritz Mühlenthaler, and Kunihiro Wasa. The perfect matching reconfiguration problem. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, pages 80:1–80:14, 2019. doi:10.4230/LIPIcs.MFCS.2019.80.

13 Marthe Bonamy, Paul Dorbec, and Paul Ouvrard. Dominating sets reconfiguration under token sliding. *Discret. Appl. Math.*, 301:6–18, 2021. doi:10.1016/J.DAM.2021.05.014.

14 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

15 Nicolas Bousquet and Marc Heinrich. A polynomial version of cereceda's conjecture. *J. Comb. Theory B*, 155:1–16, 2022. doi:10.1016/J.JCTB.2022.01.006.

16 Nicolas Bousquet and Alice Joffard. TS-reconfiguration of dominating sets in circle and circular-arc graphs. In Evripidis Bampis and Aris Pagourtzis, editors, *Fundamentals of Computation Theory International Symposium, FCT 2021, Proceedings*, volume 12867 of *Lecture Notes in Computer Science*, pages 114–134. Springer, 2021. doi:10.1007/978-3-030-86593-1_8.

17 Nicolas Bousquet, Arnaud Mary, and Aline Parreau. Token jumping in minor-closed classes. In *Fundamentals of Computation Theory - 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, pages 136–149, 2017. doi:10.1007/978-3-662-55751-8_12.

18 Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of reconfiguration problems. *Comput. Sci. Rev.*, 53:100663, 2024. doi:10.1016/J.COSREV.2024.100663.

19 L. Cereceda, J. van den Heuvel, and M. Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011. doi:10.1002/jgt.20514.

20 Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In Ravi Kannan and K. Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPIcs*, pages 157–168. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2009. doi:10.4230/LIPICS.FSTTCS.2009.2315.

21 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

22 Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.STACS.2016.31.

23 Eduard Eiben, Mithilesh Kumar, Amer E Mouawad, Fahad Panolan, and Sebastian Siebertz. Lossy kernels for connected dominating set on sparse graphs. *SIAM Journal on Discrete Mathematics*, 33(3):1743–1771, 2019. doi:10.1137/18M1172508.

24 Grzegorz Fabianski, Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. Progressive algorithms for domination and independence. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 27:1–27:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.27.

25 Parikshit Gopalan, Phokion G Kolaitis, Elitza Maneva, and Christos H Papadimitriou. The connectivity of boolean satisfiability: computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009. doi:10.1137/07070440X.

26 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

27 Arash Haddadan, Takehiro Ito, Amer E Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theoretical Computer Science*, 651:37–49, 2016. doi:10.1016/J.TCS.2016.08.016.

28 Arash Haddadan, Takehiro Ito, Amer E. Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theor. Comput. Sci.*, 651:37–49, 2016. doi:10.1016/J.TCS.2016.08.016.

29 R. A. Hearn and E. Demaine. PSPACE-completeness of Sliding-block Puzzles and Other Problems Through the Nondeterministic Constraint Logic Model of Computation. *Theor. Comput. Sci.*, 343(1-2), October 2005. doi:10.1016/J.TCS.2005.05.008.

30 Takehiro Ito, Erik D Demaine, Nicholas JA Harvey, Christos H Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011. doi:10.1016/J.TCS.2010.12.005.

31 Takehiro Ito, Marcin Jakub Kaminski, and Hirotaka Ono. Fixed-parameter tractability of token jumping on planar graphs. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2014. doi:10.1007/978-3-319-13075-0_17.

32 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical computer science*, 439:9–15, 2012. doi:10.1016/J.TCS.2012.03.004.

33 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

34 Daniel Lokshtanov and Amer E Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Transactions on Algorithms (TALG)*, 15(1):1–19, 2018. doi:10.1145/3280825.

35 Daniel Lokshtanov, Amer E. Mouawad, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. *J. Comput. Syst. Sci.*, 95:122–131, 2018. doi:10.1016/J.JCSS.2018.02.004.

36 Daniel Lokshtanov, Amer E. Mouawad, Fahad Panolan, and Sebastian Siebertz. On the parameterized complexity of reconfiguration of connected dominating sets. *Algorithmica*, 84(2):482–509, 2022. doi:10.1007/S00453-021-00909-5.

37 Amer E. Mouawad, Naomi Nishimura, Vinayak Pathak, and Venkatesh Raman. Shortest reconfiguration paths in the solution space of boolean formulas. *SIAM J. Discret. Math.*, 31(3):2185–2200, 2017. doi:10.1137/16M1065288.

38 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017. doi:10.1007/S00453-016-0159-2.

39 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.

40 Jan van den Heuvel. The complexity of change. *Surveys in combinatorics*, 409(2013):127–160, 2013. doi:10.1017/CBO9781139506748.005.

41 Tom C. van der Zanden. Parameterized complexity of graph constraint logic. In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 282–293. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.IPEC.2015.282.

42 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *J. Comput. Syst. Sci.*, 93:1–10, 2018. doi:10.1016/J.JCSS.2017.11.003.

Reconstructing Random Graphs from Distance Queries

Michael Krivelevich
School of Mathematical Sciences, Tel Aviv University, Israel

Maksim Zhukovskii
School of Computer Science, The University of Sheffield, UK

Abstract

We estimate the minimum number of distance queries that is sufficient to reconstruct the binomial random graph $G(n,p)$ with constant diameter with high probability. We get a tight (up to a constant factor) answer for all $p > n^{-1+o(1)}$ outside "threshold windows" around $n^{-k/(k+1)+o(1)}$, $k \in \mathbb{Z}_{>0}$: with high probability the query complexity equals $\Theta(n^{4-d}p^{2-d})$, where d is the diameter of the random graph. This demonstrates the following non-monotone behaviour: the query complexity jumps down at moments when the diameter gets larger; yet, between these moments the query complexity grows. We also show that there exists a non-adaptive algorithm that reconstructs the random graph with $O(n^{4-d}p^{2-d}\ln n)$ distance queries with high probability, and this is best possible.

2012 ACM Subject Classification Mathematics of computing → Random graphs; Mathematics of computing → Graph algorithms

Keywords and phrases random graphs, graph reconstruction, distance queries, query complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.30

Related Version *Full Version*: https://arxiv.org/pdf/2404.18318

Funding *Michael Krivelevich*: Research supported in part by NSF-BSF grant 2023688.

1 Introduction

Reconstruction of graphs was thoroughly studied in many different contexts and has various applications, e.g., in DNA sequencing [2, 13, 34], phylogenetics [1, 23, 29], and recovering neural networks [39]. A vast amount of literature is devoted to the average-case problem, i.e. to the reconstruction of random graphs [8, 12, 19, 21, 28, 31, 33, 35, 41]. In particular, the conjecture of Kelly and Ulam [24, 42] which is considered as one of the main reconstruction challenges is known to be true in binomial random graphs whp[1] [8, 28, 35].

Let G be a connected graph on $[n] := \{1,\dots,n\}$, and let $\mathcal{Q} \subset \binom{[n]}{2}$ be a set of pairs of vertices. In the usual way, we denote by $d_G(x,y)$ the graph distance between x and y in G. Let us say that $\mathcal{Q}$ *reconstructs* G, if G is the only graph on $[n]$ with distances $d_G(x,y)$ between $\{x,y\} \in \mathcal{Q}$, i.e. for every graph G' on $[n]$ such that $d_{G'}(x,y) = d_G(x,y)$ for all $\{x,y\} \in \mathcal{Q}$, we have $G' = G$. We will also call the graph $\mathcal{Q}$-*reconstructible* in this case.

In this paper, we study the model of reconstruction with an access to distance query oracle introduced in [5], although its variants were considered long before that for modelling reconstruction of a phylogenetic tree [18, 25, 43]. For every queried pair of vertices, the oracle answers the distance between these vertices in the hidden unknown graph G. For a queried pair of vertices $\{x,y\}$, we denote the oracle's answer to this query by $d(x,y)$, omitting the unknown graph in the subscript. An adaptive algorithm, at every step, selects the next query (a pair of vertices) based on the responses from the oracle to earlier queries.

[1] With high probability, that is, with probability tending to 1 as $n \to \infty$.


© Michael Krivelevich and Maksim Zhukovskii;
licensed under Creative Commons License CC-BY 4.0



Let $q \in \mathbb{N}$. If there exists an adaptive algorithm $\mathtt{A}$ such that, for the hidden input graph G on $[n]$, it queries at most q pairs of vertices comprising a set $\mathcal{Q} \subset \binom{[n]}{2}$ that reconstructs G, then we call G *q-reconstructible by* $\mathtt{A}$. Let us call the minimum q such that G is q-reconstructible by some algorithm *the (distance) query complexity of* G. It is easy to see that, in the worst case, the query complexity equals $\binom{n}{2}$: all pairs should be queried in order to reconstruct the graph. For example, this is the case when G is a clique. Reyzin and Srivastava [37] showed that, even for some trees, $\Omega(n^2)$ pairs are required to query. On the other hand, Mathieu and Zhou [31] presented an algorithm such that whp a uniformly random d-regular graph on $[n]$ is $(\lfloor n \ln^2 n \rfloor + dn/2)$-reconstructible by this algorithm, and wondered whether methods similar to those developed in their paper are applicable to sparse binomial random graphs $G(n,p)$. In this paper, in contrast, we study the distance query complexity of relatively dense random graphs $G(n,p)$ assuming $p > n^{-1+\varepsilon}$ for some $\varepsilon > 0$, i.e., when the random graph has bounded diameter whp. Although our main motivation lies in attempting to achieve tight bounds for the average-case complexity in a situation where the general setup is hard to analyse, additional interest in this problem is sparked by a surprising phenomenon, which will be discussed in the next two paragraphs.

One can expect that dense graphs are typically harder to recover through distance queries, as the influence of a single edge to be recovered on distances between vertices might be less pronounced and thus harder to detect. In order to get some intuition of how the query complexity of the random graph evolves, let us first assume that p is significantly above the threshold for the property of having diameter at most 2, namely, $p^2 n - 2\ln n - \ln\ln n \to \infty$ as $n \to \infty$. In this case, whp every pair of non-adjacent vertices has at least two common neighbours. Indeed, probability that a fixed pair of vertices has at most one common neighbour equals

$$\mathbb{P}(\mathrm{Bin}(n-2, p^2) \le 1) = (1-p^2)^{n-2} + (n-2)p^2(1-p^2)^{n-3} = O\left(np^2 e^{-np^2}\right) = o(n^{-2}).$$

Therefore, by the union bound, every non-adjacent pair of vertices of $G(n,p)$ has at least two common neighbours whp. Then whp the query complexity equals $\binom{n}{2}$. Indeed, for any set $\mathcal{Q}$ of pairs of vertices of size less than $\binom{n}{2}$, changing any single adjacency relation of a pair from $\binom{[n]}{2} \setminus \mathcal{Q}$ does not influence the distance between x and y for any $\{x,y\} \in \mathcal{Q}$. Therefore, any set $\mathcal{Q}$ that does not contain all pairs does not reconstruct $G(n,p)$ whp.

On the other hand, it seems likely that for p close to the connectivity threshold $\ln n/n$, the query complexity might be much less – quasilinear, as in the case of random regular graphs. Note that, in general the query complexity does not decrease as the graph becomes sparser since, in particular, as we mention above, for certain trees, the query complexity equals $\Omega(n^2)$. Nevertheless, we show that, on average, as density decreases, the query complexity jumps down at specific moments around hitting times of diameter's increments. However, between these moments the query complexity grows quite rapidly. In particular, there exist $n^{-1+\varepsilon} \ll p_1 \ll p_2 \ll n^{-1/2}$ such that whp the diameter of $G(n,p_1)$ is larger than the diameter of $G(n,p_2)$ while the query complexity of the former is also larger.

Related work

Kannan, Mathieu, and Zhou [22, 30] presented a reconstruction algorithm that uses $\tilde{O}(n^{3/2})$ distance queries for bounded degree graphs. They also proved an information–theoretic lower bound $\Omega(n \log n/\log\log n)$ for trees with maximum degree 3 and asked whether $\tilde{O}(n)$ is achievable for all bounded degree graphs. Tight results for certain families of bounded degree graphs – trees, chordal graphs, and graphs with bounded treelength – were recently obtained

by Bastide and Groenland in [4]. In particular, they proved that for every $\Delta \geq 3$, any randomised algorithm requires $\Omega(n \log n)$ queries to reconstruct n-vertex trees of maximum degree Δ for a certain sequence of sizes of trees, which is tight since there exists a deterministic algorithm to reconstruct such trees using $O(n \log n)$ queries. In [31], Mathieu and Zhou answered the question from [22] for random d-regular graphs with constant d: whp $n \ln^2 n$ distance queries is enough to reconstruct a uniformly random d-regular graph.

A modified version of the problem where the hidden graph is a tree and it is only allowed to query pairs of leaves (that are known) comes from biology: it serves as a model of reconstructing evolutionary (phylogenetic) trees and was introduced by Waterman, Smith, Singh, and Beyer [43]. This problem is well studied: lower bounds on query complexity for deterministic and randomised algorithms were obtained in [25] and [4] respectively, and upper bounds were investigated in [11, 23, 25, 27].

Other types of query oracles explored in the literature include *all-shortest-path*, *all-distances*, and *shortest-path* queries. The first two oracles answer all shortest paths [5, 38] and all distances [5, 14] from a queried vertex respectively. The shortest-path oracle answers a shortest path between a queried pair of vertices [22, 37]. Erlebach, Hall, and Mihal'ák [15] studied reconstruction of $G(n,p)$ from all-shortest-path queries for $p = \text{const}$ and $p = n^{-1+\varepsilon}$ for specific $\varepsilon > 0$.

Mathieu and Zhou [31] also showed that their construction of the set of distance queries can be used to get an upper bound for the metric dimension of random d-regular graphs. Let us recall that the *metric dimension* $\beta(G)$ of a graph G is the minimum cardinality of $S \subset V(G)$ such that all vertices of G have different vectors of distances to the vertices from S. Such *resolving sets* S or their variants were used, in particular, to label canonically vertices of random graphs [3, 26]. The metric dimension of $G(n,p)$ was studied by Bollobás, Mitsche, and Prałat in [10]. In particular, they proved that the metric dimension of $G(n,p)$ undergoes the following "zigzag" behaviour: if $p = n^{-\alpha}$, then $\log_n \beta(G(n,p)) = 1 - (1-\alpha)\lfloor 1/(1-\alpha) \rfloor$ whp; see also [36] for related results. Although in our proof of the upper bound for the distance query complexity of $G(n,p)$ we use a resolving set whose size is far from being optimal, as we show below the "jumps" of the metric dimension and of the query complexity are synchronous – both happen around $n^{-k/(k+1)}$, $k \in \mathbb{Z}_{>0}$.

Our contribution

As follows from the next result, the query complexity drops for the first time when p passes a threshold $n^{-1/2+o(1)}$ on its way down. More generally, we prove that, for every integer constant $k \geq 1$, as soon as p passes a threshold $n^{-(k+1)/(k+2)+o(1)}$, the query complexity drops by a factor of $\Theta(np)$.

▶ **Theorem 1.** *Let $k \geq 1$ be a fixed integer, $\varepsilon > 0$, $n^{-1+\varepsilon} < p < n^{-k/(k+1)-\varepsilon}$, and let $G_n \sim G(n,p)$. Then there exist $C = C(k,\varepsilon) > 0$ and an adaptive algorithm $\mathtt{A}$ such that, for $q = \lfloor C/(n^{k-2}p^k) \rfloor$, whp G_n is q-reconstructible by $\mathtt{A}$.*

We prove Theorem 1 in Section 3.

▶ **Theorem 2.** *Let $k \geq 1$ be a fixed integer, $\varepsilon > 0$, and let $n^{k+1}p^{k+2} - 2\ln n \to \infty$ as $n \to \infty$. Then whp $G_n \sim G(n,p)$ has query complexity at least $\frac{1}{2(k+1+\varepsilon)n^{k-2}p^k}$.*

Theorem 2 is proven in Section 5. Let us emphasise that its proof is based on an explicit argument which is more efficient than the usual information-theoretic approach: we show a lower bound on the query complexity of a deterministic graph in terms of its diameter and maximum degree, see Claim 14. Unlike the information-theoretic approach, our approach

allows to bound the query complexity of any given graph. So, it actually demonstrates that, when the number of queries is less than the bound, even knowing the input graph G does not help choosing the optimal quering strategy – no matter which pairs of vertices are queried, there is always a graph $G' \neq G$ with exactly the same answers.

Note that, for any $w_n \to \infty$ as $n \to \infty$ and any

$$\left(\frac{2\ln n + w_n}{n^{k+1}}\right)^{1/(k+2)} < p < n^{-k/(k+1)-\varepsilon}, \tag{1}$$

the bounds in Theorem 1 and Theorem 2 differ by a constant factor, see Figure 1. So, roughly speaking (ignoring the constant factor) the query complexity in the interval (1) increases with p.

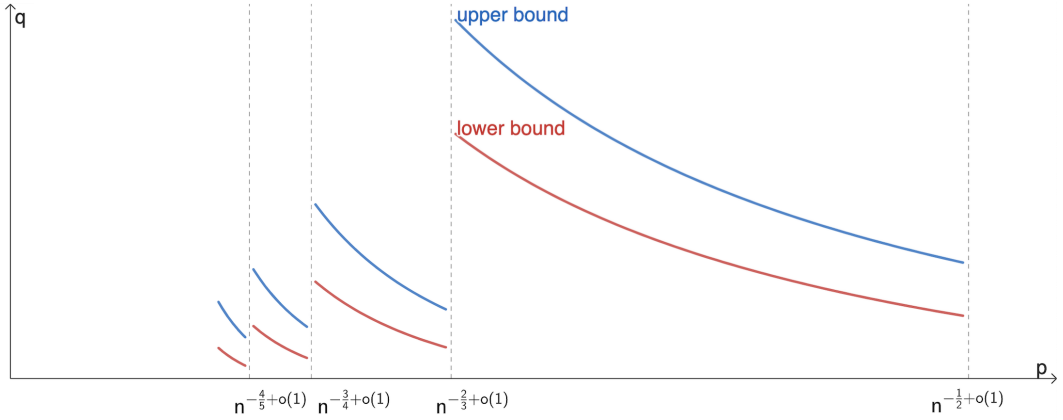


Figure 1 Upper and lower bounds for query complexity of $G(n,p)$ for p satisfying (1).

We also prove tight (up to a constant factor) bounds for "non-adaptive reconstructibility". It turns out that if the set of queried pairs $\mathcal{Q}$ is fixed, then both upper and lower bounds on the minimum $|\mathcal{Q}|$ such that $G(n,p)$ is $\mathcal{Q}$-reconstructable increase by a factor of $\log n$ when p satisfies (1).

▶ **Theorem 3.** *Let $k \geq 1$ be a fixed integer, $\varepsilon > 0$, and let $p > n^{-1+\varepsilon}$. Then there exists a set $\mathcal{Q} = \mathcal{Q}(n) \subset \binom{[n]}{2}$ of size at most $\frac{3}{n^{k-2}p^k} e^{n^k p^{k+1}} \ln n$ such that whp $G_n \sim G(n,p)$ is $\mathcal{Q}$-reconstructible.*

Theorem 3 is proven in Section 4.

▶ **Theorem 4.** *Let $k \geq 1$ be a fixed integer, $\varepsilon > 0$, and let $w_n \to \infty$ as $n \to \infty$. If*

$$\left(\frac{2\ln n + w_n}{n^{k+1}}\right)^{1/(k+2)} < p < \left(\left(\frac{1}{k+1} - \varepsilon\right)\frac{\ln n}{n^k}\right)^{1/(k+1)},$$

then there is no set $\mathcal{Q} = \mathcal{Q}(n)$ of size at most $\frac{1}{8(k+1)^2 n^{k-2}p^k} e^{n^k p^{k+1}} \ln n$ such that whp $G_n \sim G(n,p)$ is $\mathcal{Q}$-reconstructible.

Theorem 4 is proven in Section 6.

Proofs outline

For simplicity of presentation, let us assume that $p = n^{-\alpha}$, where $\frac{k}{k+1} < \alpha < \frac{k+1}{k+2}$ for some $k \in \mathbb{Z}_{>0}$. The crucial observation that allows to prove tight (up to a constant) upper bounds both for adaptive and non-adaptive query complexities is the following: if $\{v_1, x\}$ and $\{v_2, x\}$ are queried and $d(v_2, x) \geq d(v_1, x) + 2$, then v_1 and v_2 are not adjacent.

Although whp the diameter of $G_n \sim G(n,p)$ equals $k+2$, for a sufficiently large fixed set $X \subset [n]$, every vertex $v \in [n]$ has a vertex from X in its k-neighbourhood. Indeed, if X has size $\Theta\left(\frac{\ln n}{n^{k-1}p^k}\right)$, then the expected number of k-paths from a fixed vertex $v \in [n]$ to X is $\Theta(\ln n)$. Thus, using standard concentration inequalities, it is possible to overcome the union bound over the choice of $v \in [n]$ for $|X| = C\frac{\ln n}{n^{k-1}p^k}$ and C sufficiently large. Actually, the following stronger property can be proved: whp for every $v_1, v_2 \notin X$, there exists $x \in X$ such that $d_{G_n}(v_1, x) \le k$ while $d_{G_n \setminus \{v_1,v_2\}}(v_2, x) \ge k+2$, where $G_n \setminus \{v_1, v_2\}$ is a spanning subgraph of G_n that is obtained by excluding the edge $\{v_1, v_2\}$. Thus, as soon as all pairs of vertices with at least one vertex in X are queried, all the remaining adjacencies are reconstructible whp. Indeed, for non-adjacent v_1, v_2, there exists x such that $d(v_1, x) \le k$ and $d(v_2, x) \ge k+2$, thus we may reconstruct all non-adjacencies. All the other pairs of vertices must be adjacent since otherwise if we could exclude, say, the edge $\{v_1, v_2\}$, then there is an $x \in X$ such that $d(v_2, x) \le k+1$ while, in the corrupted graph, the distance between v_2 and x would be at least $k+2$. This proves Theorem 3, see details in Section 4. A counterpart of this very simple algorithm was employed by Mathieu and Zhou in [31], who showed that it allows typically to reconstruct random regular graphs with $(1+o(1))n\ln^2 n$ queries, which is tight, up to a $(\log n)^{1+o(1)}$-factor.

The proof of Theorem 1 in Section 3 is based on similar ideas but also utilises the possibility of updating queries adaptively: indeed, instead of querying all pairs $\{x \in X, v \in [n]\}$ simultaneously, we may split the set X into $\nu = \Theta(\log n)$ equal parts $X_1, \dots, X_\nu$ and then, for every $i = 1, \dots, \nu$, query a pair $\{x \in X_i, v \in [n]\}$ only if v has not yet been excluded from the consideration. A vertex v is excluded from the consideration at time i, if it has sufficiently many (at least M for some large enough constant M) vertices in the intersection of its k-neighbourhood with $X_1 \cup \ldots \cup X_i$. This allows to query the desired number of pairs whp. In order to complete the proof, it remains to observe that whp there are no v_1, v_2 that have M different vertices $x \in X$ so that $d_{G_n \setminus \{v_1,v_2\}}(v_1, x) \le k$ and $d_{G_n \setminus \{v_1,v_2\}}(v_2, x) \le k+1$. This would imply the existence of a good $x \in X$ as in the proof of Theorem 3 for any pair $\{v_1, v_2\}$.

Proofs of both lower bounds in Theorem 2 and Theorem 4 use the notion of an *undetectable pair* in a graph G having diameter $d \ge 3$ with a set of queried pairs $\mathcal{Q} \subset \binom{V(G)}{2}$: a pair of vertices $\{u_1, u_2\}$ is undetectable if $\{u_1, u_2\} \notin \mathcal{Q} \cup E(G)$ and there is no "path" $u_1 w_1 \dots w_\ell u_2$ in G, where $\ell \le d-2$, such that ℓ of its "edges" are actual edges of G, and the one remaining pair of consecutive vertices belongs to $\mathcal{Q}$. It is easy to see that if G has at least one undetectable pair, then it is not reconstructible from $\mathcal{Q}$ since the addition of the edge $\{u_1, u_2\}$ does not change answers to queries. When $\mathcal{Q}$ is small, then a simple counting proof shows that there are undetectable pairs in G_n whp. This proves Theorem 2, see details in Section 5.

In order to improve the lower bound by a factor of $\log n$ for the "non-adaptive reconstructibility" (Theorem 4), we first refine the notion of undetectability, and then introduce a randomised algorithm that finds an undetectable pair (in the refined sense) in G_n whp. Let us call a pair of vertices $\{u_1, u_2\}$ ℓ-undetectable in G, if $\{u_1, u_2\} \notin \mathcal{Q} \cup E(G)$ and there is no "path" $u_1 w_1 \dots w_\ell u_2$ in G such that ℓ of its "edges" are actual edges of G, the one remaining pair of consecutive vertices belongs to $\mathcal{Q}$, and the distance between the two vertices in this pair is at least $\ell+2$. Though the refined notion is weaker, it still allows to prove non-reconstructibility: if G has diameter d and has a pair which is ℓ-undetectable for all $\ell \le d-2$, then G is not $\mathcal{Q}$-reconstructible. Notice that, to decide whether a given pair $\{u_1, u_2\}$ is undetectable, it is sufficient to know only the edges induced by the union of k-neighbourhoods of u_1 and u_2. In Section 6, we show that the following algorithm finds an undetectable pair whp: at every step, choose a pair of vertices $\{u_1, u_2\}$ uniformly at random from the set of *unconsidered* vertices, consider all vertices in k-neighbourhoods of u_1, u_2, and check whether $\{u_1, u_2\}$ is ℓ-undetectable for all $\ell \le k$.

Structure of the paper

The rest of the paper is organised as follows. In Section 2 we recall properties of random graphs that we will use in our proofs: namely, asymptotic behaviour of sizes of balls of small radii and the diameter. Proofs of upper bounds – Theorem 1 and Theorem 3 – appear in Sections 3 and 4 respectively. In Sections 5 and 6 we prove the lower bounds, Theorems 2 and 4 respectively. Finally, in Section 7 we raise several natural questions that we leave unanswered.

2 Properties of random graphs

In this section, we describe certain properties of $G_n \sim G(n,p)$ that we will use in the proofs. We also refer the reader to monographs [9, 16, 20] for a comprehensive exposition of properties of random graphs and of probabilistic tools – especially, to [20, Chapter 2] containing well-known standard concentration inequalities (in particular, Chernoff bounds for binomial distribution [20, Theorem 2.1], Hoeffding's exponential tail bound for the hypergeometric distribution [20, Theorem 2.10], and Harris' lemma [20, Theorem 2.12]) that we will use in the proofs and will not state them explicitly.

Let us state a technical claim about the sizes of k-neighbourhoods in G_n that we will use frequently both for lower and upper bounds. For $v \in [n]$ and $r \in \mathbb{Z}_{\geq 0}$, everywhere in the paper we denote by $N_r(v)$ the (random) r-ball around v in G_n. We will also denote $N_r(U) := \cup_{v\in U} N_r(v)$ for $U \subset [n]$.

▷ Claim 5. Let k be a positive integer and let $\frac{\log^2 n}{n} \ll p \ll (n^{k-1}\log n)^{-1/k}$. Fix $v \in [n]$. With probability $1-o(n^{-2})$, in $G(n,p)$ we have $|N_r(v)| = (np)^r(1+o(\log^{-1} n))$, for all $r \in [k]$.

Proof. From the Chernoff bound, it follows that $|N_1(v)| = np(1+o(1/\log n))$ with probability $1-o(n^{-2})$. Let us prove by induction over r that, for every, $r \leq k$, $|N_r(v)| = (np)^i(1+o(1/\log n))$ with probability $1-o(n^{-2})$. Assume that for an $r \in [k-1]$, this probability bound is already obtained. Let us expose the r-ball around v and denote the set of vertices at distance exactly r by N'_r. Due to the induction assumption, we may suppose that $|N'_r| = (np)^r(1+o(1/\log n)) = o(n/\log n)$. Thus, the number of vertices in $[n]\setminus N_r(v)$ having neighbours in N'_r (denoted by $|N'_{r+1}|$) is distributed as $\mathrm{Bin}(n(1-o(1/\log n)), 1-(1-p)^{|N'_r|})$. Clearly, $N_{r+1}(v) = N_r(v) \sqcup N'_{r+1}$. Since $\mathbb{E}|N'_{r+1}| = n^{r+1}p^{r+1}(1+o(1/\log n)) \gg \log^4 n$, we get that

$$\mathbb{P}\left(\left||N'_{r+1}| - \mathbb{E}|N'_{r+1}|\right| > \mathbb{E}|N'_{r+1}|/(\ln n)^{3/2}\right) = \exp\left[-\Omega(\mathbb{E}|N'_{r+1}|/\ln^3 n)\right] = o(n^{-2})$$

by the Chernoff bound, as needed. ◁

Let us recall the result of Bollobás [7] about the asymptotic distribution of the diameter of G_n, see also [9, Chapter 10.2].

▶ **Theorem 6.** *Let $c \in \mathbb{R}$ be a constant and $p = \left(\frac{2\ln n + c}{n^k}\right)^{1/(k+1)}$. Then the diameter of G_n converges in distribution to a random variable taking values $k+1$ and $k+2$ with probabilities $e^{-e^{-c}/2}$ and $1-e^{-e^{-c}/2}$ respectively.*

Since the property of having diameter at most d is monotone, we immediately get the following corollary.

▶ **Corollary 7.** *If $n^k p^{k+1} - 2\ln n \to \infty$ as $n \to \infty$, then whp the diameter of G_n is at most $k+1$. If $n^k p^{k+1} - 2\ln n \to -\infty$ as $n \to \infty$, then whp the diameter of G_n is at least $k+2$.*

3 Proof of Theorem 1

Let us recall that $k \geq 1$ is a fixed integer, $\varepsilon > 0$, and $n^{-1+\varepsilon} < p < n^{-k/(k+1)-\varepsilon}$. We let $G_n \sim G(n,p)$. Let $M \in \mathbb{Z}$ be large enough and let us choose a large enough real $C \gg M$. Let us also set $\nu := \lceil \log_2 n \rceil$ and fix a set $X \subset [n]$ of size $\nu \left\lfloor C/(2n^{k-1}p^k) \right\rfloor$. We will need the following auxiliary statements.

▶ **Lemma 8.** *Let $i = i(n) \in [\nu]$ and let $X' \subset X$ be a fixed set of size $i \left\lfloor C/(n^{k-1}p^k) \right\rfloor$. Then with probability $1 - o(1/\log n)$, at least $(n - |X|)(1 - 2^{-i})$ vertices of $[n] \setminus X$ have at least M vertices at distance at most k in X'.*

For an edge e of a graph G, we denote by $G \setminus e$ the spanning subgraph of G obtained by deleting e. If $e \notin E(G)$, then we let $G \setminus e = G$.

▶ **Lemma 9.** *Whp there are no $v_1, v_2 \in [n]$ and $u_1, \ldots, u_M \in X$ such that, for all $j \in [M]$,*

$$d_{G_n \setminus \{v_1,v_2\}}(v_1, u_j) \leq k \quad \text{and} \quad d_{G_n \setminus \{v_1,v_2\}}(v_2, u_j) \leq k+1. \tag{2}$$

We prove Lemma 8 and Lemma 9 in Sections 3.1 and 3.2, respectively.

Let us consider a balanced partition of X into ν disjoint sets $X_1, \ldots, X_\nu$ and assume that G_n is a graph satisfying conclusions of Lemma 8 and Lemma 9 deterministically, i.e.:

1 for every $i \in [\nu]$, the number of vertices $v \in [n] \setminus X$ that have at least M vertices $u \in X_1 \cup \ldots \cup X_i$ such that $d(v,u) \leq k$ is at least $(n - |X|)(1 - 2^{-i})$;

2 there are no $v_1, v_2 \in [n]$ and $u_1, \ldots, u_M \in X$ such that (2) holds for all $j \in [M]$.

Let us prove that these conclusions imply the statement of Theorem 1.

Let us first query all pairs from $\binom{X}{2}$ in G_n. Then, we query all pairs from $X_1 \times ([n] \setminus X)$ and find all vertices $v \notin X$ such that there are at least M vertices $u \in X_1$ satisfying $d(v,u) \leq k$. Let $V_1 \subset [n] \setminus X$ be the set of all such vertices v. If $V_1 = [n] \setminus X$, then we set $\hat{i} := 1$. We then proceed by induction: assume that at step $i \in \{1, \ldots, \nu - 1\}$ disjoint sets $V_1, \ldots, V_i \subset [n] \setminus X$ are chosen and $V_1 \cup \ldots \cup V_i \neq [n] \setminus X$. We then query all pairs from $X_{i+1} \times ([n] \setminus (X \cup V_1 \cup \ldots \cup V_i))$ and find all vertices $v \notin X \cup V_1 \cup \ldots \cup V_i$ such that there are at least M vertices $u \in X_1 \cup \ldots \cup X_i \cup X_{i+1}$ satisfying $d(v,u) \leq k$. We let $V_{i+1} \subset [n] \setminus (X \cup V_1 \cup \ldots \cup V_i)$ be the set of all such v. If $V_1 \cup \ldots \cup V_{i+1} = [n] \setminus X$, then we set $\hat{i} := i+1$. From **1**, we get that, for every i, $|V_1 \cup \ldots \cup V_i| \geq (1 - 2^{-i})(n - |X|)$. Thus, the inductive procedure stops at some moment $\hat{i} \leq \nu$, i.e. $V_1 \cup \ldots \cup V_{\hat{i}} = n \setminus X$, and we query

$$\binom{|X|}{2} + \sum_{i=1}^{\hat{i}} |X_i| \cdot |[n] \setminus (X \cup V_1 \cup \ldots \cup V_{i-1})| \leq \left\lfloor \frac{C}{2n^{k-1}p^k} \right\rfloor \sum_{i=1}^{\hat{i}} n \cdot 2^{1-i} < 2n \left\lfloor \frac{C}{2n^{k-1}p^k} \right\rfloor \leq q$$

pairs.

Let us now show that G_n is reconstructible from the set of queried pairs. For every $i \leq \hat{i}$, we have to recover all adjacencies between $v_1 \in V_i$ and $v_2 \in [n]$. Without loss of generality we may assume that v_2 is either from X or from V_j for some $j > i$. Since all pairs from $V_i \times (X_1 \cup \ldots \cup X_i)$ are queried, we may consider only $v_2 \in (X_{i+1} \cup \ldots \cup X_\nu) \cup (V_{i+1} \cup \ldots \cup V_{\hat{i}})$. It is crucial that for all such v_2, the pairs $\{v_2, u\}$ for $u \in X_1 \cup \ldots \cup X_i$ are queried, so that we got both $d(v_1,u)$ and $d(v_2,u)$ from the oracle.

Let us first assume that v_1 and v_2 are non-adjacent in G_n. Let $u_1, \ldots, u_M \in X_1 \cup \ldots \cup X_i$ be at distance at most k from v_1. From **2**, it follows that there exists $u \in \{u_1, \ldots, u_M\}$ such that $d(v_1,u) \leq k$ while $d(v_2,u) \geq k+2$. It may only happen when v_1, v_2 are not adjacent. Thus, we may "reconstruct non-adjacencies" for all pairs of non-adjacent vertices in G_n.

Now, assume that G' is obtained from G_n by removing some edges that belong to $\cup_{i\le\hat{i}} V_i \times ([n]\setminus(X_1\cup\ldots\cup X_i))$ and that for G' we get exactly the same answers as for G_n for all queried pairs. Let $\{v_1,v_2\}\in G_n\setminus G'$, where $v_1\in V_i$ and $v_2\in(X_{i+1}\cup\ldots\cup X_\nu)\cup(V_{i+1}\cup\ldots\cup V_{\hat{i}})$. As above, we let $u_1,\ldots,u_M\in X_1\cup\ldots\cup X_i$ be at distance at most k from v_1 in G_n. From **2**, we get that there exists $u\in\{u_1,\ldots,u_M\}$ such that $d_{G'}(v_2,u)\ge k+2$. This inequality holds true since distances in G' cannot be less than in G_n. But then $d_{G_n}(v_2,u)\ge k+2$ as $d_{G_n}(v_2,u)=d_{G'}(v_2,u)=d(v_2,u)$ by the assumption since the pair $\{v_2,u\}$ was queried. On the other hand, $v_1\sim v_2$ in G_n and $d(v_1,u)=d_{G_n}(v_1,u)\le k$. So $d(v_2,u)\le d(v_1,u)+1\le k+1$ – a contradiction. Thus, edges are also reconstructible, Theorem 1 follows.

3.1 Proof of Lemma 8

Fix $i\in[\nu]$ and $X'\subset X$ of size $i\left\lfloor C/(2n^{k-1}p^k)\right\rfloor$. For $v\in[n]\setminus X$, we let $\mathcal{B}_v$ be the event saying that at least M vertices in X' are at distance at most k from v. We denote $\xi:=\sum_{v\in[n]\setminus X}\mathbb{1}_{\mathcal{B}_v}$ the number of such vertices. We have to prove that $\mathbb{P}(\xi<(n-|X|)(1-2^{-i}))=o(1/\log n)$.

Fix a vertex $v\in[n]\setminus X$. For any $N\in[n]$, subject to the event $\{|N_k(v)|=N\}$, the set $N_k(v)$ is a uniformly random N-subset of $[n]$ that contains v. We derive the following inequality from Hoeffding's exponential tail bound for the hypergeometric distribution (see [20, Theorem 2.10]): for every $N=(np)^k(1+o(1))$,

$$\mathbb{P}\left(|N_k(v)\cap X'|<\frac{1}{4}iC\,\middle|\,|N_k(v)|=N\right)\le\exp\left[-\frac{iC}{16-o(1)}\right]. \tag{3}$$

First of all, (3) implies that $\mathbb{P}\left(|N_k(v)\cap X'|<M\mid|N_k|=N\right)=o(1/(n\log n))$ uniformly over $N=(np)^k(1+o(1))$ if $i\ge 17\ln n/C$. By the union bound and Claim 5, we get that $\xi=n-|X|$ for such i with probability $1-o(1/\log n)$. We then further assume that $i<17\ln n/C$ and that C is so large that $e^{-2i}>n^{-2/(k+1)}$.

For $v\in[n]\setminus X'$, denote by $N_r^{\neg X'}(v)$ the r-ball around v in $G_n[[n]\setminus X']$. Due to Claim 5, with probability $1-o(1/n)$, $|N_{k-1}^{\neg X'}(v)|=(np)^{k-1}(1+o(1))$ for all $v\in[n]\setminus X'$. We then expose only edges induced by $[n]\setminus X'$ and assume that the latter event holds deterministically.

For $v\in[n]\setminus X$, we let $\mathcal{C}_v$ be the event saying that less than M vertices in X' have neighbours in $N_{k-1}^{\neg X'}(v)$. Let $\eta=\sum_{v\in[n]\setminus X}\mathbb{1}_{\mathcal{C}_v}$. Note that $\eta\ge n-|X|-\xi$. Therefore, it suffices to prove that $\mathbb{P}(\eta\ge 2^{-i}(n-|X|))=o(1/\log n)$. A vertex from X' has a neighbour in $N_{k-1}^{\neg X'}(v)$ for a fixed $v\in[n]\setminus X$ with probability

$$1-(1-p)^{(np)^{k-1}(1+o(1))}=1-e^{-n^{k-1}p^k(1+o(1))}\sim n^{k-1}p^k(1+o(1))$$

since $p<n^{-k/(k+1)-\varepsilon}$. Therefore, by the Chernoff bound,

$$\mathbb{P}(\mathcal{C}_v)\le\mathbb{P}\left(\mathrm{Bin}(|X'|,n^{k-1}p^k(1+o(1))<\frac{1}{4}iC\right)\le\exp\left[-\frac{iC}{16-o(1)}\right].$$

We get that $\mathbb{E}\eta\le e^{-i}(n-|X|)$ for $C>16$ and for large n. Since the relation $u\in N_{k-1}^{\neg X'}(v)$ on pairs (u,v) of vertices from $[n]\setminus X'$ is symmetric, every vertex from $[n]\setminus X'$ belongs to at most $(np)^{k-1}(1+o(1))$ balls $N_{k-1}^{\neg X'}(v)$, $v\in[n]\setminus X$. We shall use the following bounded difference inequality applied to η considered as a function of edges between X' and $[n]\setminus X'$:

▶ **Theorem 10** ([32]). *Let $\xi_1,\ldots,\xi_N\sim\mathrm{Bern}(p)$ be independent, and let $f:\mathbb{R}^N\to\mathbb{R}$ satisfy*

$$|f(x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_N)-f(x_1,\ldots,x_{i-1},x_i',x_{i+1},\ldots,x_N)|\le c_i$$

for some $c_1,\ldots,c_N>0$ and every $i\in[N]$, $x_1,\ldots,x_N,x_i'\in\mathbb{R}$. Then, for every $t>0$,

$$\mathbb{P}(|f(\xi_1,\ldots,\xi_N)-\mathbb{E}f(\xi_1,\ldots,\xi_N)|\ge t)\le\exp\left[-\frac{t^2}{2p(1-p)\sum_{i=1}^N c_i^2+\frac{2}{3}t\max_i c_i}\right].$$

Assuming that $\hat{G}_n$ is a graph on $[n]$ such that $\hat{G}_n[[n] \setminus X'] = G_n[[n] \setminus X']$ contains exactly the exposed edges, we get that, for any pair $e \in X' \times ([n] \setminus X')$, values $\eta(\hat{G}_n)$ and $\eta(\hat{G}_n \triangle e)$ differ by at most $(np)^{k-1}(1+o(1))$ since e may change only the values of $\mathbb{1}_{\mathcal{C}_v}$ such that the vertex of e from $[n] \setminus X'$ belongs to $N_{k-1}^{\neg X'}(v)$. Therefore, we may set all c_i equal to $(np)^{k-1}(1+o(1))$. Then Theorem 10 implies that, for some constant $c > 0$,

$$\begin{aligned}\mathbb{P}(\eta \geq 2^{-i}(n-|X|)) &\leq \mathbb{P}\left(\eta \geq \frac{e}{2} \cdot e^{-i}(n-|X|)\right) \\ &\leq \exp\left[-c\frac{e^{-2i}(n-|X|)^2}{p(n-|X'|)|X'|(np)^{2k-2} + e^{-i}(n-|X|)(np)^{k-1}}\right] \\ &\leq \exp\left[-\frac{2c}{C+1} \cdot \frac{e^{-2i}n}{i(np)^{k-1}}\right] \leq \exp\left[-n^{\varepsilon(k-1)+o(1)}\right] = o\left(\frac{1}{\log n}\right),\end{aligned}$$

where the penultimate inequality is due to $e^{-2i} > n^{-2/(k+1)}$, completing the proof.

3.2 Proof of Lemma 9

Assume that $M \gg M' \gg 1/\varepsilon$.

▷ Claim 11. Fix a vertex $v \in [n]$. With probability $1 - o(n^{-2})$, $|N_{k-1}(v) \cap X| \leq M'$.

Proof. Due to Claim 5, we have that $|N_{k-1}(v)| = (np)^{k-1}(1+o(1))$ with probability $1-o(n^{-2})$. For simplicity of presentation, let us assume that $v \notin X$ (the case $v \in X$ does not differ much). Subject to $|N_{k-1}(v)| = N+1$, the random variable $|N_{k-1}(v) \cap X|$ has a hypergeometric distribution, implying that

$$\begin{aligned}\mathbb{P}(|N_{k-1}(v) \cap X| > M') &\leq (1+o(1)) \max_{N=(np)^{k-1}(1+o(1))} \frac{\binom{|X|}{M'}\binom{n-|X|-1}{N-M'}}{\binom{n-1}{N}} \\ &\leq (1+o(1))\left(\frac{|X|-M'}{n}\right)^{M'} = o(n^{-2}).\end{aligned}$$

◁

Let us define the following family of graphs $\mathcal{H}$ on $[n]$: fix a vertex v_1 and draw from this vertex $\lfloor M/M' \rfloor$ disjoint paths of length at most k to distinct vertices from X and avoiding the edge $\{v_1, v_2\}$; fix another vertex v_2 and draw from this vertex a path of length at most $k+1$ to each of the paths starting at v_1 and avoiding the edge $\{v_1, v_2\}$: these $\lfloor M/M' \rfloor$ paths from v_2 are not necessarily disjoint.

▷ Claim 12. Whp G_n does not contain any subgraph from $\mathcal{H}$.

Proof. First of all note that any $H \in \mathcal{H}$ has at least $\lfloor M/M' \rfloor$ and at most $2k\lfloor M/M' \rfloor + 2$ vertices. Moreover, for any x in this range, the number of graphs in $\mathcal{H}$ with exactly x vertices is at most $c_x n^{x-\lfloor M/M' \rfloor}|X|^{\lfloor M/M' \rfloor}$ for a certain constant c_x – there are at most $|X|^{\lfloor M/M' \rfloor}$ choices for the $\lfloor M/M' \rfloor$ ends in X of disjoint paths emanating from v_1 and at most $n^{x-\lfloor M/M' \rfloor}$ ways to choose the remaining $x - \lfloor M/M' \rfloor$ vertices. Let us bound from below the number of edges in such a graph and apply the union bound. Let us fix vertices v_1, v_2 and $u_1, \ldots, u_{\lfloor M/M' \rfloor}$ that play the role of ends of paths from v_1 in X. Note that a graph from $\mathcal{H}$ can be obtained from this set of vertices by a sequential addition of $2\lfloor M/M' \rfloor$ paths joining two vertices that are presented in the graph at the previous step. First $\lfloor M/M' \rfloor$ paths are of length at most k, and the last $\lfloor M/M' \rfloor$ paths are of length at most $k+1$. Assuming that at the j-th step the number of added vertices equals x_j, we get that the constructed graph H has density

$$\rho(H) = \frac{|E(H)|}{|V(H)|} = \frac{\sum_{j=1}^{2\lfloor M/M' \rfloor}(x_i+1)}{2+\lfloor M/M' \rfloor + \sum_{j=1}^{2\lfloor M/M' \rfloor} x_i} = \frac{x-2+\lfloor M/M' \rfloor}{x} \geq \frac{(2k+1)\lfloor M/M' \rfloor}{2k\lfloor M/M' \rfloor + 2}.$$

Finally, assuming $\lfloor M/M' \rfloor > \frac{3}{\varepsilon(k+1)}$, the expected number of subgraphs $H \in \mathcal{H}$ in G_n is at most

$$\sum_x c_x n^{x-\lfloor M/M' \rfloor} |X|^{\lfloor M/M' \rfloor} p^{x\frac{(2k+1)\lfloor M/M' \rfloor}{2k\lfloor M/M' \rfloor + 2}} = O\left(\left(\nu n^{k+2/\lfloor M/M' \rfloor} p^{k+1}\right)^{\lfloor M/M' \rfloor}\right) = O(n^{-1}).\lhd$$

Let us now fix v_1, v_2 and assume that the graph $G' := G_n \setminus \{v_1, v_2\}$ satisfies the following properties deterministically:

- for all $v \in [n]$, $|N_{k-1}(v) \cap X| \le M'$;
- $u_1, \ldots, u_M \in N_k(v_1)$;
- G' does not contain a graph from family $\mathcal{H}$.

It is then sufficient to prove that there exists $j \in [M]$ such that $u_j \notin N_{k+1}(v_2)$.

Take a shortest path P_1 (of length at most k) between v_1 and u_1. We then proceed by induction. Assume that at step $j < \lfloor M/M' \rfloor$, we found j disjoint shortest paths $P_1, \ldots, P_j$ from v_1 to $u_1, \ldots, u_j$ respectively. Let $w_1, \ldots, w_j$ be the first vertices (after v_1) on $P_1, \ldots, P_j$ respectively. We have that there exists $j' \in [M] \setminus [j]$ such that $u_{j'} \notin N_{k-1}(w_1) \cup \ldots \cup N_{k-1}(w_j)$. Without loss of generality let $j' = j+1$. Observe that the shortest path P_{j+1} between v_1 and u_{j+1} cannot intersect any of $P_1, \ldots, P_j$ outside of v_1. Eventually we get $\lfloor M/M' \rfloor$ disjoint paths from v_1 to $u_1, \ldots, u_{\lfloor M/M' \rfloor}$. If, for some $j \in [\lfloor M/M' \rfloor]$, $u_j \notin N_{k+1}(v_2)$, we are done. Otherwise, G_n contains a graph from $\mathcal{H}$ – a contradiction.

4 Proof of Theorem 3

Let $\lambda = n^k p^{k+1}$. Without loss of generality we may assume that

$$p \le \left(\frac{(k+o(1))\ln n}{(k+1)n^k}\right)^{1/(k+1)} \tag{4}$$

so that $\frac{3e^\lambda \ln n}{n^{k-2}p^k} \le \binom{n}{2}$. Let X be a fixed set of vertices of size $\left\lfloor 3e^\lambda \ln n/(n^{k-1}p^k)\right\rfloor \le \frac{n}{2}$. Let us query all the pairs from $[X \times ([n] \setminus X)] \cup \binom{X}{2}$. Thus, we query at most $\frac{3}{n^{k-2}p^k} e^\lambda \ln n$ pairs. As in the previous section, we let $G_n \sim G(n,p)$, and, for $v \in [n]$, we let $N_k(v)$ denote the set of vertices in G_n at distance at most k from v.

We further show that whp in G_n for every $v_1, v_2 \in [n] \setminus X$, there is an $x \in X$ such that x is at distance at most k from v_1 and at least $k+2$ from v_2 in the graph obtained from G_n by removing the edge $\{v_1, v_2\}$. It would immediately imply that whp G_n is q-reconstructible. Indeed, we need only to reconstruct adjacency relations between pairs of vertices $v_1, v_2 \notin X$. If v_1, v_2 are non-adjacent, then we queried $\{v_1, x\}$, $\{v_2, x\}$ such that $d(v_1, x) \le k$, $d(v_2, x) \ge k+2$. This may only happen when v_1, v_2 are non-adjacent. Then we can reconstruct all non-adjacencies. It remains to prove that we cannot delete any subset of edges from $G_n[[n] \setminus X]$ without changing the answers to the queries. If $v_1 \sim v_2$, then we queried $\{v_1, x\}$, $\{v_2, x\}$ such that $d(v_1, x) \le k$, $d(v_2, x) \le d(v_1, x) + 1$, and any shortest path from x to v_2 follows the edge $\{v_1, v_2\}$. Thus, the edge $\{v_1, v_2\}$ must be in the graph.

Therefore, the following lemma immediately imples Theorem 3.

▶ **Lemma 13.** *Whp for every $v_1, v_2 \in [n] \setminus X$, there is an $x \in X$ such that*

$$d_{G_n \setminus \{v_1, v_2\}}(v_1, x) \le k \quad \text{and} \quad d_{G_n \setminus \{v_1, v_2\}}(v_2, x) \ge k+2.$$

Proof. Fix vertices $v_1, v_2 \notin X$ and remove the edge $\{v_1, v_2\}$ if it is in G_n. Let N' be the (closed) $(k-1)$-neghbourhood of v_1 in $G_n \setminus [X \cup \{v_2\}]$, and let N_i'' be the (closed) i-neghbourhood of v_2 in $G_n \setminus [N' \cup X]$ for $i \in \{k-1, k\}$. Note that, as soon as N' is

exposed, N''_{k-1} and N''_k are defined solely by adjacencies that are entirely outside N' and their cardinalities do not depend on edges induced by N' – only on the size of N'. Therefore, due to Claim 5, we may assume that $|N'| = (np)^{k-1}(1+o(1/\log n))$, $|N''_i| = (np)^i(1+o(1/\log n))$, $i \in \{k-1, k\}$. By the Chernoff bound and due to (4), the number of neighbours of vertices from N''_{k-1} in N' is $o((np)^{k-1}/\log n)$ with probability $1-o(n^{-2})$. Note that the above makes sense only when $k \geq 2$. In the case $k = 1$, we do not need to apply the Chernoff bound since $N' = \{v_1\}$, $N''_{k-1} = \{v_2\}$, and there is no edge $\{v_1, v_2\}$ in the considered graph.

We then exclude from N' the neighbours of N''_{k-1} and get that the refined set N'_0 still has the size $(np)^{k-1}(1+o(1/\log n))$. If we can prove that with probability $1-o(n^{-2})$, there exists a vertex $x \in X$ that has a neighbour in N'_0 and does not have neighbours in $N''_k \sqcup (N' \setminus N'_0)$, we immediately get the statement of Lemma 13: if P is a path of length at most $k+1$ from v_2 to x in $G_n \setminus \{v_1, v_2\}$, then by construction the neighbour of x on P is in N'_0. Then the farthest from x vertex in $P \cap N'_0$ has a neighbour in $P \cap N''_{k-1}$ – a contradiction. The probability bound on the latter event is also immediate since

$$\begin{aligned}
&\mathbb{P}\left(\mathrm{Bin}\left(|X|, \left(1-(1-p)^{|N'_0|}\right)(1-p)^{|N''_k|+|N'\setminus N'_0|}\right) = 0\right) \\
&= \left(1-\left(1-(1-p)^{(np)^{k-1}(1+o(1/\log n))}\right)(1-p)^{(np)^k(1+o(1/\log n))}\right)^{|X|} \\
&\overset{(4)}{=} \exp\left[-(1+o(1))|X|n^{k-1}p^k e^{-\lambda}\right] = o(n^{-2}). \qquad \blacktriangleleft
\end{aligned}$$

5 Proof of Theorem 2

Without loss of generality, we assume $p = o(1)$. Let us show that Theorem 2 follows from

▷ Claim 14. Let $d \geq 3$, $\varepsilon > 0$. There exists $C = C(d, \varepsilon) > 0$ such that, for every graph G on $[n]$ with diameter at most d and maximum degree $\Delta > C$, its query complexity q satisfies:

$$q(d-1+\varepsilon)\Delta^{d-2} \geq \binom{n}{2} - |E(G)|. \tag{5}$$

Indeed, whp G_n has $(1+o(1))\binom{n}{2}p$ edges, diameter at most $k+2$ (by Corollary 7), and maximum degree $np(1+o(1))$ (by Claim 5, see also [6]), implying that the query complexity of $G(n,p)$ is at least $\frac{1}{2(k+1+\varepsilon)n^{k-2}p^k}$ whp, as needed.

Proof of Claim 14. Let q be an integer that does not satisfy (5). Let $\mathcal{Q} \subset \binom{[n]}{2}$ have size q. Let us call a pair of vertices $\{u_1, u_2\}$ *undetectable in* G if:

- u_1, u_2 are not adjacent in G,
- the pair $\{u_1, u_2\}$ was not queried,
- there is no "path" $u_1 w_1 \dots w_\ell u_2$, $\ell \leq d-2$, such that ℓ of its "edges" are actual edges of G, and the one remaining pair of consecutive vertices belongs to $\mathcal{Q}$.

Assuming that the pair $\{u_1, u_2\}$ is undetectable in G, observe that the addition of the edge $\{u_1, u_2\}$ agrees with the answers to all the queries. Indeed, if it affects some queried pair $\{w, w'\}$, then there should be a path $w \dots u_1 u_2 \dots w'$ of length at most $d-1$ in $G \cup \{u_1, u_2\}$ – a contradiction.

It remains to prove that G has an undetectable pair. For an appropriate choice of C, the number of pairs that do not satisfy the third property of an undetectable pair is at most

$$\sum_{\ell=1}^{d-2}(\ell+1)q\Delta^\ell < q((d-1)\Delta^{d-2} + (d-2)^2\Delta^{d-3}) < q(d-1+\varepsilon)\Delta^{d-2} - q$$

On the other hand, the number of pairs that satisfy the first two properties is at least $\binom{n}{2} - |E(G)| - q > q(d-1+\varepsilon)\Delta^{d-2} - q$, implying the existence of an undetectable pair. ◁

6 Proof of Theorem 4

Let $G_n \sim G(n,p)$, $N := \left\lfloor \frac{e^\lambda \ln n}{8(k+1)^2 n^{k-2} p^k} \right\rfloor$, where $\lambda = n^k p^{k+1}$. We assume that a set $\mathcal{Q} \subset \binom{[n]}{2}$ of N pairs of vertices from $[n]$ is fixed in advance to be queried. We call a pair $\{u_1, u_2\}$ *small* if

- the number of $\{v_1, v_2\} \in \mathcal{Q}$ such that $v_1 \in N_{k'}(u_1)$ and $v_2 \in N_{k-k'}(u_2)$ for some $k' \in \{0, 1, \dots, k\}$ is at most $M := 7(k+1)(np)^k N/n^2$;
- there are no $\{v_1, v_2\} \in \mathcal{Q}$ such that $v_1 \in N_{k'}(u_1)$ and $v_2 \in N_{k-1-k'}(u_2)$ for some $k' \in \{0, 1, \dots, k-1\}$;
- there are no $\{v_1, v_2\} \in \mathcal{Q}$ such that, for some $k' \in \{0, 1, \dots, k\}$, $v_1 \in N_{k'}(u_1)$, $v_2 \in N_{k-k'}(u_2)$, and either $v_2 \in N_k(u_1)$ or $v_1 \in N_k(u_2)$.

▷ Claim 15. Whp at least $\frac{1}{2}\binom{n}{2}$ pairs of vertices are small.

Proof. Due to Claim 5, whp, for every $k' \in \{0, 1, \dots, k\}$, for every $\{v_1, v_2\} \in \mathcal{Q}$, there are at most $(1+o(1))(np)^k$ *ordered* pairs (u_1, u_2) such that $v_1 \in N_{k'}(u_1)$ and $v_2 \in N_{k-k'}(u_2)$. Then whp the following holds: if the number of pairs $\{u_1, u_2\}$ satisfying the first condition in the definition of a small pair is less than $\frac{5}{6}\binom{n}{2}$, then $\frac{1}{6}n(n-1)M < (1+o(1))(k+1)(np)^k N$. The later inequality is false for large enough n. Similarly, whp, for every $\{v_1, v_2\} \in \mathcal{Q}$, there are at most $(1+o(1))k(np)^{k-1}$ *ordered* pairs (u_1, u_2) such that $v_1 \in N_{k'}(u_1)$ and $v_2 \in N_{k-1-k'}(u_2)$ for some $k' \in \{0, 1, \dots, k-1\}$. Then whp the following holds: if the number of pairs $\{u_1, u_2\}$ satisfying the second condition in the definition of a small pair is less than $\frac{5}{6}\binom{n}{2}$, then $\frac{1}{6}n(n-1) < (1+o(1))k(np)^{k-1}N$. The later inequality is false for large enough n as well.

Let us finally show that whp the number of pairs $\{u_1, u_2\}$ satisfying the third condition in the definition of a small pair is at least $\frac{5}{6}\binom{n}{2}$. We will do it in two steps: first, we will show that whp there are only $o(n^2)$ pairs $\{u_1, u_2\}$ that have in their "neighbourhood" $N_{k'}(u_1) \times N_{k-k'}(u_2)$ queried pairs $\{v_1, v_2\}$ with $d(v_1, v_2) \le k$; second, we will observe that whp as soon as a queried pair $\{v_1, v_2\}$ has $d(v_1, v_2) > k$, then the intersection of the k'-ball around v_1 and the k-ball around v_2 has size around $(np)^{k+k'}/n$ – that will be sufficient to finish the proof.

Fix a queried pair $\{v_1, v_2\}$. Since the expected number of paths of length at most k between v_1, v_2 is at most $n^{k-1}p^k(1+o(1))$, we get that the distance between v_1, v_2 is at most k with probability at most $n^{k-1}p^k(1+o(1))$ by Markov's inequality. Therefore, the expected number of pairs in $\mathcal{Q}$ that are at distance at most k is at most $n^{k-1}p^k(1+o(1)) \cdot N$. By Markov's inequality, we get that the number of such pairs in $\mathcal{Q}$ is $O_p(n^{k-1}p^k N)$. As we noted above, whp, for every pair $\{v_1, v_2\} \in \mathcal{Q}$, there are at most $(1+o(1))(k+1)(np)^k$ pairs (u_1, u_2) such that $v_1 \in N_{k'}(u_1)$, $v_2 \in N_{k-k'}(u_2)$ for some $k' \in \{0, 1, \dots, k\}$. Let us fix a sequence w_n growing to infinity with n sufficiently slowly. We get that whp, if the number of pairs $\{u_1, u_2\}$ that have $v_1 \in N_{k'}(u_1)$ and $v_2 \in N_{k-k'}(u_2)$ for some $k' \in \{0, 1, \dots, k\}$ such that $\{v_1, v_2\} \in \mathcal{Q}$ and $d(v_1, v_2) \le k$ is at least $\frac{1}{12}\binom{n}{2}$, then $\frac{1}{12}n(n-1) < w_n(np)^k n^{k-1} p^k N$. This is false for all large enough n. This proves that whp the number of pairs satisfying the last property in the definition of small pairs with the restriction that $d(v_1, v_2) \le k$ (note that it always holds for $k' \in \{0, k\}$) is at least $\frac{11}{12}\binom{n}{2}$. It then remains to prove that whp the number of pairs $\{u_1, u_2\}$ with no $\{v_1, v_2\} \in \mathcal{Q}$ such that $d(v_1, v_2) > k$ and, for some $k' \in [k-1]$, $v_1 \in N_{k'}(u_1)$, $v_2 \in N_{k-k'}(u_2)$, and either $v_2 \in N_k(u_1)$ or $v_1 \in N_k(u_2)$, is at least $\frac{11}{12}\binom{n}{2}$ as well.

As promised, we first show that whp, for any $\{v_1, v_2\}$ such that $d(v_1, v_2) > k$ and any $k' \in [k]$, the intersection of the k'-ball around v_1 and the k-ball around v_2 has size at most $(1+o(1))(np)^{k+k'}/n + \ln^2 n$. Fix v_2 and expose the k-ball around it. Due to Claim 5, with probability $1 - o(n^{-2})$, this ball $N_k(v_2)$ has size $(1+o(1))(np)^k$. Let us now observe that we may assume that all vertices from $S := N_k(v_2) \setminus N_{k-1}(v_2)$ have degrees at most $\ln^2 n$

in $N_k(v_2)$. Indeed, the complementary event implies that there exists a vertex $w \in S$ such that the total number of $(k+1)$-paths and k-paths between u in v_2 is at least $\ln^2 n$, which is false with probability $1-o(n^{-2})$: due to [40, Theorem 5], there exists $C>0$ such that, with probability $1-o(n^{-2})$, the number of paths of length at most $k+1$ between any two vertices in $G(n,p)$ is at most $C\ln n$.

Then, fix $v_1 \notin N_k(v_2)$. For $i \in \{0,1,\dots,k'-1\}$, let N'_i be the i-neighbourhood of v_1 in $[n]\setminus N_k(v_2)$, and let $N_i^{\cap}$ be the set of neighbours of N'_i in $N_k(v_2)$. Note that, for every vertex $w \in N_{k'}(v_1)\cap N_k(v_2)$, a shortest path from w to v_1 leaves $N_k(v_2)$ at some vertex $w' \in S$. Thus, the set $N_{k'}(v_1)\cap N_k(v_2)$ coincides with $\cup_{i=0}^{k'-1}[N_{k'-1-i}(N_i^{\cap})\cap N_k(v_2)]$. Then

$$|N_k(v_2)\cap N_{k'}(v_1)| \le \sum_{i=0}^{k'-2} |N_i^{\cap}|\ln^2 n (np)^{k'-2-i} + |N_{k'-1}^{\cap}| \tag{6}$$

with probability $1-o(n^{-2})$. Moreover, due to Claim 5, with probability $1-o(n^{-2})$, for every $i\in\{0,1,\dots,k'-1\}$, $|N'_i| \le (np)^i(1+o(1))$.

Expose all N'_i, $i\in\{0,1,\dots,k'-1\}$, and assume that $|N'_i| \le (np)^i(1+o(1))$ and $|N_k(v_2)| \le (1+o(1))(np)^k$ hold deterministically. Since $|N_i^{\cap}|$ is the number of vertices in $N_k(v_2)$ that have at least one neighbour in N'_i, we get, by the Chernoff bound,

$$\mathbb{P}(|N_i^{\cap}| > (np)^{k+i}p(1+(\ln\ln n)^{-1}) + \ln^2 n) \le$$
$$\mathbb{P}\left(\mathrm{Bin}\left((np)^k(1+o(1)), 1-(1-p)^{(np)^i(1+o(1))}\right) > (np)^{k+i}p\left(1+\frac{1}{\ln\ln n}\right)+\ln^2 n\right) = o\left(n^{-2}\right).$$

We finally get from (6) that with probability $1-o(n^{-2})$,

$$|N_{k'}(v_1)\cap N_k(v_2)| \le \sum_{i=0}^{k'-2} \ln^2 n (np)^{k'-2-i}((np)^{k+i}p(1+o(1))+\ln^2 n) +$$
$$(np)^{k+k'-1}p(1+o(1))+\ln^2 n = (1+o(1))(np)^{k+k'-1}p+\ln^2 n,$$

as needed. From this and since whp, for every vertex v_2 and every $k'\in[k-1]$, $|N_{k-k'}(v_2)| = (1+o(1))(np)^{k-k'}$ due to Claim 5, we get that whp, for every $\{v_1,v_2\}\in\mathcal{Q}$ satisfying $d(v_1,v_2)>k$, there at most $k\left(\frac{(np)^{2k}}{n}+(np)^{k-1}\ln^2 n\right)$ pairs (u_1,u_2) such that,

$$\text{for some } k'\in[k-1],\quad v_1\in N_{k'}(u_1),\quad v_2\in N_{k-k'}(u_2),\quad \text{and}\quad v_2\in N_k(u_1). \tag{7}$$

So, the event that the number of pairs $\{u_1,u_2\}$ that have a pair $\{v_1,v_2\}$ satisfying $d(v_1,v_2)>k$ and (7) is at least $\frac{1}{24}\binom{n}{2}$ implies

$$\frac{1}{24}n(n-1) < k\left(\frac{(np)^{2k}}{n}+(np)^{k-1}\ln^2 n\right)N,$$

which is false for all large n. Due to symmetry, we get that whp the number of pairs $\{u_1,u_2\}$ that have a pair $\{v_1,v_2\}$ satisfying $d(v_1,v_2)>k$, $v_1\in N_{k'}(u_1)$, $v_2\in N_{k-k'}(u_2)$, and $v_1\in N_k(u_2)$ is less than $\frac{1}{24}\binom{n}{2}$ as well, completing the proof. ◁

Let $\ell\in[k]$. We call a pair of vertices (u_1,u_2) *ℓ-undetectable*, if

- u_1,u_2 are not adjacent in G_n and $\{u_1,u_2\}\notin\mathcal{Q}$,
- there is no "path" $u_1w_1\dots w_\ell u_2$, such that ℓ of the pairs of consecutive vertices $\{u_1,w_1\}$, $\{w_1,w_2\}$, …, $\{w_\ell,u_2\}$ are actual edges of G_n, the one remaining pair of consecutive vertices is from $\mathcal{Q}$, and the distance between the vertices in this pair is at least $\ell+2$ in G_n.

Let us recall that whp G_n has diameter $k+2$ due to Corollary 7. Assuming that the pair $\{u_1, u_2\}$ is ℓ-undetectable for all $\ell \in [k]$, observe that whp the addition of the edge $\{u_1, u_2\}$ agrees with the answers to all the queries. Indeed, if it affects some queried pair $\{w, w'\}$, then there should be a path $w \dots u_1 u_2 \dots w'$ of length $\ell + 1 \le k + 1$ in $G_n \cup \{u_1, u_2\}$ while in G_n the distance between w, w' should be at least $\ell + 2$. This contradicts the fact that $\{u_1, u_2\}$ is ℓ-undetectable since if we consider the sequence $u_2 \dots w' w \dots u_1$ obtained form the path $w \dots u_1 u_2 \dots w'$ by replacing the prefix $w \dots u_1$ and the suffix $u_2 \dots w'$, we get that the pair $\{w, w'\}$ is queried, $d(w, w') \ge \ell + 2$, and all the other consecutive pairs of vertices are edges of G_n. Thus, whp the event that there exists a pair $\{u_1, u_2\}$ which is ℓ-undetectable for all $\ell \in [k]$ implies that G_n is not $\mathcal{Q}$-reconstructible. It then remains to prove that whp G_n has a pair $\{u_1, u_2\}$ which is ℓ-undetectable for all $\ell \in [k]$.

Let us now consider the following iterative procedure. At every step $i \ge 1$, we are given with a set of "considered" vertices $X_i \subset [n]$ (initially $X_1 = \varnothing$), then we sample a pair $\{u_1, u_2\}$ uniformly at random from $\binom{[n] \setminus X_i}{2}$ and expose $G_n[N_k(u_1)]$ and $G_n[N_k(u_2)]$. If the pair $\{u_1, u_2\}$ is not small or u_1, u_2 are adjacent (note that the exposed edges are enough to get these decisions), then we just skip this step and switch to the step $i + 1$ with $X_{i+1} := X_i \cup N_k(u_1) \cup N_k(u_2)$.

Now, assume that at the current step i, we observe that the pair $\{u_1, u_2\}$ is small and u_1, u_2 are not adjacent. Note that the pair $\{u_1, u_2\}$ is ℓ-undetectable for every $\ell \in [k-1]$ since it is small (so, due to the first bullet point in the definition of a small pair, it cannot have any "path" of length at most k from the definition of an undetectable pair). The crucial observation is that the edges between $N_k(u_1) \setminus N_k(u_2)$ and $N_k(u_2) \setminus N_k(u_1)$ have not been exposed yet unless they are entirely inside X_i. We then expose these missing edges and show that they define an event that has sufficiently large probability and implies that the pair $\{u_1, u_2\}$ is k-undetectable (see Claim 16 below). After this step, we switch to the step $i + 1$ with $X_{i+1} := X_i \cup N_k(u_1) \cup N_k(u_2)$ as before.

We perform $\tau = \lfloor n^{1/(k+1)-o(1)} \rfloor$ steps, where τ is chosen so that $\tau(np)^k = o(n/\log n)$. Due to Claim 5, whp (in the measure of the union of the balls exposed before this step) $|X_\tau| = o(n/\log n)$, implying that whp (in the same measure), for every $i \le \tau$, the probability that the pair $\{u_1, u_2\}$ we sample at step i is small (in the measure of the random pair $\{u_1, u_2\}$) is at least $1/2 - o(1)$ by Claim 15.

▷ Claim 16. There exists an event $\mathcal{B}_{u_1,u_2}$ that depends only on edges between $N_k(u_1) \setminus N_k(u_2)$ and $N_k(u_2) \setminus N_k(u_1)$, implies that the pair $\{u_1, u_2\}$ is k-undetectable, and satisfies

$$\mathbb{P}(\mathcal{B}_{u_1,u_2} \mid G_n[X_i],\, \{u_1, u_2\} \notin E(G_n),\, \{u_1, u_2\} \text{ is small}) \ge n^{-7/(8(k+1))+o(1)}.$$

Proof. Fix $k' \le k$, $v_1 \in N_{k'}(u_1)$, $v_2 \in N_{k-k'}(u_2)$ such that the pair $\{v_1, v_2\}$ was queried. Note that since $\{u_1, u_2\}$ is small, it may only happen when $v_1 \notin N_k(u_2)$, $v_2 \notin N_k(u_1)$.

Without loss of generality, assume that $k' \le k - k'$. Note that, if there exists an edge in G_n between $A(v_1) := N_{k-k'}(v_1) \setminus (X_i \cup N_k(u_2))$ and $A(v_2) := N_{k'}(v_2) \setminus N_k(u_1)$, then the distance between v_1, v_2 is at most $k + 1$. Due to Claim 5, whp

$$|N_{k-k'}(v_1)| = (1 + o(1/\log n))(np)^{k-k'}, \quad |N_k(u_1)| = (1 + o(1/\log n))(np)^k,$$
$$|N_{k'}(v_2)| = (1 + o(1/\log n))(np)^{k'}, \quad |N_k(u_2)| = (1 + o(1/\log n))(np)^k.$$

Since $|X_i| = o(n/\log n)$, due to Hoeffding's exponential tail bound for the hypergeometric distribution, whp $|A(v_1)| = (1 + o(\log^{-1} n))(np)^{k-k'}$ and $|A(v_2)| = (1 + o(\log^{-1} n))(np)^{k'}$. Then

$$\mathbb{P}(\text{there is an edge between } A(v_1) \text{ and } A(v_2)) = 1 - (1-p)^{(1+o(1/\log n))(np)^k} = 1 - e^{-\lambda + o(1)}.$$

Since $\{u_1, u_2\}$ is small, the number of queried $\{v_1, v_2\}$ in $\bigcup_{k'=0}^{k} N_{k'}(u_1) \times N_{k-k'}(u_2)$ is at most M. Now, we define the desired event $\mathcal{B}_{u_1,u_2}$: for every k' and every queried pair $\{v_1 \in N_{k'}(u_1), v_2 \in N_{k-k'}(u_2)\}$, there exists an edge between the respective $A(v_1)$ and $A(v_2)$. Then, due to the Harris inequality [17],

$$\mathbb{P}(\mathcal{B}_{u_1,u_2}) \geq (1 - e^{-\lambda+o(1)})^M \sim \exp\left[-(7/8 + o(1)) \ln n/(k+1)\right]. \qquad \lhd$$

Whp in at least $(1/2 - o(1))\tau$ steps we get a small non-adjacent pair. Due to Claim 16, we then get that the probability that, for at least one of these small pairs $\{u_1, u_2\}$, the respective event $\mathcal{B}_{u_1,u_2}$ happens is at least

$$1 - \left(1 - n^{-7/(8(k+1))+o(1)}\right)^{(1/2-o(1))\tau} = 1 - \exp\left[-\frac{1}{2} n^{\frac{1+o(1)}{8(k+1)}}\right] = 1 - o(1),$$

completing the proof.

7 Open questions

Although we get tight bounds on distance query complexity for the binomial random graph $G(n,p)$ when the edge probability p satisfies (1), as p gets closer to a hitting time of a diameter increment, the difference between the bounds increases. In particular, there is a polynomial gap between the bounds even for the "non-adaptive reconstructibility" when $p = \Theta((n^{-k} \ln n)^{1/(k+1)})$ satisfies

$$(1 - o(1)) \left(\frac{\ln n}{(k+1)n^k}\right)^{1/(k+1)} < p < \left(\frac{2\ln n + O(1)}{n^k}\right)^{1/(k+1)}.$$

It would be interesting to get at least the right power of n in the query complexity in this case.

The further natural step is to generalise our results to growing diameter, i.e. to $p = n^{-1+o(1)}$. Although Theorems 2, 3, and 4 are generalised directly to $p \gg \ln^2 n/n$, it is unclear whether it is possible to extend Theorem 1 without any significant modification of the proof method. The main complication is in the application of the bounded difference inequality (Theorem 10) since we use the fact that $e^{-2i} > n^{-2/(k+1)}$ for all $i < \ln n/C$ and some sufficiently large constant C. If k is growing, then there is no such C.

As we mention in Introduction, Mathieu and Zhou [31] proved that, for constant d, the query complexity of the random d-regular graph on $[n]$ is at most $n \ln^2 n$ whp. On the other hand, it is easy to prove the information–theoretic lower bound $\Omega(n \log n/\log\log n)$. It seems likely that the query complexity of the random d-regular graph actually equals $n(\log n)^{1-o(1)}$ whp, and we are asking whether the lower bound is tight.

References

1 R. Afshar, M. T. Goodrich, P. Matias, and M. C. Osegueda. Reconstructing Biological and Digital Phylogenetic Trees in Parallel. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:24, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2020.3.

2 R. Arratia, D. Martin, G. Reinert, and M. S. Waterman. Poisson process approximation for sequence repeats, and sequencing by hybridization. *Journal of Computational Biology*, 3(3):425–463, 1996. doi:10.1089/CMB.1996.3.425.

3 L. Babai, P. Erdős, and S. M. Selkow. Random graph isomorphism. *SIAM Journal on Computing*, 9(3):628–635, 1980. doi:10.1137/0209047.

4 P. Bastide and C. Groenland. Optimal distance query reconstruction for graphs without long induced cycles. *arXiv preprint arXiv:2306.05979*, 2023.

5 Z. Beerliova, F. Eberhard, T. Erlebach, A. Hall, M. Hoffmann, M. Mihal'ak, and L. S. Ram. Network discovery and verification. *IEEE Journal on selected areas in communications*, 24(12):2168–2181, 2006. doi:10.1109/JSAC.2006.884015.

6 B. Bollobás. The distribution of the maximum degree of a random graph. *Discrete Mathematics*, 32:201–203, 1980. doi:10.1016/0012-365X(80)90054-0.

7 B. Bollobás. The diameter of random graphs. *Transactions of the American Mathematical Society*, 267:41–52, 1981.

8 B. Bollobás. Almost every graph has reconstruction number three. *Journal of Graph Theory*, 14(1):1–4, 1990. doi:10.1002/JGT.3190140102.

9 B. Bollobás. *Random Graphs*. Cambridge University Press, 2 edition, 2001.

10 B. Bollobás, D. Mitsche, and P. Prałat. Metric dimension for random graphs. *Electronic Journal of Combinatorics*, 20(4):P1, 2013.

11 R. Brodal, G. S. Fagerberg, C. N. S. Pedersen, and A. Östlin. The complexity of constructing evolutionary trees using experiments. *BRICS Report Series*, 8(1), 2001.

12 Y. Demidovich, Y. Panichkin, and M. Zhukovskii. Reconstruction of graph colourings. *arXiv preprint*, 2023. arXiv:2308.01671.

13 M. Dyer, A. Frieze, and S. Suen. The probability of unique solutions of sequencing by hybridization. *Journal of Computational Biology*, 1(2):105–110, 1994. doi:10.1089/CMB.1994.1.105.

14 T. Erlebach, A. Hall, M. Hoffmann, and M. Mihal'ák. Network discovery and verification with distance queries. *Algorithms and Complexity*, pages 69–80, 2006.

15 T. Erlebach, A. Hall, and M. Mihal'ák. Approximate discovery of random graphs. In *Stochastic Algorithms: Foundations and Applications, SAGA 2007*, volume 4665 of *Lecture Notes in Computer Science*. Springer, 2007. doi:10.1007/978-3-540-74871-7_8.

16 A. Frieze and M. Karoński. *Introduction to Random Graphs*. Cambridge University Press, 2015.

17 T. E. Harris. A lower bound for the critical probability in a certain percolation process. *Mathematical Proceedings of the Cambridge Philosophical Society*, 56:13–20, 1960.

18 J. J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of mathematical biology*, 51(5):597–603, 1989.

19 H. Huang and K. Tikhomirov. Shotgun assembly of unlabeled Erdős–Rényi graphs. *Probab. Theory Relat. Fields*, 192:575–624, 2025. doi:10.1007/s00440-024-01347-4.

20 S. Janson, T. Łuczak, and A. Ruciński. *Random Graphs*. J. Wiley & Sons, 2000.

21 T. Johnston, G. Kronenberg, A. Roberts, and A. Scott. Shotgun assembly of random graphs. *Probability Theory and Related Fields*, 2025. doi:10.1007/s00440-025-01380-x.

22 S. Kannan, C. Mathieu, and H. Zhou. Graph reconstruction and verification. *ACM Transactions on Algorithms*, 14(4):1–30, 2018. doi:10.1145/3199606.

23 S. K. Kannan, E. L. Lawler, and T. J. Warnow. Determining the evolutionary tree using experiments. *Journal of Algorithms*, 21(1):26–50, 1996. doi:10.1006/JAGM.1996.0035.

24 P. J. Kelly. A congruence theorem for trees. *Pacific Journal of Mathematics*, 7:961–968, 1957.

25 V. King, L. Zhang, and Y. Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 444–453, 2003.

26 L. Kučera. Canonical labeling of regular graphs in linear average time. In *Proceedings of 28th Annual Symposium on Foundations of Computer Science (FOCS'87)*, pages 271–279, 1987.

27 A. Lingas, H. Olsson, and A. Östlin. Efficient merging, construction, and maintenance of evolutionary trees. In *Automata, Languages and Programming*, pages 544–553, 1999.

28 N. Linial and J. Mosheiff. On the rigidity of sparse random graphs. *Journal of Graph Theory*, 85(2):466–480, 2017. doi:10.1002/JGT.22073.

29 G. D. Marmerola, M. A. Oikawa, Z. Dias, S. Goldenstein, and A. Rocha. On the reconstruction of text phylogeny trees: evaluation and analysis of textual relationships. *PloS one*, 11(12):e0167822, 2016.

30 C. Mathieu and H. Zhou. Graph reconstruction via distance oracles. In *International Colloquium on Automata, Languages and Programming (ICALP'13)*, pages 733–744, 2013.

31 C. Mathieu and H. Zhou. A simple algorithm for graph reconstruction. In *29th Annual European Symposium on Algorithms (ESA'21)*, pages 68:1–68:18, 2021. doi:10.4230/LIPIcs.ESA.2021.68.

32 C. McDiarmid. On the method of bounded differences. In *Surveys in Combinatorics: Invited Papers at the Twelfth British Combinatorial Conference*, London Mathematical Society Lecture Note Series, pages 148–188. Cambridge University Press, 1989.

33 E. Mossel and N. Ross. Shotgun assembly of labelled graphs. *IEEE Transactions on Network Science and Engineering*, 6(2):145–157, 2019. doi:10.1109/TNSE.2017.2776913.

34 A. S. Motahari, G. Bresler, and D. N. Tse. Information theory of dna shotgun sequencing. *IEEE Transactions on Information Theory*, 59(10):6273–6289, 2013. doi:10.1109/TIT.2013.2270273.

35 V. Müller. Probabilistic reconstruction from subgraphs. *Commentationes Mathematicae Universitatis Carolinae*, 17(4):709–719, 1976.

36 G. Odor. *The role of adaptivity in source identification with time queries*. PhD thesis, EPFL, 2022.

37 L. Reyzin and N. Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Algorithmic Learning Theory*, pages 285–297. Springer, 2007. doi:10.1007/978-3-540-75225-7_24.

38 S. Sen and V. N. Muralidhara. The covert set-cover problem with application to network discovery. In *International Conference and Workshops on Algorithms and Computation (WALCOM'10)*, pages 228–239, 2010.

39 D. Soudry, S. Keshri, P. Stinson, M.-h. Oh, G. Iyengar, and L. Paninski. Efficient "shotgun" inference of neural connectivity from highly sub-sampled activity data. *PLoS Computational Biology*, 11(12):e1004657, 2015.

40 J. Spencer. Counting extensions. *Journal of Combinatorial Theory, Series A*, 55:247–255, 1990. doi:10.1016/0097-3165(90)90070-D.

41 H. Spinoza and D. B. West. Reconstruction from the deck of k-vertex induced subgraphs. *Journal of Graph Theory*, 90:497–522, 2019. doi:10.1002/JGT.22409.

42 S. M. Ulam. *A collection of mathematical problems*. Number 8 in Interscience Tracts in Pure and Applied Mathematics. Interscience Publishers, New York–London, 1960.

43 M. S. Waterman, T. F. Smith, M. Singh, and W. A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64(2):199–213, 1977.

An Optimal Algorithm for Shortest Paths in Unweighted Disk Graphs

Bruce W. Brewer
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Haitao Wang
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

Given in the plane a set S of n points and a set of disks centered at these points, the disk graph $G(S)$ induced by these disks has vertex set S and an edge between two vertices if their disks intersect. Note that the disks may have different radii. We consider the problem of computing shortest paths from a source point $s \in S$ to all vertices in $G(S)$ where the length of a path in $G(S)$ is defined as the number of edges in the path. The previously best algorithm solves the problem in $O(n \log^2 n)$ time. A lower bound of $\Omega(n \log n)$ is also known for this problem under the algebraic decision tree model. In this paper, we present an $O(n \log n)$ time algorithm, which matches the lower bound and thus is optimal. Another virtue of our algorithm is that it is quite simple.

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

Keywords and phrases disk graphs, weighted Voronoi diagrams, shortest paths

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.31

1 Introduction

Let S be a set of n points in the plane. Each point of S is associated with a disk centered at the point. Note that the disks may have different radii. The *disk graph* $G(S)$ induced by these disks has vertex set S and an edge between two vertices if their disks intersect. We consider the single-source-shortest-path (SSSP) problem of computing shortest paths from a source point $s \in S$ to all vertices in $G(S)$ where the length of a path in $G(S)$ is defined as the number of edges in the path.

The problem was studied previously. A straightforward solution is to first construct $G(S)$ explicitly and then run a breadth-first-search algorithm, which takes $\Omega(n^2)$ time since $G(S)$ may have $\Omega(n^2)$ edges in the worst case. Kaplan, Katz, Saban, and Sharir [15] proposed the first subquadratic time algorithm, and their algorithm runs in $O(n \log^4 n)$ time. Later Klost [18] gave an $O(n \log^2 n)$ time improved solution. On the other hand, it is known that the problem has an $\Omega(n \log n)$ time lower bound under the algebraic decision tree model even if all disks have the same radius [6].

In this paper, we present a new algorithm whose runtime is $O(n \log n)$, which matches the $\Omega(n \log n)$ lower bound and thus is optimal. Another virtue of the algorithm is that it is quite simple. Indeed, the most complex step of our algorithm is constructing a static additively-weighted Voronoi diagram, which can be easily done using Fortune's sweeping algorithm [14]. Aside from that, the algorithm's description in the paper is self-contained.

Concurrently with our work, de Berg and Cabello have also achieved an $O(n \log n)$ time algorithm using a different approach [4].

Related work. Disk graphs are among the most well-studied topics in computational geometry due to their applications and geometric properties, e.g., [1, 3, 7, 8, 10, 13, 20, 24]. The SSSP problem we consider is actually an *unweighted case* because the length of each edge of


© Bruce W. Brewer and Haitao Wang;
licensed under Creative Commons License CC-BY 4.0

$G(S)$ is defined to be one. In the *weighted* case, the length of each edge is defined to be the Euclidean distance between the two disk centers corresponding to the two vertices of the edge. Kaplan, Katz, Saban, and Sharir [15] also gave an algorithm for the weighted case, and the algorithm runs in $O(n \log^6 n)$ time. An, Oh, and Xue [2] very recently derived an improved algorithm of $O(n \log^4 n)$ time.

The SSSP problem on *unit-disk graphs* (in which all disks have the same radius) has also been extensively studied. Cabello and Jejčič [6] first solved the unweighted case in $O(n \log n)$ time, matching the $\Omega(n \log n)$ lower bound [6]. If the points are presorted by x- and y-coordinates, Chan and Skrepetos [9] showed that it can be solved in $O(n)$ additional time. The weighted case was first solved by Roditty and Segal [22] in $O(n^{4/3+\epsilon})$ time for any $\epsilon > 0$. Later, Cabello and Jejčič [6] solved it in $O(n^{1+\epsilon})$ time. The algorithm by Cabello and Jejčič [6] is bottlenecked by a dynamic bicromatic closest pair data structure, which has since seen improvements by Kaplan, Mulzer, Roditty, Seiferth, and Sharir [16] and also by Liu [19]. Recently, Wang and Xue [25] gave a new algorithm of $O(n \log^2 n)$ time. This has been slightly reduced to $O(n \log^2 n / \log \log n)$ by Brewer and Wang [5]. Furthermore, Brewer and Wang [5] proved that the problem can be solved using $O(n \log n)$ comparisons under the algebraic decision tree model, matching an $\Omega(n \log n)$ lower bound in the same model [6].

Kaplan, Katz, Saban, and Sharir [15] also studied a "reversed version" of the shortest path problem in disk graphs. Chan and Huang [8] very recently improved the algorithm of [15]. Both algorithms used the previous SSSP algorithm for disk graphs as a decision procedure. With our new SSSP algorithm, these algorithms can be slightly improved and also simplified.

In the following, after introducing some notation and concepts in Section 2, we present our algorithm in Section 3.

2 Preliminaries

We follow the notation already defined in Section 1, e.g., n, S, s, $G(S)$.

We define S_i for $i = 0, 1, 2, \ldots$ to be the set of vertices of $G(S)$ whose distance from s in $G(S)$ is equal to i. Note that $S_0 = \{s\}$. For convenience, we define $S_{\leq i} = \cup_{j \leq i} S_j$.

For any two points p, q in the plane, let $|pq|$ denote the (Euclidean) distance between p and q.

To differentiate from other points in the plane, we refer to points of S as *sites*. For any site $v \in S$, let D_v denote the input disk centered at v and r_v the radius of D_v. Notice that two sites $u, v \in S$ have an edge (u, v) in $G(S)$ if and only if $|uv| - r_u - r_v \leq 0$. We consider r_v as the *weight* of v, and for any point $p \in \mathbb{R}^2$, we define the *(additively) weighted distance* from p to v as $d_v(p) = |vp| - r_v$. Note that if p is outside the disk D_v, then $d_v(p)$ is the smallest distance from p to any point of D_v.

For any subset $S' \subseteq S$ and a point $p \in \mathbb{R}^2$, we define a *nearest neighbor* of p to be a site in S' whose weighted distance to p is minimal. Formally, we define $\beta_p(S') = \arg\min_{v \in S'} d_v(p)$ to be a nearest neighbor of p in S'. In cases where p has more than one nearest neighbor in S', we let $\beta_p(S')$ refer to an arbitrary one. We also define $d_{S'}(p) = \min_{v \in S'} d_v(p)$.

With respect to the weights r_v of the sites v of S, we will make heavy use of the *(additively) weighted Voronoi diagram* of S, which is a partition of the plane into Voronoi regions, edges, and vertices [14, 23]. The Voronoi region $\mathrm{Vor}(u)$ of a site $u \in S$ is usually defined as the set of points nearer to u than to any other site in S. For convenience, we will let $\mathrm{Vor}(u)$ also include its boundary, defining $\mathrm{Vor}(u)$ as the set of points at least as near to u than to any other site in S. Formally, $\mathrm{Vor}(u) = \{p \in \mathbb{R}^2 : \forall\, v \in S \setminus \{u\},\ d_u(p) \leq d_v(p)\}$. This way, the

plane is covered by the Voronoi regions. The Voronoi edge between two sites $u, v \in S$ is defined by $\mathcal{E}(u,v) = \{p \in \mathbb{R} : \forall\, w \in S \setminus \{u,v\}, d_u(p) = d_v(p) < d_w(p)\}$. Note that $\mathcal{E}(u,v)$ may have multiple connected components, each of which is a straight line segment or a hyperbolic arc [14, 23]. Voronoi vertices are the points that are equidistant to at least three sites in S such that all other sites in S are farther than these. We use $\mathcal{VD}(S)$ to denote the weighted Voronoi diagram of S.

Let $\mathcal{DT}(S)$ denote the dual graph of $\mathcal{VD}(S)$. Specifically, S is the vertex set of $\mathcal{DT}(S)$, and the edge set of $\mathcal{DT}(S)$ consists of all pairs of sites $u, v \in S$ whose Voronoi regions share a Voronoi edge, i.e., $\mathcal{E}(u,v) \neq \emptyset$. Note that if the sites of S all have the same weight (i.e., all disks have the same radius), then $\mathcal{DT}(S)$ is the Delaunay triangulation of S. We define the neighborhood $N_{\mathcal{DT}}(u)$ of a site $u \in S$ to be the set of sites that share an edge with u in $\mathcal{DT}(S)$. By extension, we define the neighborhood $N_{\mathcal{DT}}(S')$ of a set $S' \subseteq S$ by $N_{\mathcal{DT}}(S') = \bigcup_{u \in S'} N_{\mathcal{DT}}(u)$.

For ease of exposition, we make the general position assumption that no three sites of S lie on the same line. The assumption can be lifted without affecting the results, e.g., by standard perturbation techniques [12, 26].

3 The algorithm

In this section, we present our algorithm to compute the sets S_i for $i = 0, 1, 2, \ldots$. It is easy to extend our algorithm to also compute predecessor information for the shortest paths.

We follow the same algorithmic scheme of Cabello and Jejčič [6] for unit-disk graphs and manage to extend their algorithm to our general disk graph case. In what follows, we first describe the algorithm, then discuss the implementation and time analysis, and finally prove its correctness.

3.1 Algorithm description

We begin with constructing the weighted Voronoi diagram $\mathcal{VD}(S)$. Then the algorithm runs in iterations, and each i-th iteration will compute the set S_i. Initially, we have $S_0 = \{s\}$. In the i-th iteration for $i \geq 1$, we compute S_i as follows, assuming that $S_{\leq i-1}$ has already been computed. Let $Q = N_{\mathcal{DT}}(S_{i-1}) \setminus S_{\leq i-1}$, i.e., the neighbors of S_{i-1} in $\mathcal{DT}(S)$ that are not in S_{i-1}. We remove a site $v \in Q$ from Q and determine whether $d_{S_{i-1}}(v) \leq r_v$. If it is, then we add v to S_i and update $Q = Q \cup (N_{\mathcal{DT}}(v) \setminus S_{\leq i})$, i.e., we add the neighbors of v in $\mathcal{DT}(S)$ that are not in $Q \cup S_{\leq i}$ to Q. We repeat this until $Q = \emptyset$, at which point the i-th iteration is complete. After the i-th iteration, we check if $S_i = \emptyset$. If so, then we are done, and the algorithm stops. Otherwise, we continue to the $i+1$-th iteration. Algorithm 1 gives the pseudocode.

3.2 Algorithm implementation and time analysis

We now discuss how to implement the algorithm efficiently.

First of all, computing $\mathcal{VD}(S)$ and thus $\mathcal{DT}(S)$ can be done in $O(n \log n)$ time [14]. Note that the combinatorial sizes of both $\mathcal{VD}(S)$ and $\mathcal{DT}(S)$ are $O(n)$ [14, 23].

Next, notice that the number of vertices added to Q on Line 7 in iteration i is at most the sum of the degrees of the vertices of S_{i-1} in the graph $\mathcal{DT}(S)$. Because the sets S_i are disjoint and $\mathcal{DT}(S)$ has $O(n)$ edges, this means that the number of vertices added to Q by Line 7 throughout the whole algorithm is $O(n)$. By the same reasoning, we conclude that

Algorithm 1 SSSP algorithm.

Input: A set of point sites S, radius r_v for each site $v \in S$, source site $s \in S$.
Output: The sets S_i for $i = 0, 1, \ldots$

```
1  i ← 0;
2  S_0 ← {s};
3  Build VD(S) and DT(S);
4  while S_i ≠ ∅ do
5  |  i ← i + 1;
6  |  S_i ← ∅;
7  |  Q ← N_DT(S_{i−1}) \ S_{≤i−1} ;
8  |  while Q ≠ ∅ do
9  |  |  v ← pop(Q);
10 |  |  if d_{S_{i−1}}(v) ≤ r_v then
11 |  |  |  S_i ← S_i ∪ {v};
12 |  |  |  Q ← Q ∪ (N_DT(v) \ S_{≤i}) ;
13 return S_i for i = 0, 1, ...;
```

the number of vertices added to Q by Line 12 throughout the whole algorithm is also $O(n)$. We can also know that the while loop on Line 8 has at most $O(n)$ iterations throughout the whole algorithm because a vertex is removed from Q in each iteration.

It remains to determine the time complexity for the operation of checking whether $d_{S_{i-1}}(v) \leq r_v$ on Line 10. To this end, it suffices to find $\beta_v(S_{i-1})$, i.e., the nearest neighbor of v in S_{i-1}. To find $\beta_v(S_{i-1})$, in the beginning of the i-th iteration, we compute the weighted Voronoi diagram for S_{i-1}, denoted by $\mathcal{VD}(S_{i-1})$, which takes $O(|S_{i-1}| \log |S_{i-1}|)$ time [14], and then construct a point location data structure for $\mathcal{VD}(S_{i-1})$ in $O(|S_{i-1}|)$ time [11,17]. After that, $\beta_v(S_{i-1})$ can be found in $O(\log n)$ time by a point location query with v on $\mathcal{VD}(S_{i-1})$. As such, the total time of Line 10 in the whole algorithm is $O(n \log n)$. As $\sum_i |S_{i-1}| = n$, the overall time for constructing $\mathcal{VD}(S_{i-1})$ in the whole algorithm is $O(n \log n)$.

We thus conclude that the time complexity of the whole algorithm is $O(n \log n)$.

3.3 Algorithm correctness

We start with proving the following two observations that our algorithm's correctness relies on.

▶ **Observation 1.** *For any two sites $u, v \in S$, they are connected by an edge in $G(S)$ if and only if $d_v(u) \leq r_u$ or $d_u(v) \leq r_v$.*

Proof. Recall that u and v have an edge if and only if $|uv| - r_u - r_v \leq 0$. Notice that $|uv| - r_u - r_v \leq 0$ holds if and only if $d_v(u) \leq r_u$ or $d_u(v) \leq r_v$. ◀

▶ **Observation 2.** *For any site $v \in S \setminus S_{\leq i-1}$, v is in S_i if and only if $d_{S_{i-1}}(v) \leq r_v$.*

Proof. By definition, S_{i-1} has a site u with $d_u(v) = d_{S_{i-1}}(v)$. If $d_{S_{i-1}}(v) \leq r_v$, then $d_u(v) \leq r_v$. By Observation 1, $G(S)$ has an edge connecting u and v. As $u \in S_{i-1}$ and $v \in S \setminus S_{\leq i-1}$, we obtain that $v \in S_i$.

If $v \in S_i$, then there is an edge in $G(S)$ connecting v to a site $u \in S_{i-1}$. By Observation 1, $d_u(v) \leq r_v$. It then follows that $d_{S_{i-1}}(v) \leq d_u(v) \leq r_v$. ◀

We now prove that the algorithm is correct. Let S'_i be the set S_i computed by our algorithm and let S_i be the "correct" set of vertices at distance i from s in $G(S)$. Then, our goal is to prove that $S'_i = S_i$ for all i. We do so by induction on i. Our base case is that $S'_0 = S_0$, which is obviously true because they are both $\{s\}$. Our inductive hypothesis is that $S'_j = S_j$ for all $j < i$. To prove that $S'_i = S_i$, we will argue both $S'_i \subseteq S_i$ and $S'_i \supseteq S_i$.

First of all, since $S'_j = S_j$ for all $j < i$, our algorithm guarantees that every site of S'_i is at distance i from s in $G(S)$ and thus $S'_i \subseteq S_i$. To see this, a site v is added to S'_i in Line 11 because $d_{S_{i-1}}(v) \leq r_v$. By Observation 2, $v \in S_i$.

We next prove that $S'_i \supseteq S_i$. Consider a site $v \in S_i$. Our goal is to show that v is also in S'_i. We will prove in Lemma 3 that there exists a vertex $u \in S_{i-1}$ and a u-v path π in $\mathcal{DT}(S)$ such that all internal vertices of π are in S_i (note that *internal vertices* refer to vertices of π excluding u and v). Let the vertices of π be $u = w_0, w_1, \ldots, w_k = v$. We argue that all w_j with $1 \leq j \leq k$ are added to S'_i. To this end, because $w_j \in S_i$ for all $1 \leq j \leq k$, it is sufficient to argue that w_j is added to Q because when w_j is popped from Q, it will pass the check on Line 10 by Observation 2 and then be added to S'_i. Indeed, w_1 is added to Q on Line 7 because $w_1 \in N_{\mathcal{DT}}(u)$. Since $w_2 \in N_{\mathcal{DT}}(w_1)$, w_2 will be added to Q on Line 12 no later than right after w_1 is added to S'_i on Line 11. Following the same argument, $w_3, w_4, \ldots, w_k = v$ will all be added to Q and thus added to S'_i as well. This proves that $v \in S'_i$. Therefore, $S'_i \supseteq S_i$.

In summary, we have proved $S'_i = S_i$ and thus the correctness of the algorithm.

▶ **Lemma 3.** *For any $i > 0$ and any site $v \in S_i$, there exist a site $u \in S_{i-1}$ and a u-v path in $\mathcal{DT}(S)$ such that all internal vertices of the path are in S_i.*

Proof. We extend the proof of [6, Lemma 1] for the unit-disk graph case to our general disk graph problem.

Let $u = \beta_v(S_{i-1})$, i.e., u is a nearest neighbor of v in S_{i-1}. Consider the line segment $\overline{uv}$ and the sites whose Voronoi regions in $\mathcal{VD}(S)$ intersect $\overline{uv}$. Let $w_0, w_1, \ldots, w_k$ be these sites in the order that their Voronoi regions intersect $\overline{uv}$ with $w_0 = u$ and $w_k = v$. Note that repetitions are possible, so it could be that $w_j = w_{j'}$ for $j \neq j'$. See Figure 1 for an example. For ease of discussion, we assume that $\overline{uv}$ does not contain any Voronoi vertices of $\mathcal{VD}(S)$ (otherwise, we could replace v by a point arbitrarily close to v and then follow the same argument). This implies that, for every $0 \leq j \leq k-1$, $\overline{uv}$ crosses the Voronoi edge between w_j and w_{j+1}. This further implies that w_j and w_{j+1} are adjacent in $\mathcal{DT}(S)$, so $\pi = w_0, w_1, \ldots, w_k$ is a u-v path in $\mathcal{DT}(S)$.

Next, we need to show that all internal vertices w_j $(1 \leq j \leq k-1)$ of π are in S_i. We will first prove that there is an edge (u, w_j) in $G(S)$ connecting u and w_j, meaning that $w_j \in S_{i-2} \cup S_{i-1} \cup S_i$ since $u \in S_{i-1}$. Then, we will show that $w_j \notin S_{i-2} \cup S_{i-1}$, so $w_j \in S_i$ must hold. This will complete the proof of the lemma.

Proving $G(S)$ has an edge (u, w_j). We first argue that $G(S)$ has an edge (u, v). Indeed, since $v \in S_i$, there is a vertex $u' \in S_{i-1}$ connecting to v by an edge in $G(S)$. By Observation 1, $d_{u'}(v) \leq r_v$. Since $u = \beta_v(S_{i-1})$, we have $d_u(v) \leq d_{u'}(v) \leq r_v$. By Observation 1, $G(S)$ has an edge (u, v).

Consider any w_j with $1 \leq j \leq k-1$. We now argue that $G(S)$ has an edge (u, w_j). Recall that $\overline{uv}$ intersects $\mathrm{Vor}(w_j)$. Let p be a point of $\overline{uv} \cap \mathrm{Vor}(w_j)$; see Figure 2. We can derive the following:

$$d_{w_j}(u) = |w_j u| - r_{w_j} \overset{(1)}{\leq} |w_j p| + |pu| - r_{w_j} \overset{(2)}{\leq} |vp| + |pu| - r_v \overset{(3)}{=} |vu| - r_v = d_v(u) \overset{(4)}{\leq} r_u,$$

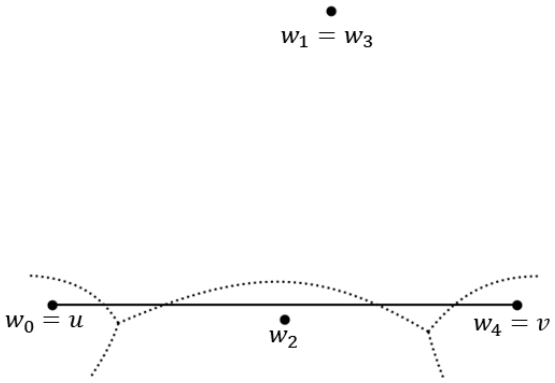


Figure 1 The segment $\overline{uv}$ passes through $\mathrm{Vor}(u)$, $\mathrm{Vor}(w_1)$, $\mathrm{Vor}(w_2)$, $\mathrm{Vor}(w_3)$, and $\mathrm{Vor}(v)$, where $w_1 = w_3$.

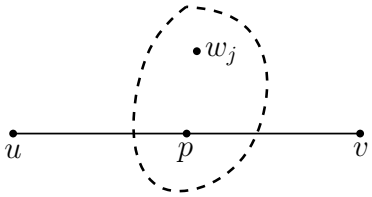


Figure 2 The region bounded by the dashed curve is $\mathrm{Vor}(w_j)$.

where (1) is due to the triangle inequality, (2) is due to $p \in \mathrm{Vor}(w_j)$ (therefore, $|w_jp| - r_{w_j} = d_{w_j}(p) \leq d_v(p) = |vp| - r_v$), (3) is due to $p \in \overline{uv}$, and (4) is due to Observation 1 and the fact that $G(S)$ has an edge (u, v). Consequently, by Observation 1, $G(S)$ has an edge (u, w_j).

Proving $w_j \notin S_{i-2} \cup S_{i-1}$. To prove that $w_j \notin S_{i-2} \cup S_{i-1}$, $1 \leq j \leq k-1$, as before, we pick a point $p \in \overline{uv} \cap \mathrm{Vor}(w_j)$ and derive the following:

$$d_{w_j}(v) \overset{(1)}{<} |w_jp| + |pv| - r_{w_j} \overset{(2)}{\leq} |up| + |pv| - r_u \overset{(3)}{=} |uv| - r_u = d_u(v) \overset{(4)}{\leq} r_v,$$

where (1) is due to the triangle inequality and our general position assumption (because $p \in \overline{uv}$, equality occurs only if u, v, and w_j are collinear, violating our general position assumption), (2) is due to $p \in \mathrm{Vor}(w_j)$ (therefore, $|w_jp| - r_{w_j} = d_{w_j}(p) \leq d_u(p) = |up| - r_u$), (3) is due to $p \in \overline{uv}$, and (4) is due to Observation 1 and the fact that $G(S)$ has an edge (u, v), as proven above.

Notice that $w_j \notin S_{i-1}$ since otherwise $d_{w_j}(v) < d_u(v)$ proved above would contradict with $u = \beta_v(S_{i-1})$. Also, since $G(S)$ has an edge (w_j, v) and $v \in S_i$, w_j cannot be in S_{i-2} (since otherwise v would be in $S_{\leq i-1}$, contradicting with that $v \in S_i$). This proves that $w_j \notin S_{i-2} \cup S_{i-1}$. ◀

The following theorem summarizes our result.

▶ **Theorem 4.** *Given a set of n disks in the plane and a source disk, shortest paths from the source to all other vertices in the disk graph induced by all disks can be computed in $O(n \log n)$ time.*

Extension to the L_∞ (resp., L_1) metric. In the L_∞ metric, each disk becomes a square centered at a site of S. For this case, an $O(n \log n)$-time algorithm was given by Klost [18] to solve the SSSP problem, which uses the involved techniques in [3]. We can solve this problem

in $O(n \log n)$ time in a much simpler way by making the following changes to Algorithm 1. First, use the L_∞ metric to measure the distance $|pq|$ of any two points p, q in the plane. Second, use additively-weighted Voronoi diagrams in the L_∞ metric instead. Note that constructing an additively-weighted Voronoi diagram for a set of n points in the L_∞ metric is equivalent to constructing a Voronoi diagram for a set of n squares in the L_∞ metric. This latter problem can be solved in $O(n \log n)$ time, e.g., by the algorithm of Papadopoulou and Lee [21]. As such, Algorithm 1 can be modified to solve the L_∞ metric case in $O(n \log n)$ time. The algorithm is much simpler than the one in [18]. The L_1 case can be treated analogously.

References

1. Shinwoo An, Kyungjin Cho, and Eunjin Oh. Faster algorithms for cycle hitting problems on disk graphs. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS)*, pages 29–42, 2023. doi:10.1007/978-3-031-38906-1_3.
2. Shinwoo An, Eunjin Oh, and Jie Xue. Single-source shortest path problem in weighted disk graphs. In *Proceedings of the 41st International Symposium on Computational Geometry (SoCG)*, pages 7:1–7:15, 2025. doi:10.4230/LIPIcs.SoCG.2025.7.
3. Alexander Baumann, Haim Kaplan, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Dynamic connectivity in disk graphs. *Discrete and Computational Geometry*, 71:214–277, 2024. doi:10.1007/s00454-023-00621-x.
4. Mark de Berg and Sergio Cabello. An $O(n \log n)$ algorithm for single-source shortest paths in disk graphs. In *Proceedings of the 33rd Annual European Symposium on Algorithms (ESA)*, page to appear, 2025. doi:10.48550/arXiv.2506.07571.
5. Bruce W. Brewer and Haitao Wang. An improved algorithm for shortest paths in weighted unit-disk graphs. In *Proceedings of the 36th Canadian Conference on Computational Geometry (CCCG)*, pages 57–64, 2024. arXiv:2407.03176.
6. Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry: Theory and Applications*, 48:360–367, 2015. doi:10.1016/j.comgeo.2014.12.003.
7. Sergio Cabello and Wolfgang Mulzer. Minimum cuts in geometric intersection graphs. *Computational Geometry: Theory and Applications*, 94(101720), 2021. doi:10.1016/j.comgeo.2020.101720.
8. Timothy M. Chan and Zhengcheng Huang. Faster algorithms for reverse shortest path in unit-disk graphs and related geometric optimization problems: Improving the shrink-and-bifurcate technique. In *Proceedings of the 41st International Symposium on Computational Geometry (SoCG)*, pages 32:1–32:14, 2025. doi:10.4230/LIPIcs.SoCG.2025.32.
9. Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016. doi:10.4230/LIPIcs.ISAAC.2016.24.
10. Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86:165–177, 1990. doi:10.1016/0012-365X(90)90358-O.
11. Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986. doi:10.1137/0215023.
12. Herbert Edelsbrunner and Ernst P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9:66–104, 1990. doi:10.1145/77635.77639.
13. Jared Espenant, J. Mark Keil, and Debajyoti Mondal. Finding a maximum clique in a disk graph. In *Proceedings of the 39th International Symposium on Computational Geometry (SoCG)*, pages 30:1–30:17, 2023. doi:10.4230/LIPIcs.SoCG.2022.49.

14 Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987. doi:10.1007/BF01840357.

15 Haim Kaplan, Matthew J. Katz, Rachel Saban, and Micha Sharir. The unweighted and weighted reverse shortest path problem for disk graphs. In *Proceedings of the 31st Annual European Symposium on Algorithms (ESA)*, pages 67:1–67:14, 2023. doi:10.4230/LIPIcs.ESA.2023.67.

16 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discrete and Computational Geometry*, 64:838–904, 2020. doi:10.1007/s00454-020-00243-7.

17 David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12:28–35, 1983. doi:10.1137/0212002.

18 Katharina Klost. An algorithmic framework for the single source shortest path problem with applications to disk graphs. *Computational Geometry: Theory and Applications*, 111:101979, 2023. doi:10.1016/j.comgeo.2022.101979.

19 Chih-Hung Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. *SIAM Journal on Computing*, 51:723–765, 2022. doi:10.1137/20M1388371.

20 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. Subexponential parameterized algorithms on disk graphs (extended abstract). In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2005–2031, 2022. doi:10.1137/1.9781611977073.80.

21 Evanthia Papadopoulou and D.T. Lee. The L_∞-Voronoi diagram of segments and VLSI applications. *International Journal of Computational Geometry and Application*, 11(101720):503–528, 2001. doi:10.1142/S0218195901000626.

22 Liam Roditty and Michael Segal. On bounded leg shortest paths problems. *Algorithmica*, 59:583–600, 2011. doi:10.1007/s00453-009-9322-3.

23 Micha Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM Journal on Computing*, 14:448–468, 1985. doi:10.1137/0214034.

24 Anastasiia Tkachenko and Haitao Wang. Dominating set, independent set, discrete k-center, dispersion, and related problems for planar points in convex position. In *Proceedings of the 42nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 73:1–73:20, 2025. doi:10.4230/LIPIcs.STACS.2025.73.

25 Haitao Wang and Jie Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete and Computational Geometry*, 64:1141–1166, 2020. doi:10.1007/s00454-020-00219-7.

26 C.-K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *Journal of Computer and System Sciences*, 40:2–18, 1990. doi:10.1016/0022-0000(90)90016-E.

Linear-Time Multilevel Graph Partitioning via Edge Sparsification

Lars Gottesbüren
Google Research, Zürich, Switzerland

Nikolai Maas
Karlsruhe Institute of Technology, Germany

Dominik Rosch
Karlsruhe Institute of Technology, Germany

Peter Sanders
Karlsruhe Institute of Technology, Germany

Daniel Seemaier
Karlsruhe Institute of Technology, Germany

Abstract

The current landscape of balanced graph partitioning is divided into high-quality but expensive multilevel algorithms and cheaper approaches with linear running time, such as single-level algorithms and streaming algorithms. We demonstrate how to achieve the best of both worlds with a *linear time multilevel algorithm*. Multilevel algorithms construct a hierarchy of increasingly smaller graphs by repeatedly contracting clusters of nodes. Our approach preserves their distinct advantage, allowing refinement of the partition over multiple levels with increasing detail. At the same time, we use *edge sparsification* to guarantee geometric size reduction between the levels and thus linear running time.

We provide a proof of the linear running time as well as additional insights into the behavior of multilevel algorithms, showing that graphs with low modularity are most likely to trigger worst-case running time. We evaluate multiple approaches for edge sparsification and integrate our algorithm into the state-of-the-art multilevel partitioner KaMinPar, maintaining its excellent parallel scalability. As demonstrated in detailed experiments, this results in a 1.49× average speedup (up to 4× for some instances) with only 1% loss in solution quality. Moreover, our algorithm clearly outperforms state-of-the-art single-level and streaming approaches.

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

Keywords and phrases Graph Partitioning, Graph Algorithms, Linear Time Algorithms, Graph Sparsification

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.32

Related Version *Full Version*: `https://arxiv.org/abs/2504.17615` [30]

Supplementary Material *Software (Source Code)*: `https://github.com/KaHIP/KaMinPar/commit/73eeaa2371c6826e166c9ca1383996f14d8c7a2a` [27]
archived at `swh:1:dir:2ec38dbccb676136f71dc1796d6a06b450c48c6d`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882500).

1 Introduction

Balanced graph partitioning aims to divide a graph into blocks of roughly equal size while minimizing the number of edges cut by the partition. As this is a crucial subtask in many applications [8, 14], it is of considerable interest to compute high-quality partitions within a minimal amount of time. Unfortunately, this goal seems unattainable from the viewpoint of

© Lars Gottesbüren, Nikolai Maas, Dominik Rosch, Peter Sanders, and Daniel Seemaier;
licensed under Creative Commons License CC-BY 4.0

complexity theory – even approximating balanced graph partitioning to a constant factor is NP-hard [4]. Consequently, heuristic approaches are used in practice, covering a wide spectrum of options along the running time versus quality trade-off.

In the high-quality category, the most successful approaches use the multilevel framework. By repeatedly contracting clusters of nodes, multilevel algorithms first construct a hierarchy of increasingly smaller graphs in the *coarsening phase*. On the smallest graph, more expensive heuristics can be used to find a good *initial partition*. Finally, the *uncoarsening phase* undoes the contractions in reverse order while further improving the partition quality via local search algorithms (this is called *refinement*). Overall, multilevel partitioning combines a good initial solution with iterative refinement on a series of summarized graph representations with increasingly finer granularity. This has proven highly successful in practice, consistently achieving better solution quality than alternative approaches on real-world inputs [13, 26, 36]. However, due to lacking constraints on the size of the contracted representations, current multilevel implementations have superlinear running time.

On the other hand, *single-level* algorithms that use only a fixed number of passes on the input graph can run in linear time [50]. This is motivated by applications where the partitioning time is a potential bottleneck. For example, graph partitioning is used in various domains to efficiently distribute workloads across parallel machines [7, 11, 48]. This requires the graph partitioning step to be less expensive than the downstream computation. Taking this to the extreme, *streaming* approaches only consider a small part of the graph at once, assigning nodes greedily while using only a minimal representation of the partition state [17, 21, 31]. However, the running time guarantees of single-level and streaming algorithms come at the cost of inferior solution quality when compared to multilevel algorithms [6, 26, 53].

Contributions. In this work, we show that the described trade-off can be avoided by constructing a *linear time multilevel algorithm*. Our coarsening algorithm enforces that the graph shrinks by a constant factor with every successive contraction step, using edge sparsification to reduce the number of edges if necessary. We prove that this guarantees $\mathcal{O}(n + m)$ expected total work for n nodes and m edges, without any assumptions on the input graph. Our analysis provides a framework to understand the running time behavior of a broad class of existing multilevel algorithms. In addition, we demonstrate that graphs with low modularity are most likely to trigger worst-case running time behavior, while graphs with high modularity might already allow linear running time without using edge sparsification.

We integrate our approach into the KaMinPar shared-memory graph partitioner [29], preserving its excellent scaling behavior while guaranteeing linear work. For instance classes that approximate the worst case, our algorithm achieves practical speedups of up to 4× (1.49× in the geometric mean) over a baseline KaMinPar configuration – which is the fastest available shared-memory multilevel partitioner according to Ref. [26]. Despite this, the loss in partition quality is only 1% on average. Our algorithm outperforms both the single-level partitioner PuLP [50] and the state-of-the-art streaming partitioner CUTTANA [31], achieving 24% and 66% smaller average cuts, respectively, as well as a faster running time.

2 Preliminaries

Notation and Definitions. Let $G = (V, E, c, \omega)$ be an undirected graph with node weights $c : V \to \mathbb{N}_{>0}$, edge weights $\omega : E \to \mathbb{N}_{>0}$, $n := |V|$, and $m := |E|$. We extend c and ω to sets, i.e., $c(V' \subseteq V) := \sum_{v \in V'} c(v)$ and $\omega(E' \subseteq E) := \sum_{e \in E'} \omega(e)$. $N(v) := \{u \mid \{u, v\} \in E\}$ denotes the neighbors of $v \in V$ and $E(v) := \{e \mid v \in e\}$ denotes the edges incident to v. We

are looking for k *blocks* of nodes $\Pi \coloneqq \{V_1, \dots, V_k\}$ that partition V, i.e., $V_1 \cup \dots \cup V_k = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. The *balance constraint* demands that $\forall i \in \{1, \dots, k\}$: $c(V_i) \leq L_{\max} \coloneqq (1+\varepsilon)\lceil \frac{c(V)}{k} \rceil$ for some imbalance parameter $\varepsilon > 0$. The objective is to minimize $\text{cut}(\Pi) \coloneqq \sum_{i<j} \omega(E_{ij})$ (weight of all cut edges), where $E_{ij} \coloneqq \{\{u,v\} \in E \mid u \in V_i, v \in V_j\}$. A *clustering* $\mathcal{C} \coloneqq \{C_1, \dots, C_b\}$ is also a partition of V, where the number of blocks b is not given in advance (there is also no balance constraint).

Multilevel Graph Partitioning. Virtually all high-quality, general-purpose graph partitioners are based on the multilevel paradigm, which consists of three phases. During coarsening, the algorithms construct a hierarchy $\mathcal{H} = \langle G =: G_1, G_2, \dots, G_\ell \rangle$ of successively coarser representations of the input graph G. Coarse graphs are built by either computing node clusterings or matchings and afterwards *contracting* them. A clustering $\mathcal{C} = \{C_1, \dots, C_b\}$ is contracted by replacing each cluster C_i with a coarse node c_i with weight $c(c_i) = c(C_i)$. For each pair of clusters C_i and C_j, there is a coarse edge $e = \{c_i, c_j\}$ with weight $\omega(e) = \omega(E_{ij})$ if $E_{ij} \neq \emptyset$, where E_{ij} is the set of all edges between clusters C_i and C_j. Once the number of coarse nodes falls below a threshold (typically, kC for some tuning constant C), *initial partitioning* computes an initial solution of the coarsest graph G_ℓ. Subsequently, contractions are undone, projecting the current solution to finer graphs and refining it. The total running time of a multilevel partitioner is the cumulative time for coarsening, initial partitioning, and refinement across all levels of the hierarchy $\mathcal{H}$.

3 Related Work

There has been a lot of research on graph partitioning, thus we refer the reader to surveys [8, 13] for a general overview and only focus on work closely related to our contributions here. As described above, modern general-purpose, high-quality graph partitioners such as Mt-Metis [38], Mt-KaHIP [3], Mt-KaHyPar [26], KaMinPar [29], and Jet [25] are mostly based on the multilevel paradigm, which constructs a hierarchy of coarser graphs during the coarsening phase.

Graph Coarsening. Early multilevel partitioners, like Chaco [32] and Metis [36], primarily employed coarsening strategies based on contracting graph matchings. While effective for mesh-like graphs due to high matching coverage (often 85-95% [35]), these strategies struggle with graphs exhibiting irregular structures, such as scale-free networks. On these graphs, small maximal matchings can result in much slower coarsening and potentially a linear number of levels. Subsequent developments addressed this limitation. Mt-Metis [39] introduced 2-hop matchings, extending small maximal matchings by further pairing nodes that have some degree of overlap in their neighborhoods until $\geq 75\%$ of nodes are contracted. This technique was subsequently also implemented by other partitioners [18, 25]. Alternative strategies focus on accelerating coarsening by grouping multiple nodes. These include methods based on cluster contraction [44, 3, 26, 29] and pseudo-matchings where nodes can match with multiple neighbors [1]. While enabling faster node reductions, they often constrain the weight of the clusters to ensure that finding a balanced initial partition is feasible. This can be problematic on graphs with highly connected hubs (e.g., the center of a star graph), potentially limiting the achievable coarsening ratio.

Graph Sparsification. Graph sparsification techniques aim to approximate a given graph with a sparser one (called *sparsifier*), typically containing substantially fewer edges while

preserving specific structural properties important for downstream tasks. This allows handling massive data sets where considering the full graph is computationally infeasible, as well as speeding up a variety of algorithms on graphs or matrices [2, 9, 22]. For graph partitioning, preserving cut properties (and thus approximately preserving the partition objective) is particularly relevant. An *ε-cut sparsifier* guarantees that every cut in the sparsifier has a weight within a $1\pm\varepsilon$ factor of the original cut. Benczúr and Karger showed that such sparsifiers with $O(n \log n/\varepsilon^2)$ edges exist for any graph and gave near-linear time constructions [9].

There are several approaches to construct sparsifiers. Spielman and Srivastava [51] introduced sparsification based on *effective resistance*, which often yields high-quality sparsifiers and preserves spectral properties closely related to cuts. However, this method can be computationally demanding. Alternatively, various heuristic sampling techniques exist, such as uniform edge sampling, k-neighbor sampling, and Forest Fire sampling [40, 42], which uses an analogy to a spreading wildfire. Chen et al. [16] provide a comparative study, suggesting that Forest Fire sampling outperforms uniform sampling for preserving cut-related properties.

KaMinPar. We integrate the techniques described in this paper into the KaMinPar [29] framework. KaMinPar is a shared-memory parallel multilevel graph partitioner. Its coarsening and uncoarsening phases are based on the *size-constrained label propagation* [44] algorithm, which is parameterized by a maximum cluster size (resp. block weight) U. In the coarsening resp. uncoarsening phase, each node is initially assigned to its own cluster resp. to its corresponding block of the partition. The algorithm then proceeds in rounds. In each round, the nodes are visited in some order. A node u is moved to the cluster resp. block K that contains the most neighbors of u without violating the size constraint U, i.e., $c(K) + c(u) \leq U$. The algorithm terminates once no nodes have been moved during a round or a maximum number of rounds has been exceeded. The coarsening further implements a 2-hop clustering strategy [29], which reduces the number of coarse nodes further whenever label propagation alone yields a node reduction factor less than 2. Since each round of size-constrained label propagation runs in linear time, and there is only a constant number of rounds, KaMinPar achieves linear time per hierarchy level for coarsening and uncoarsening.

The original paper [29] shows that KaMinPar achieves overall linear-time complexity under two key assumptions: (i) a constant node reduction factor between hierarchy levels, and (ii) bounded average degree for coarse graphs. While we will demonstrate in Theorem 1 that KaMinPar's coarsening strategy satisfies assumption (i), the inability to guarantee assumption (ii) results in a worst-case running time with an extra $log(n)$ factor.

4 Linear Time Multilevel Graph Partitioning

Multilevel algorithms construct a hierarchy $\mathcal{H} = \langle G =: G_1, G_2, \ldots, G_\ell \rangle$ of successively coarser representations of the input graph G. Each level of $\mathcal{H}$ is considered twice, during coarsening (to construct the next level) and during refinement (to improve the current partition). Assuming linear time for the coarsening and refinement on each level (see Section 3), the total sequential running time is $\Theta(\sum_{i=1}^{\ell} |V_i| + |E_i|)$. Without additional constraints on the number and size of the levels, the worst-case running time might be $\Theta(nm)$ or worse.

To obtain better guarantees, we need a geometric size reduction per level.[1] As a first step, we require that $|V_{i+1}| \leq \gamma |V_i|$ for some constant $\gamma < 1$ that is independent of G. If this is the case, the coarsened graph has constant size after a logarithmic number of steps, which

[1] In general, any series with a sum of $\mathcal{O}(1)$ works – a geometric series is, however, the most straightforward.

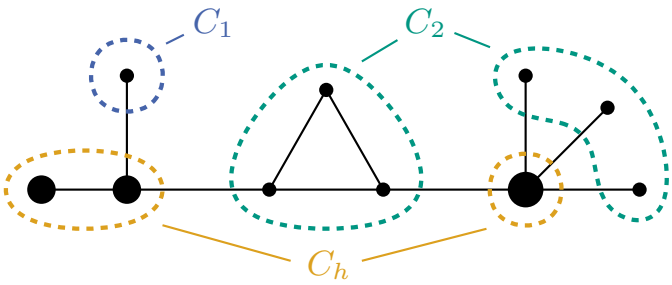


Figure 1 Illustration of Theorem 1, with examples of the different cluster types. Note that the green cluster to the right is created by 2-hop clustering.

already achieves a running time of $\mathcal{O}(n + m \log n)$. Combined with a similar guarantee for the number of edges, we get a linear total running time.

4.1 Reducing the Number of Nodes

As discussed in Section 3, the coarsening algorithms used in practice start by computing either a matching or a clustering of adjacent nodes. Typically, a maximum allowed node weight U is enforced for clusters. We use the term *size-constrained label propagation* to refer to a broad class of coarsening algorithms that form clusters of adjacent nodes and use a weight constraint. We require one essential property. In the resulting clustering, a node v never forms a singleton cluster as long as there is any adjacent cluster K with $c(K) + c(v) \leq U$.

Due to the weight constraint, size-constrained label propagation by itself is not sufficient for reducing the number of nodes (consider, e.g., a star graph). To solve this, partitioners use *2-hop clustering* as a second step, forming clusters of nodes that are not adjacent but instead have a common neighbor cluster. In the following, we provide the first formal proof that this guarantees a constant factor node reduction.

Consider a (non-isolated) node v in a singleton cluster $S = \{v\}$. Formally, we will assume that the algorithm assigns a *favorite cluster* K_S to S, out of the clusters adjacent to S. The 2-hop clustering then merges any nodes with the same favorite cluster, as long as this does not violate the weight constraint. Note that only considering favorite clusters is more restrictive than general 2-hop clustering, but is already sufficient for our purpose.

▶ **Theorem 1.** *The number of clusters obtained by size-constrained label propagation and 2-hop clustering with a maximal cluster weight $U \geq 2\frac{c(V)}{|V|}$ is at most*

$$|\mathcal{C}| \leq \frac{1}{2}|V| + \frac{c(V)}{U}$$

on any graph without isolated nodes.

Proof. We divide the set of clusters $\mathcal{C}$ into multiple subsets (see Figure 1 for an illustration). C_h is the set of *heavy* clusters with weight larger than $\frac{1}{2}U$. C_1 is the set of singleton clusters with weight at most $\frac{1}{2}U$ and C_2 is the set of clusters with multiple nodes and weight at most $\frac{1}{2}U$. Note that $\mathcal{C} = C_h \cup C_1 \cup C_2$. Let $r := \frac{1}{|C_2|}\sum_{K \in C_2} |K|$ be the average number of nodes for clusters in C_2. In combination, this results in the inequality $|V| \geq |C_h| + |C_1| + r|C_2|$.

Each singleton cluster $S \in C_1$ is only adjacent to clusters with weight larger than $U - c(S)$, and thus only clusters in C_h – otherwise, the node would have joined the lighter adjacent cluster. Consider the favorite cluster $K_S \in C_h$ of S. Due to 2-hop coarsening, there is no other cluster in C_1 with the same favorite (otherwise, 2-hop clustering would have joined them).

Algorithm 1 Graph Coarsening with Sparsification.

```
1 i ← 1, G_i ← G                                              // Input: graph G
2 while G_i not small enough do
3 |   G'_{i+1} ← Coarsen(G_i)
4 |   m̂ ← min{τ_e · |E(G_i)|, τ_d · (|E(G_i)|/|V(G_i)|) · |V(G'_{i+1})|}
5 |   if |E(G'_{i+1})| > ρ · m̂ then G_{i+1} ← Sparsify(G'_{i+1}, m̂)
6 |   else G_{i+1} ← G'_{i+1}
7 |   i += 1
8 return ⟨G_1, ..., G_i⟩                                      // Output: hierarchy H = ⟨G_1, ..., G_i⟩
```

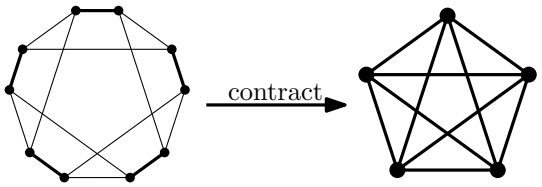


Figure 2 Contracting the bolded edges leads to increased density on the coarse graph.

Consequently, $|C_1| \leq |C_h|$. Moreover, $c(S)+c(K_S) > U$ gives, when summed over all clusters and combined with the definition of C_h, the inequality $c(V) \geq \sum_{K\in C_h} c(K)+\sum_{K\in C_1} c(K) = \sum_{K\in C_1}(c(K)+c(K_S)) + \sum_{K\in C_h\setminus\{K'_S|K'\in C_1\}} c(K) > U|C_1| + \frac{1}{2}U(|C_h|-|C_1|)$. Rearranged, this is $|C_h|+|C_1| \leq 2\frac{c(V)}{U}$.

Combining all inequalities, we get

$$\begin{aligned}
|\mathcal{C}| &= |C_h| + |C_1| + |C_2| \\
&\leq \frac{1}{r}|V| + (1-\frac{1}{r})|C_h| + (1-\frac{1}{r})|C_1| \\
&\leq \frac{1}{r}|V| + 2(1-\frac{1}{r})\frac{c(V)}{U} \\
&\leq \frac{1}{2}|V| + \frac{c(V)}{U}
\end{aligned}$$

For the final step, we use the observation that $xa+(1-x)b \leq \frac{1}{2}a+\frac{1}{2}b$ for $b \leq a$ and $x \leq \frac{1}{2}$. Since $r \geq 2$ and $U \geq 2\frac{c(V)}{|V|}$, we can apply this with $x = \frac{1}{r}$, $a = |V|$ and $b = 2\frac{c(V)}{U}$. ◀

Isolated nodes (i.e., nodes without a neighbor) are a special case as standard clustering algorithms do not handle them. Therefore, they are omitted from Theorem 1. However, it is trivial to either remove isolated nodes and reinsert them in the uncoarsening, or alternatively cluster them with each other (we do the latter).

Note that the precondition $U \geq 2\frac{c(V)}{|V|}$ is no limitation for the applicability of Theorem 1. In practice, much larger values are used for U (in our case $U = \frac{c(V)}{160k}$, see Section 4.3). However, the theorem does not include clustering approaches which limit the *number* of nodes in a cluster (e.g., allowing only matchings). We note that in this case similar, but weaker, bounds can be obtained with an analogous line of reasoning.

4.2 Reducing the Number of Edges via Sparsification

As discussed above, geometric node reduction in coarsening strategies can still lead to superlinear total running time due to increasing graph density at coarser levels (e.g., Figure 2). Achieving linear time necessitates preventing this phenomenon. Since common clustering algorithms can not guarantee a geometric reduction in the number of edges, we propose an alternative strategy: sparsifying the contracted graph when the edge count does not shrink at a sufficient rate. To validate this concern, we first show that the issue can arise on important classes of graphs using the example of Erdős-Rényi graphs. Subsequently, we will detail our sparsification approach.

Consider the coarsening hierarchy of a sparse Erdős-Rényi graph $G_0 = G(n_0, c/n_0)$ with n_0 nodes and edge probability $p_0 := c/n_0$ for some constant c. Assume that coarsening halves the number of nodes at each level by contracting pairs of nodes, and that coarse graphs also behave like Erdős-Rényi graphs. In other words, $G_i = G(n_i, p_i)$ with $n_i = n_{i-1}/2$ and $p_i \approx 1 - (1 - p_{i-1})^4$ for $i > 0$ (there is an edge between two coarse nodes if any of the four potential edges between the corresponding nodes in G_{i-1} existed). Note that $n_i = n_0/2^i$ and $p_i = 1 - (1 - p_0)^{4^i} = 1 - (1 - c/n_0)^{4^i} \approx 1 - e^{-4^i \cdot c/n_0}$. Let $i = \alpha \log(n_0)$, then $p_i \approx 1 - e^{-cn_0^{2\alpha - 1}} \xrightarrow{n_0 \to \infty} 0$ for $\alpha < \frac{1}{2}$, i.e., there are $\Theta(\log n_0)$ sparse levels. On these,

$$\frac{\mathbb{E}[m_{i+1}]}{\mathbb{E}[m_i]} = \frac{1 - (1 - p_i)^4}{p_i} \frac{n_i(n_i - 2)/8}{n_i(n_i - 1)/2} \xrightarrow{n_0 \to \infty} \frac{1 - (1 - p_i)^4}{4p_i} \approx \frac{1 - (1 - 4p_i)}{4p_i} = 1,$$

since $n_i = n_0/2^i \geq n_0/2^{\alpha \log n_0} > \sqrt{n_0} \to \infty$ and $(1 - p_i)^4 \approx 1 - 4p_i$ for small p_i. Thus, the number of coarse edges remains relatively constant, leading to overall $\mathcal{O}(m_0 \log(n_0))$ time.

To achieve linear time, we therefore limit the number of edges through sparsification as outlined in Algorithm 1. Let $G'_{i+1} = (V_{i+1}, E'_{i+1})$ denote the current graph before sparsification, obtained by contracting the previous graph G_i (line 1). We obtain $G_{i+1} = (V_{i+1}, E_{i+1})$ by sparsifying the edges of G'_{i+1} so that the size of E_{i+1} is bounded by a threshold $\hat{m}$, defined as

$$\hat{m} := \min\{\tau_e \cdot |E_i|, \tau_d \cdot \frac{|E_i|}{|V_i|} \cdot |V_{i+1}|\}.$$

Here, τ_e is the *edge threshold* parameter, limiting the coarse edge count relative to the current graph's edge count, and τ_d is the *density threshold* parameter, likewise limiting the average degree of the coarse graph. Since sparsification itself introduces computational overhead, we only apply it if the potential edge reduction is significant. Specifically, we trigger sparsification only if $|E'_{i+1}| > \hat{m}$ *and* the target edge count $\hat{m}$ represents a substantial reduction from the current edge count $|E'_{i+1}|$, quantified by the condition $|E'_{i+1}|/\hat{m} \geq \rho$, where $\rho \geq 1$ is a tunable constant (line 1). Once triggered, we use one of the following sampling algorithms to reduce the edge count to $\hat{m}$ (in expectation), before adding the sparsified graph to the hierarchy (line 1). Since our goal is to achieve overall linear time, we only consider linear time sparsification algorithms. Further, sparsification must be fast in practice for speedups to be attainable.

Uniform Sampling: UR. As a simple baseline, we consider uniform random sampling. Each edge $e \in E'_{i+1}$ is selected independently with probability $p := \hat{m}/|E'_{i+1}|$, resulting in expected $\hat{m}$ edges. Note that this approach is oblivious to edge weights – although heavier edges have larger influence on the partitioning objective and are thus likely more important.

Algorithm 2 Weighted Forest Fire: graph $G = (V, E)$, burn ratio ν, probability p. The only difference to the original Forest Fire [41] algorithm is highlighted blue.

```
1  S ← new Array() of size |E|                                   // Scores
2  b ← 0                                                         // Number of burnt edges
3  while b ≤ ν|E| do in parallel
4      Q ← new FIFO(); Q.push(random node from V)                 // BFS queue
5      T ← new Set()                                             // Visited nodes
6      while Q ≠ ∅ do
7          u ← Q.pop()
8          while N(u) \ T ≠ ∅ do
9              Sample v from N(u) \ T with prob. ω({u,v}) / Σ_{v∈N(u)\T} ω({u,v})
10             T.insert(v); Q.push(v)
11             S[{u,v}] atomic+= 1; b atomic+= 1
12             Break with prob. p
13 return S
```

Weighted Threshold Sampling: T-Weight. To incorporate edge weights, we consider a weighted threshold sampling strategy. First, we identify the weight threshold $\omega_t := \omega(e_t)$ corresponding to the $\hat{m}$-th heaviest edge e_t in G'_{i+1}. This can be done in expected time $\mathcal{O}(|E'_{i+1}|)$ using the quickselect algorithm. Based on ω_t, we partition E'_{i+1} into three disjoint sets $E'^{<}_{i+1}$, $E'^{=}_{i+1}$, and $E'^{>}_{i+1}$, for coarse edges with weight smaller than, equal to, or larger than ω_t. Edges in $E'^{<}_{i+1}$ are discarded, while edges in $E'^{>}_{i+1}$ are kept. To reach the target size $\hat{m}$, we further sample edges from $E'^{=}_{i+1}$ uniformly with probability $p := \frac{\hat{m}-|E'^{>}_{i+1}|}{|E'^{=}_{i+1}|}$.

(Weighted) Forest Fire Sampling: T-(W)FF. We further include a variation of threshold sampling that uses *Forest Fire* [42] scores rather than edge weights, as this performed well as a cut-preserving sparsifier in Ref. [16]. We include a brief description for self-containment and extend the algorithm to take edge weights into consideration (Algorithm 2). The algorithm computes edge scores by simulating fires spreading through the graph via multiple traversals starting from random nodes. When visiting a node u, the number of neighbors X to be visited is drawn from a geometric distribution paramterized by a tunable parameter p. The standard forest fire algorithm subsequently samples X distinct nodes (without replacement) from u's unvisited neighbors. We incorporate edge weights by making this sampling weight-dependent: the probability of selecting neighbor v of u is proportional to the edge weight $\omega(\{u, v\})$ relative to the total edge weight between u and its unvisited neighbors (line 9). Note that this modification (marked blue in Algorithm 2) is the only difference to Ref. [41]. Each edge traversal during this process increments the *burn* score of the edge (line 11). The algorithm stops scheduling fires once the cumulative burn score `b` exceeds $\nu|E|$ (line 3) for some *burn ratio* $\nu > 0$. After computing the edge importance scores, we use the weighted threshold sampling strategy to sparsify the graph (using the importance scores instead of edge weights).

4.3 Putting it Together

Based on the discussed insights, we propose a linear time multilevel algorithm that builds upon KaMinPar [29]. We leverage the existing clustering and refinement algorithms

available in KaMinPar, whose running time is linear in the size of the current hierarchy level (see Section 3). We introduce two necessary changes to achieve linear time for the overall algorithm. Most importantly, we introduce edge sparsification as discussed in Section 4.2, ensuring the number of edges shrinks geometrically. In addition, we replace the coarsening and initial partitioning used by the default configuration of KaMinPar with a more traditional approach. This is because the default configuration is amenable to scenarios where expensive bipartitioning happens on a relatively large graph, adding a $\Omega(n \log n)$ term to the running time in the worst-case. We refer to our technical report [30] for details.

Instead, we use size-constrained label propagation with subsequent recursive bipartitioning, following other state-of-the-art multilevel algorithms [3, 28, 44]. Similar to Mt-KaHyPar [26], the cluster weight limit is $U = \frac{c(V)}{160k}$ and we limit the node reduction per coarsening step to at most 2.5×. As this is combined with 2-hop clustering, Theorem 1 guarantees a geometric shrink factor until a size of $|V_i| = 320k$ is reached. The coarsening terminates at $160k$ nodes or if the current shrink factor is too small, thereby resulting in a graph with size $\mathcal{O}(k)$.[2] The recursive bipartitioning then requires total time $\mathcal{O}(k \log k)$, which is linear under the extremely weak assumption that $k \log k \in \mathcal{O}(n + m)$.

So far, we have argued from a sequential point of view. In the parallel setting, the consequence is that our algorithm needs only linear work. With regards to scalability, the sparsification algorithms described in Section 4.2 lend themselves to a rather straightforward parallelization. Combined with the excellent scalability of the coarsening and refinement algorithms of KaMinPar [29] and the fact that initial partitioning is insignificant for the total running time, we maintain the scalability of default KaMinPar while reducing the required work.

5 Quantifying Worst-Case Instances

As discussed, we need edge sparsification to achieve linear time if coarsening does not shrink the number of edges geometrically. However, it would be useful to understand for which graphs sparsification is required and for which it is not – both for theoretical insights into the structure of worst-case instances and to allow empirical estimates. Given a clustering C of a graph G, we are thus interested in the number of edges of G', where G' is the graph created by contracting C. If $|E(G')| \approx |E(G)|$ for clusterings computed by the coarsening algorithm, sparsification is required to further reduce the number of edges.

Intuitively, this is the case for graphs with low locality – edges might lead anywhere and are thus hard to contain in small clusters. For many random graph models, this is rather easy to decide. For example, sparse Erdős-Rényi graphs are highly non-local, thus necessitating sparsification (see Section 4.2). On the other hand, for random geometric graphs (i.e., unit disk graphs) coarsening algorithms reduce the number of edges very efficiently. However, to classify real-world instances or more complex graph models, a general criterion is needed.

Classification via Modularity. We propose that the *modularity* of a graph allows to estimate whether sparsification is necessary.[3] Modularity was introduced by Newman and Girvan to evaluate the quality of a clustering with regards to community structure [46], and modularity based community detection algorithms are used in many applications [33, 54, 55]. Given

[2] Note that sparsification ensures $\mathcal{O}(k)$ edges – although this is not necessary for linear time.
[3] There are also multiple other locality metrics, but these are less useful. For example, the clustering coefficient is based on the number of triangles. However, this does not result in any useful bounds.

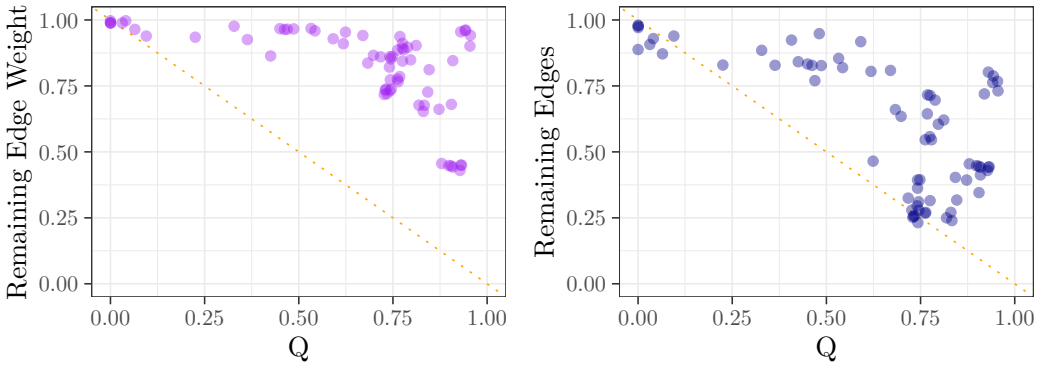


Figure 3 Remaining total edge weight (**left**) and number of edges (**right**) after one coarsening step, compared to the modularity of the graph. The y-values are denoted as a fraction of the initial value. Based on Lemma 2, we expect most points (i.e., graphs) to be in the upper right half.

a clustering $C = \{V_1, \ldots, V_{|C|}\}$, let $e_{ij} := \frac{1}{2|E|}|E(V_i, V_j)|$ denote the fraction of edges that connect cluster i and cluster j (only counted in one direction). Further, let $a_i := \sum_j e_{ij}$ be the fraction of edges with one endpoint in cluster i. The modularity of the clustering is then defined as $Q_C := \sum_i \left(e_{ii} - a_i^2\right)$, where $Q_C \in [-\frac{1}{2}, 1]$. The modularity $Q \in [0, 1]$ of the graph itself is the maximum modularity of all possible clusterings. As demonstrated in the following, modularity is a good fit for our purpose.

▶ **Lemma 2.** *For a given clustering, the total fraction of edges that connect nodes within the same cluster is bounded by*

$$Q_C \leq \sum_i e_{ii} \leq Q_C + \alpha_C$$

where $\alpha := \max_i a_i$ is the maximum fraction of edges with endpoints in the same cluster.

Proof. Since $Q_C = \sum_i \left(e_{ii} - a_i^2\right)$, the left side of the inequality follows immediately. Further, $\sum_i e_{ii} = Q_C + \sum_i a_i^2 \leq Q_C + \sum_i (\max_j a_j) a_i = Q_C + \max_i a_i$. Note that $\sum_i a_i = 1$ since each edge is connected to exactly two clusters. ◀

Lemma 2 provides a lower bound of $1 - Q_C - \alpha_C$ for the fraction of edges connecting different clusters. Unfortunately, this does not correspond directly to the edges of G' – if multiple edges connect the same pair of clusters (we say that the edges are *parallel*), they are combined into a single edge, thereby further reducing the number of remaining edges. The actual bound is thus $1 - Q_C - \alpha_C - p_C \leq \frac{|E(G')|}{|E(G)|}$, where p_C is the number of parallel edges.

Let us consider the case where the clusters are small, such as during the first steps of multilevel coarsening. Then, $1 - Q$ is an approximate lower bound for the number of edges. If clusters are small, any cluster is only incident to a small fraction of all edges and edges are unlikely to be parallel, i.e., both α_C and p_C are small. Moreover, Q_C is almost certainly smaller than Q as achieving maximum modularity often necessitates large clusters [23]. Consequently, we expect that $1 - Q \lesssim 1 - Q_C - \alpha_C - p_C \leq \frac{|E(G')|}{|E(G)|}$ if clusters are small, making $1 - Q$ an accurate bound.

Empirical Effect of Modularity. To verify whether $1 - Q$ is a useful bound in practice, we provide an empirical evaluation on a set of 71 large graphs which is used in multiple recent

works on graph partitioning [37, 43, 49]. Figure 3 shows computed modularity scores of the graphs in relation to the fraction of edges that remains after one step of our multilevel coarsening algorithm (see Section 4.1). Since computing the modularity of a graph is itself NP-hard [12], we calculate approximate scores with the well-known Louvain algorithm [10], using the implementation from NetworKit [5, 52].

With regards to the total edge weight of G' (left, corresponds to inter-cluster edges in G), $1 - Q$ is almost a strict lower bound. For most graphs, the fraction of remaining weight is much larger than $1 - Q$. The actual number of edges after combining parallel edges (right) is often significantly smaller than the weight. However, $1 - Q$ is still a mostly accurate bound. Therefore, modularity is indeed useful to predict the coarsening behavior of multilevel algorithms – graphs with low modularity are likely to be worst-case instances.

6 Experiments

Setup. We implemented the described sparsification algorithms within the KaMinPar [29] framework and compiled it using `gcc` 14.2.0 with flags `-O3 -mtune=native -march=native`. The code is parallelized using TBB [47]. All experiments are performed on a machine equipped with two 32-core Intel Xeon Gold 6530 processors (2.1 GHz) and 3 TB RAM running Rocky Linux 9.5. We only use one of the two processors (i.e., 32 cores) to avoid NUMA effects.

Competitors. In Section 6.3, we compare our algorithm against PuLP [50] (v1.1) and Cuttana [31] (commit `ed0c182` in the official GitHub repository[4]). PuLP is a linear time single-level partitioner which focuses on shared-memory scalability and low memory usage. PuLP starts by growing (unbalanced) initial clusters from a random assignment and then performs multiple iterations of alternating balancing and cut minimization. Each phase is implemented with a fixed number of label propagation rounds. The authors showed that PuLP achieves considerable speedups in comparison to multiple well-known multilevel algorithms. Cuttana is a streaming partitioner which improves upon the solution quality of previous streaming approaches with a node buffering technique. Instead of assigning each node greedily, it uses a buffer of fixed size to delay assignment until more information on the neighborhood is available or the buffer overflows. In addition, Cuttana uses a more fine-grained subpartition which allows to further refine the solution quality after the initial assignment. We use the default settings for PuLP and configure Cuttana following the parameters described by the authors, i.e., $\frac{\mathcal{K}'}{\mathcal{K}} = 4\,096$, $D_{\max} = 1\,000$ and max_qsize $= 10^6$.

Instances. We focus on graphs for which coarsening increases edge density substantially. This happens on 17 out of 71 graphs of a benchmark set used in previous works on graph partitioning [43, 37, 49] (mostly real-world k-mer and social graphs, and graphs deduced from text recompression [34]). We only use these graphs since sparsification is not triggered on the remaining instances, thus leaving the algorithm unchanged. Note that there is no running time overhead in this case. We further include 6 social graphs from the Sparse Matrix Collection [19] and generate random graphs: Erdős-Rényi graphs (using KaGen [24]), as well as Chung-Lu [45], Planted Partition, and R-MAT [15] graphs (using NetworKit [5]). These graphs are inherently non-local, thus especially challenging for linear-time partitioning. Overall, the benchmark set comprises 39 graphs (see Ref. [30]) with 511 K to 1.8 G undirected edges. The graphs deduced from text recompression feature node weights. All other graphs are

[4] `https://github.com/cuttana/cuttana-partitioner`

unweighted. Tuning experiments are performed on a subset containing 8 randomly drawn graphs spanning different types.

Methodology. We consider an *instance* as the combination of a graph and a number of blocks k. We set the imbalance tolerance to $\varepsilon = 3\%$, use $k \in \{3, 7, 8, 16, 37, 64\}$ and perform 5 repetitions for each instance using different seeds. Results (running time, edge cut) are averaged arithmetically per instance over these repetitions. When aggregating across multiple instances, we use the geometric mean to ensure that each instance has equal influence.

Performance Profiles. To compare the edge cuts of different algorithms, we use *performance profiles* [20]. Let $\mathcal{A}$ be the set of all algorithms we want to compare, $\mathcal{I}$ the set of instances, and $\text{cut}_A(I)$ the edge cut of algorithm $A \in \mathcal{A}$ on instance $I \in \mathcal{I}$. For each algorithm A, we plot the fraction of instances

$$\mathcal{P}_A(\tau) \coloneqq \frac{|\{I \in \mathcal{I} \ : \ \text{cut}_A(I) \leq \tau \cdot \min_{A' \in \mathcal{A}} \text{cut}_{A'}(I)\}|}{|\mathcal{I}|}$$

on the y-axis and τ on the x-axis. Achieving higher fractions at lower τ-values is considered better. In particular, $\mathcal{P}_A(1)$ denotes the fraction of instances for which algorithm A performs best, while $\mathcal{P}_A(\tau)$ for $\tau > 1$ illustrates the robustness of the algorithm. For example, an algorithm A with $\mathcal{P}_A(1) = 0.49$ but $\mathcal{P}_A(1.01) = 1.0$ (i.e., never more than 1% worse than the best) might be preferable to an algorithm B with $\mathcal{P}_B(1) = 0.51$ that only achieves $\mathcal{P}_B(\tau) = 1.0$ at much larger τ (indicating much worse partitions on some inputs).

6.1 Parameter Study

We begin our evaluation by tuning the parameters introduced in Section 4.2. Recall that these are the edge and density thresholds τ_e and τ_d, which control the number of coarse edges, and the minimum reduction factor ρ, which controls whether sparsification is triggered on a given hierarchy level. We use the tuning benchmark subset and $k = 16$ for this experiment to limit computational costs.

The results are shown in Figure 4, where we plot geometric mean edge cuts and running times relative to the KaMinPar baseline without sparsification for $\tau_e \in \{1/4, 1/2, 1\}$, $\tau_d \in \{1/2, 1, 2\}$ and $\rho \in \{1, 2, 3, 4, 8\}$. We observe similar speedups of up to 1.63× for weighted threshold sampling (T-Weight) and uniform sampling (UR). T-Weight achieves the highest speedup (1.63×) at $\tau_e = 1/2$, $\tau_d = 1/2$ and $\rho = 3$, while UR achieves the same speedup at slightly different parameters ($\tau_e = 1/4$, $\tau_d = 1$ and $\rho = 2$). With these parameters, edge cuts increase by 12.6% and 27.6% for T-Weight and UR, respectively. Surprisingly, more aggressive sparsification (i.e., smaller τ_e, τ_d and ρ) does not achieve larger speedups. This is likely due to several factors. First, the sparsification process itself introduces computational overhead which can counteract potential speedups, particularly when the size reduction is modest. Second, excessive sparsification degrades partition quality considerably, thereby increasing the workload required for the refinement algorithm to converge to a local optimum. Hence, moderate sparsification seems favorable.

Larger ρ seems beneficial for maintaining partition quality. At $\rho = 4$ (i.e., only sparsify if reducing the number of edges by a factor of ≥ 4), both T-Weight and UR show similar speedups as with smaller ρ and partition quality close to the baseline. We therefore pick $\tau_e = \tau_d = 1/2$ and $\rho = 4$ for subsequent experiments, where T-Weight and UR achieve speedups of 1.56× and 1.59×, while increasing edge cuts by 0.9% and 5.3%, respectively.

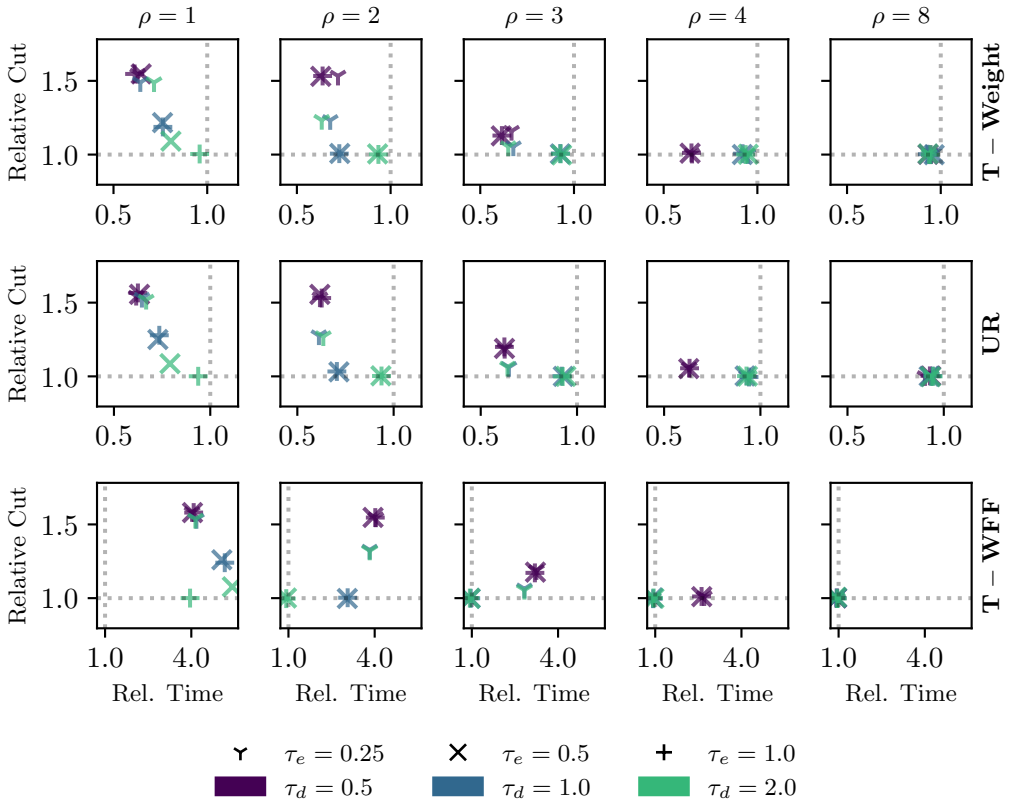


Figure 4 Relative cut and running time of KaMinPar with weighted threshold sampling (T-Weight), uniform sampling (UR), or threshold sampling via Weighted Forest Fire scores (T-WFF) versus baseline (KaMinPar without sparsification) on the tuning benchmark set with $k = 16$.

Lastly, we look at threshold sampling using Weighted Forest Fire (T-WFF) scores. For T-WFF itself, we use $p = 0.6$ and $\nu = 0.5$, since these parameters performed best during preliminary experimentation. We observe the fastest running times using parameters that do not trigger sparsification, suggesting that T-WFF does not provide practical speedups. At $\tau_e = \tau_d = 1/2$ and $\rho = 4$, T-WFF is 2.62× slower while incurring a 1.0% increase in cut size.

6.2 Effects of Sparsification

Next, we evaluate the proposed sparsification techniques on the full benchmark set. As can be seen in Figure 5, T-Weight (geometric mean running time 1.43 s) and UR (1.40 s) achieve similar speedups of 1.49× resp. 1.52× over the baseline (no sparsification, 2.13 s). T-Weight achieves considerably better partition quality (increase in average edge cut by 1.5%) than UR (increase by 5.5%). T-WFF outperforms T-FF, but is not competitive: its partition quality is slightly worse (increase by 3.9%) while much slower (7.04 s). We thus focus on T-Weight.

Looking at Figure 6, we can see that sparsification reduces the number of edges on coarse graphs considerably. Without sparsification, the graphs on the first hierarchy levels (i.e., after the first coarsening step) contain, on average, 75% of the edges of the input graphs, but only 39% of the nodes. With sparsification, the average edge count reduces to 28%.

As shown in Figure 7 (left), the speedup from T-Weight sparsification varies considerably

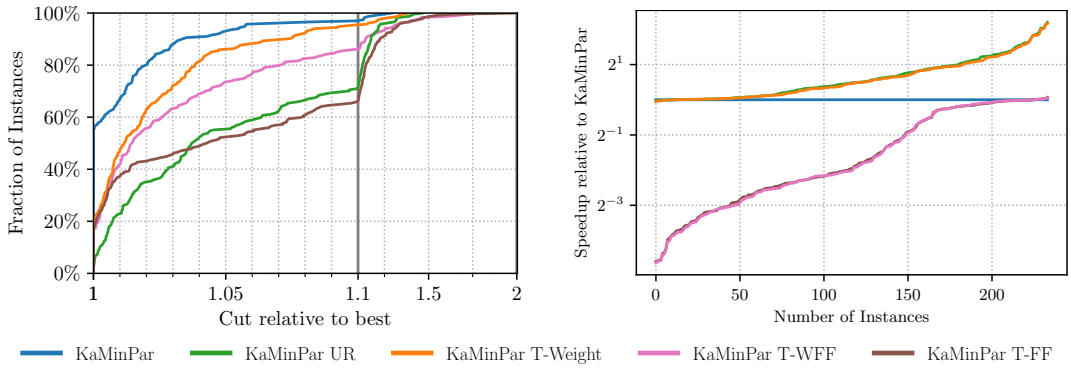


Figure 5 Partition quality as *performance profile* (**left**) and speedup over baseline (no sparsification, **right**) of sparsification algorithms: weighted threshold sampling (T-Weight), uniform sampling (UR), and threshold sampling via (Weighted) Forest Fire scores (T-(W)FF).

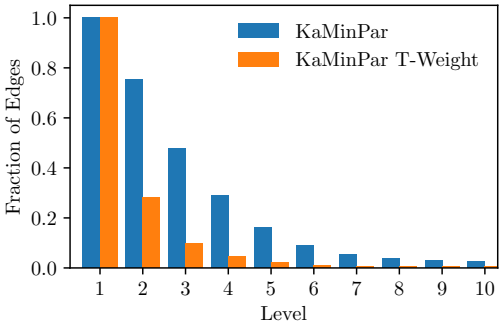


Figure 6 Relative geometric mean number of edges per hierarchy level (levels 1-10), comparing no sparsification against T-Weight (weighted threshold sampling) sparsification. Edge counts are relative to the input graphs. The final value is propagated for hierarchies shorter than 10 levels.

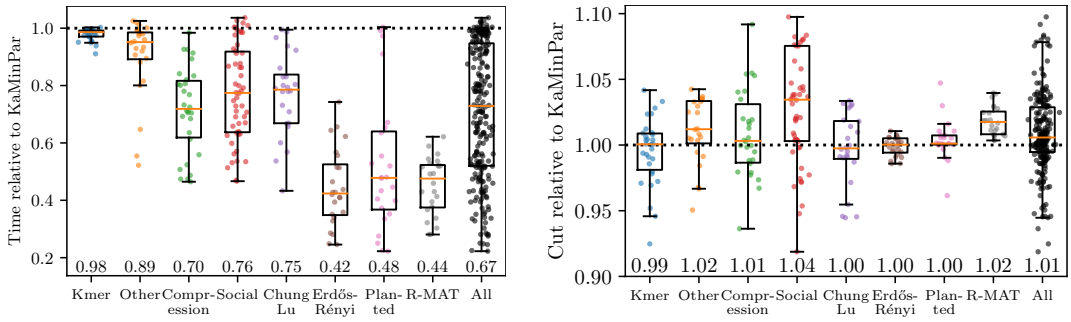


Figure 7 Comparison of KaMinPar with T-Weight sparsification relative to the baseline without sparsification grouped by graph class (lower is faster resp. higher-quality). **Left:** Relative running times with the geometric mean relative time annotated per class. **Right:** Relative cuts with the geometric mean relative cut annotated per class.

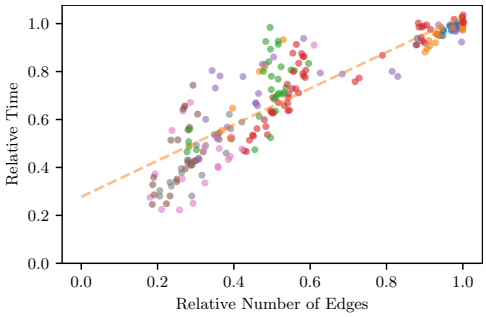


Figure 8 Relationship between running time of KaMinPar T-Weight relative to KaMinPar without sparsification and the hierarchy size ratio (number of total edges across all hierarchy levels after sparsification relative to no sparsification). Note the strong correlation (coefficient ≈ 0.893).

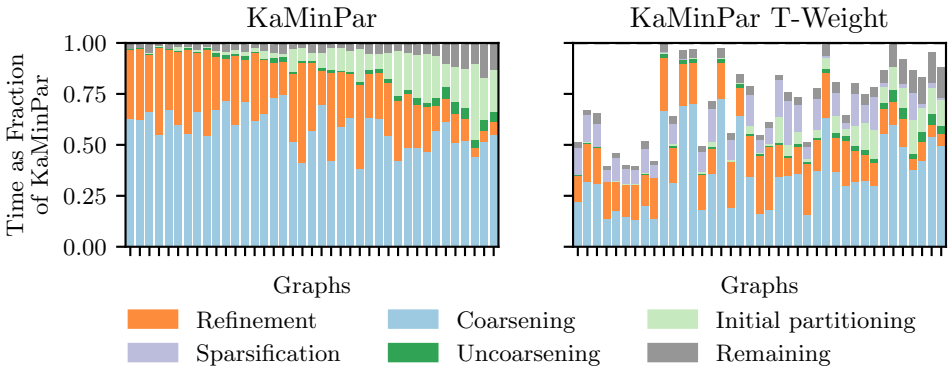


Figure 9 Relative running time attribution for KaMinPar without sparsification (**left**) and with T-Weight sparsification (**right**) using $k = 16$. Graphs are sorted by the total running time of KaMinPar without sparsification in descending order.

with graph class, with moderate cut size increases on real-world grahs (Figure 7, right). Non-local graphs with extremely low modularity (Erdős-Rényi, Planted Partition, R-MAT) show substantial gains (average speedup $> 2\times$, up to $4\times$), while real-world text recompression ($\approx 1.43\times$) and social graphs ($\approx 1.32\times$) exhibit moderate speedups. K-mer graphs see negligible benefit. This variation correlates strongly with the reduction in graph hierarchy size (number of edges across all hierarchy levels): instances with greater reduction achieve faster relative running times (Figure 8). The observed speedups stem primarily from reduced time in the coarsening and refinement phases, see Figure 9. Without sparsification, these phases consume on average 55% (1.17 s) and 23% (0.48 s) of the total partitioning time (2.13 s), respectively. Sparsification reduces these to 0.65 s and 0.32 s, respectively. This improvement comes at low cost, as the sparsification step itself averages only 0.10 s out of 1.62 s when triggered (94% of the instances).

6.3 Comparison against Competing Partitioners

Finally, we compare KaMinPar with T-Weight sparsification against alternative linear-time partitioners: single-level PuLP [50] and streaming Cuttana [31]. We found that Cuttana is rather slow and does not support node weights. Thus, we limit our benchmark set to unweighted graphs with $m \leq 2^{26}$ edges (18 out of 39 graphs).

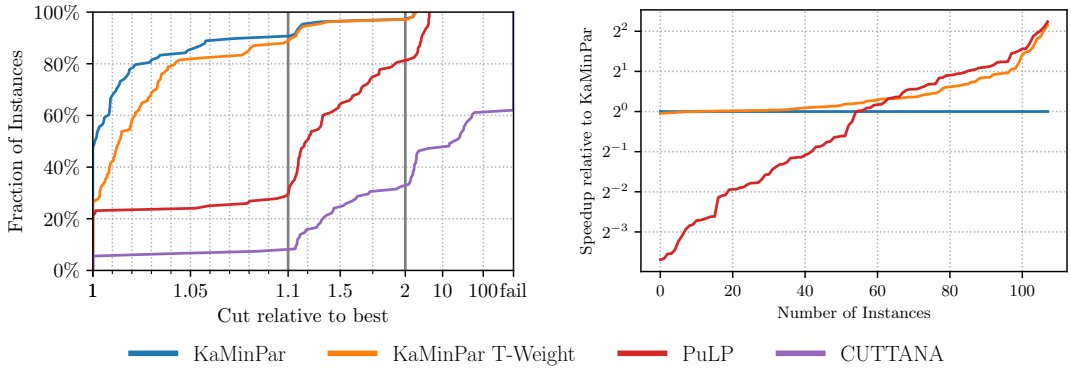


Figure 10 Partition quality (**left**) and relative running time (**right**) on the reduced benchmark set and all k values for KaMinPar without and with T-Weight sparsification, PuLP and Cuttana. Speedups are plotted relative to KaMinPar without sparsification. Cuttana is omitted from the speedup plot since it is $\geq 72\times$ slower than all other algorithms.

As shown in Figure 10 (left), KaMinPar with T-Weight sparsification computes considerably better partitions with average cuts 30% and 66% smaller than those of PuLP and Cuttana, respectively. Compared to KaMinPar *without* sparsification, edge cuts are slightly larger (cutting 1% more edges on average), but are still within a factor of 1.10 to the best cut found on 88% of all instances (vs. 90% for non-sparsifying KaMinPar). In contrast, PuLP and Cuttana compute edge cuts within factors 1.26 and 1.93 to the best cuts found on only *half of the instances*, respectively. PuLP computes the best partitions for 21% of the instances, predominantly Erdős-Rényi and Planted Partition graphs. Cuttana crashes on 39% of the instances (we exclude these instances in pairwise aggregates).

Through sparsification, the geometric mean running time of KaMinPar reduces from 0.48 s to 0.37 s. PuLP (0.63 s) is slower than non-sparsifying KaMinPar on average, but proves faster on 53 (resp. 48 vs. sparsifying KaMinPar) and twice as fast on 23 (resp. 2) out of 108 instances. Cuttana is $72\times$ slower than PuLP (and thus the other algorithms), although we note that comparing running times fairly is difficult since Cuttana interleaves computation with graph I/O from SSD (I/O times are excluded for the other algorithms).

7 Conclusion

Current graph partitioning algorithms can be classified into high-quality but superlinear multilevel algorithms, and cheaper linear time approaches such as single-level partitioning and streaming partitioning. We demonstrate both in theory and in practice that it is possible to achieve the best of both worlds at once. Our linear time multilevel algorithm uses edge sparsification to constrain the size of subsequent coarser levels, which provably guarantees linear work while maintaining scalability to many cores. We minimize quality loss by choosing appropriate thresholds for triggering the sparsification step and, if triggered, removing the edges with lowest weight. As a result, our multilevel algorithm is faster than state-of-the-art single-level and streaming approaches while consistently computing better solutions – making multilevel the preferable choice even if extremely short running time is required.

References

1 Amine Abou-Rjeili and George Karypis. Multilevel Algorithms for Partitioning Power-Law Graphs. In *20th International Conference on Parallel and Distributed Processing (IPDPS)*. IEEE Computer Society, 2006. doi:10.1109/IPDPS.2006.1639360.

2 Nesreen K. Ahmed, Jennifer Neville, and Ramana Rao Kompella. Network Sampling: From Static to Streaming Graphs. *ACM Transactions on Knowledge Discovery from Dat*, 8(2):7:1–7:56, 2013. doi:10.1145/2601438.

3 Yaroslav Akhremtsev, Peter Sanders, and Christian Schulz. High-Quality Shared-Memory Graph Partitioning. In *24th European Conference on Parallel Processing (Euro-Par)*, pages 659–671. Springer, August 2018. doi:10.1007/978-3-319-96983-1_47.

4 Konstantin Andreev and Harald Räcke. Balanced Graph Partitioning. In *16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 120–124, 2004. doi:10.1145/1007912.1007931.

5 Eugenio Angriman, Alexander van der Grinten, Michael Hamann, Henning Meyerhenke, and Manuel Penschuck. Algorithms for Large-Scale Network Analysis and the NetworKit Toolkit. In *Algorithms for Big Data: DFG Priority Program 1736*, pages 3–20. Springer, 2023. doi:10.1007/978-3-031-21534-6_1.

6 Amel Awadelkarim and Johan Ugander. Prioritized Restreaming Algorithms for Balanced Graph Partitioning. In *26th Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 1877–1887. ACM, 2020. doi:10.1145/3394486.3403239.

7 Cevdet Aykanat, Berkant Barla Cambazoglu, Ferit Findik, and Tahsin M. Kurç. Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids. *Journal of Parallel and Distributed Computing*, 67(1):77–99, 2007. doi:10.1016/J.JPDC.2006.05.005.

8 David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. *Graph Partitioning and Graph Clustering*, volume 588. American Mathematical Society Providence, 2013. doi:10.1090/conm/588.

9 András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *28th Symposium on Theory of Computing (STOC)*, pages 47–55. ACM, 1996. doi:10.1145/237814.237827.

10 Vincent D. Blondel, Jean Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008. doi:10.1088/1742-5468/2008/10/P10008.

11 Erik G. Boman, Karen D. Devine, and Sivasankaran Rajamanickam. Scalable Matrix Computations on Large Scale-Free Graphs Using 2D Graph Partitioning. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 50:1–50:12. ACM, 2013. doi:10.1145/2503210.2503293.

12 Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. Maximizing Modularity is Hard, 2006. doi:10.48550/arXiv.physics/0608255.

13 Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent Advances in Graph Partitioning. In *Algorithm Engineering*, volume 9220, pages 117–158. Springer, 2016. doi:10.1007/978-3-319-49487-6_4.

14 Ümit Çatalyürek, Karen Devine, Marcelo Faraj, Lars Gottesbüren, Tobias Heuer, Henning Meyerhenke, Peter Sanders, Sebastian Schlag, Christian Schulz, Daniel Seemaier, et al. More Recent Advances in (Hyper)Graph Partitioning. *ACM Computing Surveys*, 55(12):253–253, 2023. doi:10.1145/3571808.

15 Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A Recursive Model for Graph Mining. In *4th International Conference on Data Mining (ICDM)*, pages 442–446. SIAM, 2004. doi:10.1137/1.9781611972740.43.

16 Yuhan Chen, Haojie Ye, Sanketh Vedula, Alex M. Bronstein, Ronald G. Dreslinski, Trevor N. Mudge, and Nishil Talati. Demystifying Graph Sparsification Algorithms in Graph Properties

Preservation. *Proceedings of the VLDB Endowment*, 17(3):427–440, 2023. doi:10.14778/3632093.3632106.

17 Adil Chhabra, Florian Kurpicz, Christian Schulz, Dominik Schweisgut, and Daniel Seemaier. Partitioning Trillion Edge Graphs on Edge Devices, 2024. doi:10.48550/arXiv.2410.07732.

18 Timothy A. Davis, William W. Hager, Scott P. Kolodziej, and S. Nuri Yeralan. Algorithm 1003: Mongoose, a Graph Coarsening and Partitioning Library. *ACM Transactions on Mathematical Software*, 46(1), 2020. doi:10.1145/3337792.

19 Timothy A. Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1:1–1:25, November 2011. doi:10.1145/2049662.2049663.

20 Elizabeth D. Dolan and Jorge J. Moré. Benchmarking Optimization Software with Performance Profiles. *Mathematical Programming*, 91(2):201–213, 2002. doi:10.1007/s101070100263.

21 Marcelo Fonseca Faraj and Christian Schulz. Buffered Streaming Graph Partitioning. *ACM Journal of Experimental Algorithmics (JEA)*, 27:1.10:1–1.10:26, 2022. doi:10.1145/3546911.

22 Sebastian Forster and Tijn de Vos. Faster Cut Sparsification of Weighted Graphs. *Algorithmica*, 85(4):929–964, 2022. doi:10.1007/s00453-022-01053-4.

23 Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *National Academy of Sciences*, 104(1):36–41, 2007. doi:10.1073/pnas.0605965104.

24 Daniel Funke, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. Communication-free Massively Distributed Graph Generation. In *32nd International Parallel and Distributed Processing Symposium (IPDPS)*, pages 336–347. IEEE Computer Society, 2018. doi:10.1109/IPDPS.2018.00043.

25 Michael S. Gilbert, Kamesh Madduri, Erik G. Boman, and Siva Rajamanickam. Jet: Multilevel Graph Partitioning on Graphics Processing Units. *SIAM Journal of Scientific Computing.*, 46(5):700, 2024. doi:10.1137/23M1559129.

26 Lars Gottesbüren, Tobias Heuer, Nikolai Maas, Peter Sanders, and Sebastian Schlag. Scalable High-Quality Hypergraph Partitioning. *ACM Transactions on Algorithms*, 20(1):9:1–9:54, 2024. doi:10.1145/3626527.

27 Lars Gottesbüren, Nikolai Maas, Dominik Rosch, Peter Sanders, and Daniel Seemaier. KaMinPar. Software, swhId: swh:1:dir:2ec38dbccb676136f71dc1796d6a06b450c48c6d (visited on 2025-09-03). URL: https://github.com/KaHIP/KaMinPar/commit/73eeaa2371c6826e166c9ca1383996f14d8c7a2a, doi:10.4230/artifacts.24666.

28 Lars Gottesbüren, Tobias Heuer, Peter Sanders, and Sebastian Schlag. Scalable Shared-Memory Hypergraph Partitioning. In *23st Workshop on Algorithm Engineering & Experiments (ALENEX)*, 2021. doi:10.1137/1.9781611976472.2.

29 Lars Gottesbüren, Tobias Heuer, Peter Sanders, Christian Schulz, and Daniel Seemaier. Deep Multilevel Graph Partitioning. In *29th European Symposium on Algorithms (ESA)*, pages 48:1–48:17, 2021. doi:10.4230/LIPIcs.ESA.2021.48.

30 Lars Gottesbüren, Nikolai Maas, Dominik Rosch, Peter Sanders, and Daniel Seemaier. Linear-time multilevel graph partitioning via edge sparsification, 2025. doi:10.48550/arXiv.2504.17615.

31 Milad Rezaei Hajidehi, Sraavan Sridhar, and Margo I. Seltzer. CUTTANA: Scalable Graph Partitioning for Faster Distributed Graph Databases and Analytics. *Proceedings of the VLDB Endowment*, 18(1):14–27, 2024. doi:10.14778/3696435.3696437.

32 Bruce Hendrickson and Robert Leland. A Multi-Level Algorithm For Partitioning Graphs. In *ACM/IEEE Conference on Supercomputing*, pages 28–es. ACM, 1995. doi:10.1145/224170.224228.

33 Tobias Heuer and Sebastian Schlag. Improving Coarsening Schemes for Hypergraph Partitioning by Exploiting Community Structure. In *16th International Symposium on Experimental Algorithms (SEA)*, pages 21:1–21:19, June 2017. doi:10.4230/LIPIcs.SEA.2017.21.

34 Artur Jez. Faster Fully Compressed Pattern Matching by Recompression. *ACM Transactions on Algorithms (TALG)*, 11(3):20:1–20:43, 2015. doi:10.1145/2631920.

35 George Karypis and Vipin Kumar. Analysis of Multilevel Graph Partitioning. In *ACM/IEEE Conference on Supercomputing*, pages 29–es. ACM, 1995. doi:10.1145/224170.224229.

36 George Karypis and Vipin Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. doi:10.1137/S1064827595287997.

37 Robert Krause, Lars Gottesbüren, and Nikolai Maas. Deterministic Parallel High-Quality Hypergraph Partitioning, 2025. doi:10.48550/arXiv.2504.12013.

38 Dominique Lasalle and George Karypis. Multi-threaded Graph Partitioning. In *27th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 225–236, 2013. doi:10.1109/IPDPS.2013.50.

39 Dominique LaSalle, Md. Mostofa Ali Patwary, Nadathur Satish, Narayanan Sundaram, Pradeep Dubey, and George Karypis. Improving Graph Partitioning for Modern Graphs and Architectures. In *5th Workshop on Irregular Applications - Architectures and Algorithms (IA3)*, pages 14:1–14:4. ACM, 2015. doi:10.1145/2833179.2833188.

40 Jure Leskovec and Christos Faloutsos. Sampling from Large Graphs. In *12th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 631–636. ACM, 2006. doi:10.1145/1150402.1150479.

41 Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1):2–es, 2007. doi:10.1145/1217299.1217301.

42 Gerd Lindner, Christian L. Staudt, Michael Hamann, Henning Meyerhenke, and Dorothea Wagner. Structure-Preserving Sparsification of Social Networks. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 448–454. ACM, 2015. doi:10.1145/2808797.2809313.

43 Nikolai Maas, Lars Gottesbüren, and Daniel Seemaier. Parallel Unconstrained Local Search for Partitioning Irregular Graphs. In *26st Workshop on Algorithm Engineering & Experiments (ALENEX)*, pages 32–45. SIAM, 2024. doi:10.1137/1.9781611977929.3.

44 Henning Meyerhenke, Peter Sanders, and Christian Schulz. Partitioning Complex Networks via Size-Constrained Clustering. In *13th International Symposium on Experimental Algorithms (SEA)*, pages 351–363. Springer, 2014. doi:10.1007/978-3-319-07959-2_30.

45 Joel C. Miller and Aric A. Hagberg. Efficient Generation of Networks with Given Expected Degrees. In *8th International Workshop on Algorithms and Models for the Web Graph (WAW)*, pages 115–126. Springer, 2011. doi:10.1007/978-3-642-21286-4_10.

46 Mark E. J. Newman and Michelle Girvan. Finding and Evaluating Community Structure in Networks. *Physical Review E*, 69, February 2004. doi:10.1103/PhysRevE.69.026113.

47 Chuck Pheatt. Intel Threading Building Blocks. *Journal of Computing Sciences in Colleges*, 23(4):298–298, 2008.

48 Semih Salihoglu and Jennifer Widom. GPS: A Graph Processing System. In *Conference on Scientific and Statistical Database Management (SSDBM)*, pages 22:1–22:12. ACM, 2013. doi:10.1145/2484838.2484843.

49 Daniel Salwasser, Daniel Seemaier, Lars Gottesbüren, and Peter Sanders. Tera-Scale Multilevel Graph Partitioning, 2024. doi:10.48550/arXiv.2410.19119.

50 George M. Slota, Kamesh Madduri, and Sivasankaran Rajamanickam. PuLP: Scalable Multi-Objective Multi-Constraint Partitioning for Small-World Networks. In *IEEE International Conference on Big Data*, pages 481–490. IEEE, 2014. doi:10.1109/BIGDATA.2014.7004265.

51 Daniel A. Spielman and Nikhil Srivastava. Graph Sparsification by Effective Resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. doi:10.1137/080734029.

52 Christian L. Staudt and Henning Meyerhenke. Engineering Parallel Algorithms for Community Detection in Massive Networks. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):171–184, 2016. doi:10.1109/TPDS.2015.2390633.

53 Charalampos E. Tsourakakis, Christos Gkantsidis, Bozidar Radunovic, and Milan Vojnovic. FENNEL: Streaming Graph Partitioning for Massive Scale Graphs. In *7th International*

Conference on Web Search and Data Mining (WSDM), pages 333–342. ACM, 2014. doi:10.1145/2556195.2556213.

54 Jianshu Weng and Bu-Sung Lee. Event Detection in Twitter. In *5th International Conference on Weblogs and Social Media*. AAAI, 2011. doi:10.1609/icwsm.v5i1.14102.

55 Tom Chao Zhou, Hao Ma, Michael R. Lyu, and Irwin King. UserRec: A User Recommendation Framework in Social Tagging Systems. In *24th AAAI Conference on Artificial Intelligence*, pages 1486–1491. AAAI, 2010. doi:10.1609/AAAI.V24I1.7524.

Better Indexing for Rectangular Pattern Matching

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland

Adam Górkiewicz
Institute of Computer Science, University of Wrocław, Poland

Abstract

We revisit the complexity of building, given a two-dimensional string of size n, an indexing structure that allows locating all k occurrences of a two-dimensional pattern of size m. While a structure of size $\mathcal{O}(n)$ with query time $\mathcal{O}(m+k)$ is known for this problem under the additional assumption that the pattern is a square [Giancarlo, SICOMP 1995], a popular belief was that for rectangular patterns one cannot achieve such (or even similar) bounds, due to a lower bound for a certain natural class of approaches [Giancarlo, WADS 1993]. We show that, in fact, it is possible to construct a very simple structure of size $\mathcal{O}(n \log n)$ that supports such queries for any rectangular pattern in $\mathcal{O}(m + k \log^{\varepsilon} n)$ time, for any $\varepsilon > 0$.

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

Keywords and phrases 2D strings, pattern matching, string indexing

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.33

Related Version *Full Version*: https://arxiv.org/abs/2508.17365

Funding Partially supported by the Polish National Science Centre grant number 2023/51/B/ST6/01505.

Acknowledgements We would like to thank Itai Boneh and Panagiotis Charalampopoulos for participating in the initial discussion.

1 Introduction

In the area of algorithms on strings, two basic algorithmic questions are pattern matching and string indexing. In the former, we aim to locate an occurrence of the pattern in the text, while in the latter the goal is to preprocess the text for multiple such queries. The complexity of both problems is well understood, at least for the case of regular strings and exact occurrences. In particular, a text of length n can be indexed in $\mathcal{O}(n)$ space, so that all k occurrences of a pattern of length m can be retrieved in $\mathcal{O}(m+k)$ time: this is a textbook application of suffix trees, already explained in the original article by Weiner [22].

In this paper, we consider two-dimensional strings, which are simply (two-dimensional) arrays of characters. Such a generalisation is naturally motivated by possible applications in image processing, and the complexity of pattern matching for two-dimensional strings has been already considered in the 70s [4, 5], and further investigated in the early 90s [1, 10]. A common assumption in all of these papers is that both the text and the pattern are square arrays, but this assumption is just to avoid clutter and can be removed without encountering any technical difficulties.

To state the results concerning indexing two-dimensional strings, let the dimensions of the text be $H \times W$, with the total size $n = HW$, and the dimensions of the pattern be $h \times w$, with the total size $m = hw$. Giancarlo [12] introduced the LSuffix tree of a matrix, based on a linearisation of a two-dimensional string (similar concept has been also used by Amir and Farach [2]). This allowed him to design an index of (asymptotically optimal) size $\mathcal{O}(n)$ that supports queries in $\mathcal{O}(m)$ time (for simplicity, we restate the bounds for constant alphabets), but only if the pattern is a square matrix. This assumption was

© Paweł Gawrychowski and Adam Górkiewicz;
licensed under Creative Commons License CC-BY 4.0

in fact inherent to his approach, as otherwise it is not clear how to linearise the strings. For the general case, he designed another structure, called the submatrix tree [11], of size $\mathcal{O}(\min(H, W)n)$ that supports queries in $\mathcal{O}(m)$ time. In the same paper, he defined an abstract notion of an index for a two-dimensional text, and proved that the size of any such index must be $\Omega(\min(H, W)n)$, making his construction essentially optimal (perhaps up to a logarithmic factor). Consequently, subsequent work focused on the case of square patterns [7, 9, 13–16, 18, 21].

Our contribution. We revisit the complexity of indexing a two-dimensional text for the general case of rectangular patterns. We observe that, in fact, the abstract notion of an index, as defined by Giancarlo [11], is somewhat restricted: he defines what it means for one pattern to be a prefix of another, and then requires that the index has a form of a compacted tree, with every node corresponding to some pattern occurring in the text, and the parent of each node corresponding to its prefix. This is consistent with the notion of one-dimensional suffix trees, but suffix trees are not the only known indexing structures. For example, suffix arrays [20] use $\mathcal{O}(n)$ space and allow retrieving the occurrences in $\mathcal{O}(m + \log n + k)$ time, even though they are not based on storing a compacted trie (although they are of course related to suffix arrays). We show that a very simple idea allows us to obtain the following result.

▶ **Theorem 1.** *For a two-dimensional text of size n, there is an $\mathcal{O}(n \log n)$-space data structure that, given a two-dimensional pattern of size m, reports all k occurrences of the pattern in the text in time $\mathcal{O}(m + k \log^{\varepsilon} n)$, for any constant $\varepsilon > 0$.*

Computational model. In the above theorem we assume the standard word RAM model with words of logarithmic (in the size of the input) length. Basic arithmetic operations on such words (and indirect addressing) are assumed to take constant time. Each character is assumed to fit in a single machine word. We measure the size of our data structures in the number of words.

2 Preliminaries

One-dimensional strings. A (one-dimensional) string is a sequence of characters. We index positions in a string starting from 1. For a string S of length n, we write $S[i]$ to denote its i-th character and $S[i\,..\,j]$ to denote the substring spanning positions i through j, inclusive. If only one endpoint is provided, we interpret the interval as a prefix or suffix: $S[..\,i] := S[1\,..\,i]$ is the prefix of length i; $S[i\,..] := S[i\,..\,n]$ is the suffix starting at position i.

Two-dimensional strings. We define two-dimensional strings as rectangular arrays of characters. We refer to the total number of characters in a two-dimensional string as its *size*. Rows and columns are indexed starting from 1. For a two-dimensional string S we write $S[i]$ to denote its i-th row, interpreted as a one-dimensional string. Consequently, $S[i][j]$ denotes the character in the i-th row and j-th column.

Meta-characters. In our algorithm, we reduce two-dimensional indexing to a collection of problems concerning one-dimensional strings. This is done by treating fixed-length fragments of rows of the two-dimensional text as atomic symbols, which we refer to as *meta-characters*. Formally, for a two-dimensional string S and a fixed width w, a meta-character is a substring of the form $S[i][j\,..\,j + w - 1]$. To effectively operate on such meta-characters, we will

represent each of them by an integer from $[\text{poly}(n)]$ that fits inside a single machine word[1], called the identifier. The identifiers of two meta-characters will be different if and only if the meta-characters themselves are different.

Compacted trie. We use the standard compacted trie, also known as a compressed trie or Patricia trie, for storing a set of strings. The strings consist of either characters or meta-characters. In either case, we assume that the symbols fit in a single machine word. A compacted trie built for a set of k strings is of size $\mathcal{O}(k)$, as we collapse maximal paths consisting of nodes with exactly one child into a single edge, thus the number of inner nodes is strictly smaller than the number of leaves. The remaining nodes are called explicit, while the dissolved nodes are called implicit.

Tools. We will need a few data structures. The first is called a deterministic dictionary.

▶ **Theorem 2** ([8]). *Given a set S of n keys, we can build a dictionary structure of size $\mathcal{O}(n)$ that allows constant-time access to any element of S (and its associated information).*

The second is a range reporting structure (we note that other trade-offs between the space and reporting time are possible, but we state only one of them to avoid clutter).

▶ **Theorem 3** ([6, Theorem 2.1]). *Given a collection of n points in $\{1,\ldots,n\}^2$, we can build a structure of size $\mathcal{O}(n)$ that, given an axis-aligned rectangle, reports all k points inside it in time $\mathcal{O}((1+k)\log^{\varepsilon} n)$, for any constant $\varepsilon > 0$.*

Finally, we need to implement *prefix search* on a compacted trie. That is, given a query string, we traverse the trie to determine whether it occurs as a prefix of any stored string. If so, we return the corresponding node (which may be implicit); otherwise, we report that no such prefix match exists.

▶ **Lemma 4.** *A compacted trie storing k strings uses $\mathcal{O}(k)$ additional space (on top of the strings themselves). Assuming constant-time read-only random access to any character, prefix search for a query string of length m takes $\mathcal{O}(m)$ time.*

Proof. To implement prefix search, we start at the root and descend following the appropriate edge. This requires storing all the edges outgoing from an explicit node in a dictionary structure, which we implement with Theorem 2. Since the total degree over all explicit nodes is $\mathcal{O}(k)$, the combined space used by all dictionaries is $\mathcal{O}(k)$. For implicit nodes, we only verify whether the subsequent characters of the query string are the same as on the edge, using constant-time read-only random access to one of the stored strings. Then, the time per each character of the query string is constant, so $\mathcal{O}(m)$ overall. ◀

3 Two-Dimensional Index Construction

In this section, we present our data structure for indexing a two-dimensional text and prove Theorem 1. The idea behind our approach is to reduce the two-dimensional pattern matching problem to a collection of one-dimensional problems. Each of these problems can then be solved efficiently by plugging in range reporting structures, as usual in the one-dimensional setting. Throughout this section, we write T to denote the $H \times W$ input text and P to denote the $h \times w$ query pattern, with $n = HW$ and $m = hw$ denoting their respective sizes.

[1] It is in fact enough that it fits in a constant number of machine words.

The data structure consists of two symmetric components, designed to handle the query depending on whether the pattern is taller or wider. Given an $h \times w$ pattern, we use one component if $h \geq w$ and the symmetric one (with rows and columns interchanged) otherwise. We describe how to handle tall patterns, i.e., those with $h \geq w$, as the other case is fully symmetric. To handle patterns of different widths, we store a separate index for each possible pattern width. More precisely, for each width $w \in \{1, \ldots, W\}$, we construct a dedicated data structure that supports searching for patterns of fixed width w and any height $h \geq w$.

For a fixed width w, we reduce the two-dimensional matching problem to a one-dimensional problem. Specifically, we interpret the pattern as a one-dimensional string of length h, where the i-th symbol is a *meta-character* corresponding to the i-th row $P[i]$ of the pattern, that is, a contiguous block of w characters from that row. As explained earlier, we will assign identifiers to these meta-characters, so that we can treat them as integers from $[\text{poly}(n)]$. We first explain how to ensure that we can access the identifier of any meta-character in the pattern and the text in constant time, after $\mathcal{O}(n \log n)$ space preprocessing of the text, and then define the one-dimensional problem.

Encoding the text. The construction follows the classical Karp-Miller-Rosenberg (KMR) approach [17], and is based on assigning integer identifiers to all fragments of the form $T[i][j\,..\,j+2^k-1]$, for all k such that $2^k \leq W$. For each such fragment, we define its identifier as the lexicographic rank of its substring among all distinct substrings of the same length, and store this value. Then, the identifier of $T[i][j\,..\,j+w-1]$ consists of the identifiers of two overlapping fragments of length $2^{\lfloor \log w \rfloor}$ that together cover the whole $T[i][j\,..\,j+w-1]$. Since there are $\mathcal{O}(n)$ fragments per length and $\mathcal{O}(\log W)$ relevant lengths, the total space required is $\mathcal{O}(n \log n)$.

Encoding the pattern. To encode a pattern row $P[i]$ of width w, we represent the corresponding meta-character by the pair of identifiers of its prefix and suffix of length 2^k, where $k = \lfloor \log w \rfloor$. To support this, during text preprocessing, we collect all distinct substrings $T[i][j\,..\,j+2^k-1]$ and store them in a compacted trie, where each leaf is labeled with the lexicographic rank among all substrings of T of length 2^k. We build a separate compacted trie for each relevant value of k; since each trie requires $\mathcal{O}(n)$ space and there are $\mathcal{O}(\log W)$ values of k, the total space used by all the compacted tries is $\mathcal{O}(n \log n)$. At query time, we extract the 2^k-length prefix and suffix of $P[i]$ and search for them in the corresponding compacted trie to obtain their identifiers. If either substring is not found, we report that the pattern does not occur in the text. Otherwise, in $\mathcal{O}(m)$ time we obtain the sought identifier for each row of the pattern.

One-dimensional problems. To complete the reduction, we must identify all occurrences of the one-dimensional meta-character pattern within the two-dimensional text. To that end, we consider all windows of w consecutive columns. Each such w-column strip defines a one-dimensional text of length H, where the j-th character is a meta-character corresponding to the block of w consecutive characters in the j-th row of the strip. Formally, to define the one-dimensional texts corresponding to each w-column strip, we fix a horizontal offset $i \in \{1, \ldots, W-w+1\}$ and define a one-dimensional text of length H consisting of the meta-characters $T[j][i\,..\,i+w-1]$ for all positions $j \in \{1, \ldots, H\}$. This gives a collection of $W-w+1$ one-dimensional texts over the same meta-character alphabet. Each occurrence of the meta-character pattern in one of these one-dimensional texts corresponds one-to-one to a two-dimensional occurrence of the original pattern in the text.

In each of these one-dimensional instances, the derived meta-character pattern has length $h \geq w$. We exploit this assumption when designing a (simple) indexing structure in Section 4.

▶ **Lemma 5.** *For a parameter w and a collection of one-dimensional texts of total length n, there is a data structure requiring $\mathcal{O}(n/w)$ additional space (on top of the strings) that, given a one-dimensional pattern of length $h \geq w$, reports all k occurrences of the pattern in the texts in time $\mathcal{O}(hw + (w+k)\log^{\varepsilon} n)$, for any constant $\varepsilon > 0$.*

We could apply Lemma 5 for each width $w \in \{1, \ldots, W\}$ to handle patterns of width w and height $h \geq w$. The total additional space is bounded by $\sum_{w=1}^{W} \mathcal{O}(n/w) = \mathcal{O}(n \log n)$, matching the requirements of Theorem 1. However, when h is smaller than $\log^{\varepsilon} n$, it holds that $w \log^{\varepsilon} n$ in the query time is larger than m. We thus handle the case of small w separately as follows.

▶ **Lemma 6.** *For a collection of one-dimensional texts of total length n, there is an $\mathcal{O}(n)$-space data structure that, given a one-dimensional pattern of any length h, reports all k occurrences of the pattern in the texts in time $\mathcal{O}(h + k)$.*

Proof. We store all suffixes of all input texts, each terminated with a distinct delimiter, in a compacted trie implemented with Lemma 4. The total number of suffixes is $\mathcal{O}(n)$, so the total required space is $\mathcal{O}(n)$. Querying for a pattern of length h is implemented by performing a prefix search for the pattern in the trie and reporting all leaves in the corresponding subtree. ◀

We apply Lemma 6 for every width $w \leq \log n$, and otherwise use Lemma 5. The total space is still $\mathcal{O}(n \log n)$. To bound the query time, we observe that whenever we use Lemma 6 the query time is $\mathcal{O}(h + k) = \mathcal{O}(m + k)$, and whenever we use Lemma 5 we can assume $w > \log n$ so the query time is $\mathcal{O}(hw + (w+k)\log^{\epsilon} n) = \mathcal{O}(m + k \log^{\epsilon} n)$.

4 One-Dimensional Index for Long Patterns

In this section, we prove Lemma 5 by designing an indexing structure for the one-dimensional setting that takes advantage of the assumption that the pattern is sufficiently long. We briefly comment that such a setup can be seen as related to locally consistent anchors, and such an assumption has been used in the literature [3]. In our case, a simple and self-contained approach based on defining some compacted tries and storing a range reporting structure is enough, though. The idea of using range searching to solve indexing is, of course, quite common in the literature [19].

Setup. We assume that the input texts allow constant-time read-only random access to the individual characters, and each of them fits in a single machine word. Throughout this section, we fix a parameter w, corresponding to the lower bound on the length of the pattern. Texts shorter than w can be discarded, as they cannot contain an occurrence of a pattern of length $h \geq w$.

Preprocessing. For each text T, we define a collection of *cuts*: positions between characters spaced at regular intervals of length w. Formally, we insert a cut at every position i such that $i \equiv 0 \pmod{w}$, including $i = 0$. Each cut partitions T into a prefix $T_1 = T[..i]$ and a suffix $T_2 = T[i+1..]$, where either may be empty. We associate the cut with the pair (T_1, T_2),

which serves as its representation. We organize the collection of cuts using a two-dimensional range reporting structure. Let $\mathcal{T}_1$ denote the set of all prefixes T_1 reversed, and let $\mathcal{T}_2$ denote the set of all suffixes T_2 extracted from the cuts. We build two compacted tries:

- $\mathcal{S}_1$, storing the strings in $\mathcal{T}_1$,
- $\mathcal{S}_2$, storing the strings in $\mathcal{T}_2$.

To ensure that each string from $\mathcal{T}_1$ and from $\mathcal{T}_2$ corresponds to a unique leaf in the respective trie, we conceptually prepend and append a distinct delimiter character to each prefix and suffix, respectively.

Next, we assign a pre-order number to each leaf of $\mathcal{S}_1$ and $\mathcal{S}_2$ via a depth-first traversal. Each cut then defines a point $(x, y) \in \mathbb{Z}^2$, where:

- x is the pre-order number of the leaf in $\mathcal{S}_1$ corresponding to the reversal of T_1,
- y is the pre-order number of the leaf in $\mathcal{S}_2$ corresponding to T_2.

This yields a collection of $\mathcal{O}(n/w)$ such points, one per cut. We store these points in an instance of Theorem 3.

Answering a query. Given a pattern P of length $h \geq w$, our goal is to report all of its occurrences in the input texts. Since the pattern has length at least w, any occurrence must span at least one cut position from the collection defined for the texts. We iterate over all positions at which such a cut could intersect an occurrence of the pattern. We refer to each such position within the pattern as an *anchor*, and consider only the first w possible anchors, to ensure that every occurrence of P is reported exactly once.

Specifically, we consider all positions $j \in \{0, 1, \ldots, w-1\}$ where the pattern may be anchored. Each such position induces a partition of P into a prefix $P_1 = P[..j]$ and a suffix $P_2 = P[j+1..]$. The task is to find and report all cuts in the texts that partition some text into (T_1, T_2) such that P_1 is a suffix of T_1 and P_2 is a prefix of T_2. This is implemented as follows. First, we extract the corresponding prefix-suffix pair (P_1, P_2) from the pattern. We then locate the (possibly implicit) node v_1 in $\mathcal{S}_1$ corresponding to the reversal of P_1, and the node v_2 in $\mathcal{S}_2$ corresponding to P_2. If either node does not exist, the current position cannot yield any occurrences and is skipped. Otherwise, let $[\ell_1, r_1]$ and $[\ell_2, r_2]$ be the ranges of pre-order numbers of leaves in the subtrees of v_1 and v_2, respectively.

The problem now reduces to a two-dimensional orthogonal range reporting query: report all stored points (x, y) that lie within the rectangle $[\ell_1, r_1] \times [\ell_2, r_2]$. Each reported point corresponds to a valid cut that certifies an occurrence of the pattern. Querying the instance of Theorem 3 yields the query bound claimed in Lemma 5.

References

1 Amihood Amir, Gary Benson, and Martin Farach. An alphabet independent approach to two-dimensional pattern matching. *SIAM J. Comput.*, 23(2):313–323, 1994. doi:10.1137/S0097539792226321.

2 Amihood Amir and Martin Farach. Two-dimensional dictionary matching. *Inf. Process. Lett.*, 44(5):233–239, 1992. doi:10.1016/0020-0190(92)90206-B.

3 Lorraine A. K. Ayad, Grigorios Loukides, and Solon P. Pissis. Text indexing for long patterns: Anchors are all you need. *Proc. VLDB Endow.*, 16(9):2117–2131, 2023. doi:10.14778/3598581.3598586.

4 Theodore P. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM J. Comput.*, 7(4):533–541, 1978. doi:10.1137/0207043.

5 Richard S. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6(5):168–170, 1977. doi:10.1016/0020-0190(77)90017-5.

6 Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *SCG*, pages 1–10. ACM, 2011. doi:10.1145/1998196.1998198.

7 Ying Choi and Tak Wah Lam. Dynamic suffix tree and two-dimensional texts management. *Inf. Process. Lett.*, 61(4):213–220, 1997. doi:10.1016/S0020-0190(97)00018-5.

8 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

9 Kimmo Fredriksson, Gonzalo Navarro, and Esko Ukkonen. An index for two dimensional string matching allowing rotations. In *IFIP TCS*, volume 1872 of *Lecture Notes in Computer Science*, pages 59–75. Springer, 2000. doi:10.1007/3-540-44929-9_5.

10 Zvi Galil and Kunsoo Park. Alphabet-independent two-dimensional witness computation. *SIAM J. Comput.*, 25(5):907–935, 1996. doi:10.1137/S0097539792241941.

11 Raffaele Giancarlo. An index data structure for matrices, with applications to fast two-dimensional pattern matching. In *WADS*, volume 709 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 1993. doi:10.1007/3-540-57155-8_260.

12 Raffaele Giancarlo. A generalization of the suffix tree to square matrices, with applications. *SIAM J. Comput.*, 24(3):520–562, 1995. doi:10.1137/S0097539792231982.

13 Raffaele Giancarlo and Roberto Grossi. Parallel construction and query of suffix trees for two-dimensional matrices. In *SPAA*, pages 86–97. ACM, 1993. doi:10.1145/165231.165243.

14 Raffaele Giancarlo and Roberto Grossi. On the construction of classes of suffix trees for square matrices: Algorithms and applications. In *ICALP*, volume 944 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 1995. doi:10.1007/3-540-60084-1_67.

15 Raffaele Giancarlo and Roberto Grossi. Parallel construction and query of index data structures for pattern matching on square matrices. *J. Complex.*, 15(1):30–71, 1999. doi:10.1006/JCOM.1998.0496.

16 Raffaele Giancarlo and Daniela Guaiana. On-line construction of two-dimensional suffix trees. *J. Complex.*, 15(1):72–127, 1999. doi:10.1006/JCOM.1998.0495.

17 Richard M. Karp, Raymond E. Miller, and Arnold L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *STOC*, pages 125–136. ACM, 1972. doi:10.1145/800152.804905.

18 Dong Kyue Kim, Joong Chae Na, Jeong Seop Sim, and Kunsoo Park. Linear-time construction of two-dimensional suffix trees. *Algorithmica*, 59(2):269–297, 2011. doi:10.1007/S00453-009-9350-Z.

19 Moshe Lewenstein. Orthogonal range searching for text indexing. In *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of *Lecture Notes in Computer Science*, pages 267–302. Springer, 2013. doi:10.1007/978-3-642-40273-9_18.

20 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.

21 Joong Chae Na, Raffaele Giancarlo, and Kunsoo Park. On-line construction of two-dimensional suffix trees in $O(n^2 \log n)$ time. *Algorithmica*, 48(2):173–186, 2007. doi:10.1007/S00453-007-0063-X.

22 Peter Weiner. Linear pattern matching algorithms. In *SWAT*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.

On the Approximability of Train Routing and the Min-Max Disjoint Paths Problem

Umang Bhaskar
TIFR Mumbai, India

Katharina Eickhoff
RWTH Aachen University, Germany

Lennart Kauther
RWTH Aachen University, Germany

Jannik Matuschke
KU Leuven, Belgium

Britta Peis
RWTH Aachen University, Germany

Laura Vargas Koch
RWTH Aachen University, Germany

Abstract

In train routing, the headway is the minimum distance that must be maintained between successive trains for safety and robustness. We introduce a model for train routing that requires a fixed headway to be maintained between trains, and study the problem of minimizing the makespan, i.e., the arrival time of the last train, in a single-source single-sink network. For this problem, we first show that there exists an optimal solution where trains move in convoys – that is, the optimal paths for any two trains are either the same or are arc-disjoint. Via this insight, we are able to reduce the approximability of our train routing problem to that of the min-max disjoint paths problem, which asks for a collection of disjoint paths where the maximum length of any path in the collection is as small as possible.

While min-max disjoint paths inherits a strong inapproximability result on directed acyclic graphs from the multi-level bottleneck assignment problem, we show that a natural greedy composition approach yields a logarithmic approximation in the number of disjoint paths for series-parallel graphs. We also present an alternative analysis of this approach that yields a guarantee depending on how often the decomposition tree of the series-parallel graph alternates between series and parallel compositions on any root-leaf path.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases Train Routing, Scheduling, Approximation Algorithms, Flows over Time, Min-Max Disjoint Paths

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.34

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2507.03687` [1]

Funding *Umang Bhaskar*: supported by the Department of Atomic Energy, Government of India, under Project No. RTI4001.
Katharina Eickhoff: supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 2236/2.
Lennart Kauther: supported by the German Centre for Rail Traffic Research (DZSF) at the Federal Railway Authority under research project "Methodik der Kapazitätsbewertung des Gesamtsystems und Knotenberechnung".
Jannik Matuschke: supported by Fonds Wetenschappelijk Onderzoek (FWO) under Research Project G072520N "Optimization and Analytics for Stochastic and Robust Project Scheduling".



© Umang Bhaskar, Katharina Eickhoff, Lennart Kauther, Jannik Matuschke, Britta Peis, and Laura Vargas Koch;
licensed under Creative Commons License CC-BY 4.0



Acknowledgements We thank Daniel Schmand, Khai Van Tran and Andreas Wiese for fruitful discussions, as well as Nils Nießen and his group for providing us with foundational insights into train planning. We also thank the participants of the Dagstuhl Seminar 24281 [8], the Oberwolfach Workshop in Combinatorial Optimization 2024 [25], and the Chile Summer Workshop on Combinatorial Optimization 2025 for many interesting comments and questions. Finally, we thank the three anonymous reviewers for their careful reading of our paper and many useful suggestions.

1 Introduction

Given a rail network and a set of trains, the problem of train routing and scheduling refers to the problem of determining a route as well as a schedule for the trains to optimise some objective, such as the total travel time or the maximum travel time (also denoted as makespan). Train routing usually involves a number of real-world constraints, including train acceleration and deceleration, stopping times at stations, bidirectional arcs, presence of other trains, etc. Practical approaches to solving these complex problems often include mixed integer programming, heuristics, and simulations [3, 4, 15, 16, 22, 24], see [21, 22, 5] for surveys and an overview. However, theoretical guarantees on solutions are hard to come by, apart from some results that show NP-hardness already for some very restricted cases [15, 19].

One of the most important constraints in train routing and scheduling is that of *headway* [6, 18], which determines the minimum gap between successive trains on a link. The headway thus corresponds to a safety distance between two successive trains and is important for at least two different reasons. The first is safety: Maintaining sufficient headway between trains reduces the chances of a collision between trains. The second reason is robustness: If a train is delayed somewhat, then the headway can help ensure that this delay is not propagated to all the following trains, and the schedule can recover quickly. In fact, the importance of maintaining a headway is not restricted to train routing. In vehicular traffic, such as cars on roads, traffic often naturally forms a *platoon* – a group of vehicles following each other closely. Here, headway is again essential in maintaining safety of vehicles in the platoon, see [26] for a survey.

In this paper, our objective is understanding the impact of headway on the complexity of train routing and scheduling.[1] While numerous papers study efficient algorithms for both dynamic and static routing problems, research on efficient algorithms that account for headway is scarce. Towards this, we introduce a basic model of train routing with an exogenously defined headway requirement Δ. In our model, we are given a directed network $D = (V, A)$, where each arc a has a travel time $\tau_a \in \mathbb{Z}_{\geq 0}$, and d trains that must travel from a source s to a sink t. Headway constraints require a minimum time separation of Δ between any two trains using the same arc. Our objective is minimising the makespan – the time the last train reaches the sink t. We call this problem Train Makespan Optimization (TMO) and refer to Section 1.1 for a formal description.

Two interesting special cases of TMO arise when the headway is either very large ($\Delta = \infty$) or very small ($\Delta = 1$). In the former case, no arc can be used by two trains, and hence the paths used by the trains must be pairwise arc-disjoint. The problem for $\Delta = \infty$ thus becomes the static problem of finding d arc-disjoint s-t-paths while minimizing the length of the longest path. This is known in the literature as the Min-Max Disjoint Paths (MinMaxDP) problem [20]. In particular, the temporal dynamics, i.e., the scheduling aspect of the problem,

[1] Although, as noted, headway plays an important role in routing other vehicles as well, we use the nomenclature for trains as this is what initially motivated our work.

become irrelevant. Conversely, if $\Delta = 1$, the problem is equivalent to finding an integral *quickest flow over time*, a problem widely studied in traffic networks and transportation [28]. Hence, as Δ reduces, temporal dynamics play a larger role.

The quickest flow over time problem with aggregated arc capacities [11] generalizes the quickest flow over time problem. Here, only a limited amount of flow is allowed to traverse an arc in each time interval of length Δ. By setting these capacities to one, TMO can be seen as an integral special case of the quickest flow over time problem with aggregated arc capacities.

When studying the computational complexity of the problem from a theoretical perspective, one should note that, for large d, the size of the input is $\log(d)$ rather than d. Hence, a polynomial time algorithm must run in time polynomial in $\log(d)$ (and the other parameters). Note that this is not even sufficient time to describe a path for each train. However, we show that optimal solutions can be represented efficiently, by establishing the existence of an optimal *convoy* solution to TMO that can be described by a polynomial number of different paths and does not require waiting on any intermediate nodes. Furthermore, we show that approximating TMO, for any value of Δ, is equivalent to approximating MinMaxDP. We further develop new approximation algorithms for MinMaxDP, which translate to bounds for TMO as well. This work thus provides new results on a classical routing problem and introduces a new model to capture an important parameter in traffic routing and scheduling.

1.1 Model

We are given a directed graph $D = (V, A)$ with travel times $\tau_a \in \mathbb{Z}_{\geq 0}$ for every $a \in A$, a source $s \in V$ as well as a sink $t \in V$. We say $|A| = m$. Furthermore, we are given a minimum headway time $\Delta \in \mathbb{Z}_{\geq 1}$ (between the heads of the trains) and a number d of identical trains which we want to send from s to t. An instance of train routing is thus denoted (D, τ, d, Δ). In the following, we use $[k] \coloneqq \{1, \dots, k\}$ for $k \in \mathbb{Z}_{\geq 1}$ and $\delta^-(v)$ to denote the set of all arcs entering a node $v \in V$.

A *train routing* $(\boldsymbol{P}, \boldsymbol{\lambda})$ is a tuple consisting of a collection $\boldsymbol{P} = (P_i)_{i \in [d]}$ of s-t-paths P_i, where P_i consists of the sequence of arcs train i traverses on its way from s to t, together with a collection $\boldsymbol{\lambda} = (\lambda_i)_{i \in [d]}$ of *entry functions* $\lambda_i : P_i \to \mathbb{Z}_{\geq 0}$, where $\lambda_i(a)$ denotes the time when the nose of train i enters arc $a \in P_i$.

A train routing $(\boldsymbol{P}, \boldsymbol{\lambda})$ is *feasible* if it complies with the following two constraints:

- $\lambda_i(a) + \tau_a \leq \lambda_i(a')$ for all $i \in [d]$ and any pair of consecutive arcs a, a' on P_i,
- $|\lambda_i(a) - \lambda_{i'}(a)| \geq \Delta$ for all $i, i' \in [d]$ with $i \neq i'$ and $a \in P_i \cap P_{i'}$.

The first constraint implies that the entry function is consistent with the travel times along the path of each train. Note that this allows trains to wait at any node. The second constraint ensures that a feasible routing respects the minimum headway requirement, i.e., if two trains use the same arc, they can only do so with a temporal distance of at least Δ.

Given a feasible train routing $(\boldsymbol{P}, \boldsymbol{\lambda})$, we denote by $C_i(P_i, \lambda_i) \coloneqq \max_{a \in P_i}(\lambda_i(a) + \tau_a)$ the time when the nose of train i arrives at t. The *makespan* $C_{\max}(\boldsymbol{P}, \boldsymbol{\lambda}) \coloneqq \max_{i \in [d]} C_i(P_i, \lambda_i)$ is the time when the nose of the latest train reaches the sink t. We consider the Train Makespan Optimization (TMO) problem, where the objective is to find a feasible train routing that minimises the arrival of the last train at t. That is, we want to solve

$$\min C_{\max}(\boldsymbol{P}, \boldsymbol{\lambda}) \quad \text{s.t. } (\boldsymbol{P}, \boldsymbol{\lambda}) \text{ is a feasible train routing.}$$

1.2 The min-max disjoint paths problem

An interesting special case of TMO arises when Δ is sufficiently large compared to the travel times. In that case any (even approximately) optimal train routing sends all trains along disjoint paths – assuming that the network supports d disjoint s-t-paths. This leads us to the Min-Max Disjoint Paths (MinMaxDP) problem, which can be formalized as follows: Given a directed graph $D = (V, A)$ with two designated vertices $s, t \in V$ and travel times $\tau : A \to \mathbb{Z}_{\geq 0}$, find k pairwise arc-disjoint s-t-paths $P_1, \ldots, P_k$ minimizing the maximum travel time (i.e., makespan) $\max_{i \in [k]} \tau(P_i)$, where we use $\tau(P) \coloneqq \sum_{a \in P} \tau_a$ to shorten notation for each path P. We call $\boldsymbol{P} = (P_1, \ldots, P_k)$ a *path profile*.

Early work on MinMaxDP has focused on the case where k is constant. Indeed, Li et al. [20] – in the first paper that studied MinMaxDP – showed that, even when $k = 2$, the problem is strongly NP-hard for general digraphs and weakly NP-hard for directed acyclic graphs (DAGs). On the positive side, they also showed that MinMaxDP can be solved in pseudo-polynomial on DAGs when k is constant. By standard rounding techniques, this result can be turned into an FPTAS [13] for the same setting.

Special cases of MinMaxDP when k is part of the input have been studied under various names. Most notably, the Multi-level Bottleneck Assignment (MLBA) problem is equivalent to a special case of MinMaxDP in a DAG consisting of b layers. Dokka and Goerigk [10] recently established a strong inapproximability bound of $\Omega(b)$ for MLBA. In the full version of this article [1, Appendix A], we observe that their construction also implies that MinMaxDP (and, by extension, TMO) in DAGs does not admit approximation factors significantly better than logarithmic in k, unless every problem in NP can be solved in quasi-polynomial time (violating the exponential time hypothesis). This leads to the following theorem.

▶ **Theorem 1.** *There is no $\log^{1-\varepsilon}(k)$-approximation for MinMaxDP in DAGs for any $\varepsilon > 0$, unless NP $\subseteq$ QP.*

Moreover, MinMaxDP in bundle graphs, i.e., digraphs consisting of a sequence of parallel arcs, is equivalent to the so-called *complete* case of MLBA, from which it also inherits strong NP-hardness. For this problem, a long line of approximation algorithms starting in the 1980s [7, 17] has recently culminated in a PTAS by Das and Wiese [9] (formulated for an equivalent scheduling problem).

1.3 Contributions and overview

In the following, we provide an overview of our main contributions and the techniques used to derive these results.

Existence of optimal convoy routings. We show that any instance of TMO has an optimal solution in which any two trains either travel along the same path (with sufficient temporal distance), or follow disjoint paths (Theorem 2 in Section 2). We call such solutions *convoy routings*. In particular, convoy routings have a compact representation that is polynomial in the size of the input, even when the number of trains is exponential in the size of the network. A further consequence is that allowing trains to wait at intermediate nodes does not help to improve the optimal solution value. The proof of this result relies on an uncrossing algorithm that takes an arbitrary train routing and modifies it to ensure that any train that is the first to traverse the final arc of its path is also the first to traverse any other arc it uses. Conflicts, i.e., situations where this property is violated, are resolved iteratively, starting with those closest to the sink, with a potential function argument ensuring eventual termination of the procedure.

A flow-based additive approximation for TMO. We further provide a simple flow-based approximation algorithm for TMO that returns a convoy routing whose makespan is bounded by OPT $+\Delta$ (Theorem 4 in Section 3). This result is achieved by solving an appropriately defined quickest flow over time problem in the underlying network. Note that in practice, the headway Δ is typically much smaller than the total transit time of a train from source to sink, and hence the additive error Δ is small compared to OPT.

Reduction to MinMaxDP. Building on the previous two results, we establish a strong connection between MinMaxDP and TMO by showing that any α-approximation algorithm for MinMaxDP yields a $\max\{1+\varepsilon, \alpha\}$-approximation algorithm for TMO with polynomial runtime in the size of the input and $\frac{1}{\varepsilon}$ (Theorem 5 in Section 4). To achieve this result, we devise a gadget that encodes the choices on the number of trains on each path of a convoy routing into the routing decisions in the network. The size of this gadget is polynomial in the size of the original network and d. It thus provides an exact reduction of TMO to MinMaxDP when the number of trains d is polynomial in the size of the network. When d is large, on the other hand, there must be an arc traversed by a large number of trains and hence OPT is much larger than Δ. In this case, the additive guarantee of OPT $+\Delta$ of our flow-based algorithm for TMO mentioned above translates to a $(1+\varepsilon)$-approximation. As we mentioned previously, for sufficiently large values of Δ, TMO coincides with MinMaxDP. Thus, finding good approximation algorithms for MinMaxDP is not only sufficient but also necessary to obtain good approximations for TMO.

Approximation algorithms for MinMaxDP in series-parallel networks. In the remainder of this paper, we therefore focus on approximation algorithms for MinMaxDP. We show that a natural greedy approach achieves an approximation which is logarithmic in k on series-parallel (SePa) networks (Theorem 8 in Section 5), where k is the number of paths to be constructed. This constitutes the first non-trivial approximation result for MinMaxDP going beyond the aforementioned PTAS for bundle graphs [9], without assuming that either k or the number of layers of the graph is constant.

We prove the approximation guarantee via induction on a series-parallel decomposition of the graph. The key ingredient is showing that the greedy algorithm maintains a solution fulfilling a strong balance condition on the path lengths. Additionally, we provide an alternative analysis of the same approach that yields an upper bound of $\varphi(D)+1$ on the approximation factor, where $\varphi(D)$ denotes the number of changes from series to parallel composition on a root-leaf path in the binary tree describing the SePa-graph. To the best of our knowledge, the parameter $\varphi(D)$, which we demonstrate to be independent of the choice of the decomposition tree, has not been used in the literature before. We believe that this natural parameter can be of independent interest for designing and analysing algorithms for other problems in SePa-graphs. Our alternative analysis implies the classic greedy 2-approximation for bundle graphs by Hsu [17] as a special case. By the aforementioned reduction of TMO to MinMaxDP, the approximation results for MinMaxDP on SePa-graphs, translate to our train routing problem.

2 Existence of optimal convoy routings

Let (D, τ, d, Δ) be an instance of TMO. A *convoy routing* consists of a collection of $k \leq d$ pairwise arc-disjoint s-t-paths $\boldsymbol{P} = (P_1, \dots, P_k)$ in D, together with a vector $\boldsymbol{\sigma} \in \mathbb{Z}^k_{\geq 1}$ with $\sum_{i=1}^k \sigma_i = d$, whose i-th component σ_i denotes the number of trains using path P_i. Note that any such convoy routing $(\boldsymbol{P}, \boldsymbol{\sigma})$ can be turned into a (feasible) train routing $(\boldsymbol{P}, \boldsymbol{\lambda})$ by sending

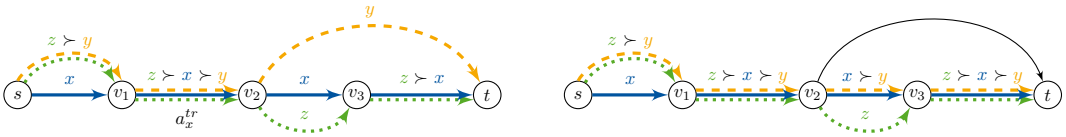


Figure 1 An example iteration of the uncrossing phase. Here $x \succ y$ means that x follows y. Train $x \in L$ is chosen by the uncrossing algorithm. As train y precedes x on a_x^{tr}, it is rerouted to use the same subpath as x from a_x^{tr} to t. The routing after this iteration is shown on the right.

σ_i trains along path P_i in single file with temporal spacing Δ between any two consecutive trains. Thus, the makespan of this train routing is $\max_{i\in[k]} \{\tau(P_i) + (\sigma_i - 1) \cdot \Delta\}$, where $\tau(P_i) := \sum_{a\in P_i} \tau_a$.

▶ **Theorem 2** (Convoy theorem). *Every instance of TMO admits an optimal solution that is a convoy routing.*

Our proof of Theorem 2 is constructive. Note that, since we allow waiting at nodes, there is always an optimal train routing $(\boldsymbol{P}, \boldsymbol{\lambda})$ which is acyclic in the sense that none of the paths contains a cycle. In order to show that any instance of TMO admits an optimal routing which is a convoy solution (i.e., to prove Theorem 2), we take an optimal acyclic train routing $(\boldsymbol{P}, \boldsymbol{\lambda})$, and modify this train routing step by step without increasing the makespan until the routing corresponds to a convoy routing.

The algorithm, which we from now on call **uncrossing algorithm**, proceeds in two phases. First, in the *uncrossing phase*, we reroute trains such that at the end of the phase, the trains that are the first ones to use their respective ingoing arc to the sink t, travel along disjoint paths. We call these trains *leaders*, and the remaining trains *non-leaders*. Afterwards, in the *assignment phase*, we assign every non-leader train a route based on the arc it uses to enter t. In particular, each such train follows its leader from s to t.

We now outline the algorithm in more detail and briefly sketch its correctness. A detailed description can be found in the full version of this article [1, Appendix B].

2.1 The uncrossing phase

We first introduce some notation. For an arc a entering t, i.e., $a \in \delta^-(t)$, we define the *leader* L_a on a as the first train using arc a. We denote by $L = \{L_a\}_{a\in\delta^-(t)}$ the set of leaders. We say that a leader x *has a conflict on arc* $a \in A$ with another train y if y uses some arc a before x. Note that this relation is not symmetric. For a leader $x \in L$, we define the *transition arc* a_x^{tr} as the first arc on which a conflict of x with another train exists when traversing P_x from t to s, i.e., in the reverse direction. Note that a_x^{tr} might not exist, in which case x is the first train on every arc on its path.

In the uncrossing phase, we pick a leader x for which a_x^{tr} exists. We reroute all trains preceding x on a_x^{tr} such that, after rerouting, they all use the same subpath as x from a_x^{tr} to t. The rerouted trains traverse every arc of this a_x^{tr}-t-path with the same temporal distance to train x as they had on arc a_x^{tr}. Hence, the rerouted trains keep their respective pairwise temporal distances on the entire path from a_x^{tr} to t and no new conflicts among the rerouted trains arise. Figure 1 shows an example of such a rerouting step. Note that this rerouting procedure also cannot create new conflicts with other trains as x was the first train on every arc of P_x from a_x^{tr} to t. Hence, the new routing is feasible and does not increase the makespan as x arrives after any rerouted train and the routing of x remains unchanged.

We iterate this procedure with the updated train routing (and the corresponding leaders and transition arcs) until every leader is the first train on all arcs of its path, which implies that all leaders use disjoint paths. In the proof of Lemma 3 below, we establish that this termination criterion is reached after a finite number of uncrossing steps.

2.2 The assignment phase

After successfully uncrossing the paths of all leaders, we receive an updated routing $(\boldsymbol{P}, \boldsymbol{\lambda})$. To achieve a convoy routing $(\boldsymbol{R}, \boldsymbol{\sigma})$, define $\boldsymbol{R} = (P_x)_{x \in L}$ where P_x is the s-t-path used by leader x in $\boldsymbol{P}$. We assign all trains that use the same arc as x to enter t in $(\boldsymbol{P}, \boldsymbol{\lambda})$ to P_x and define σ_x as the resulting number of trains that use P_x. As each train uses exactly one arc in $\delta^-(t)$, this induces a partition of $[d]$ and thus a feasible convoy routing.

To see why the makespan remains unchanged, note the following: Fix an arc $a \in \delta^-(t) \cap P_x$ for some $x \in L$. The first train from our convoy routing $(\boldsymbol{R}, \boldsymbol{\sigma})$ arrives at a no later than the corresponding leader x in $(\boldsymbol{P}, \boldsymbol{\lambda})$. This is because they use the same path, and the train from $(\boldsymbol{R}, \boldsymbol{\sigma})$ departs s immediately. Further, in both routings, the last train entering a must have a minimum temporal distance of $(\sigma_x - 1) \cdot \Delta$ to the first train entering a. In the convoy routing, this bound is tight and thus the assignment phase does not increase the makespan.

2.3 Analysis

By its construction, if the uncrossing algorithm terminates, it returns a convoy routing with a makespan no worse the one of the input routing. Theorem 2 thus follows as a consequence of the following lemma.

▶ **Lemma 3.** *The uncrossing algorithm terminates.*

Proof (sketch). We prove this using a potential function. To define this function, we extend the definition of a_y^{tr} to arbitrary, i.e., also non-leader trains, y. Informally, a_y^{tr} is the last arc on train y's path after which the set (or order) of trains driving before y changes.

The potential of a train y is the number of arcs of P_y from s to a_y^{tr}. We prove the following two statements: (i) When resolving a conflict for some leader x, we decrease its potential, (ii) The potential of all other trains does not increase when resolving a conflict of x. Together, (i) and (ii) imply that the sum of all potentials decreases in every iteration of the uncrossing phase, implying finite termination of this phase and thus of the entire algorithm. The proof of (i) and (ii) involves a careful analysis, which reveals that the transition node of a train is always on its path in the original routing, and a case distinction of possible configurations arising in the uncrossing steps. The details of this analysis can be found in the full version of this article [1, Appendix B]. ◀

3 A flow-based additive approximation for TMO

As mentioned in Section 1, TMO with $\Delta = 1$ reduces to Quickest Flow Over Time in a network with unit capacities. This problem asks for a flow over time of a given value (d units in our case) to be sent from a source to a sink in the shortest time T possible. For an in-depth discussion of flows over time, we refer to [28]. By guessing the optimal makespan T (e.g., by using parametric search [27]), Quickest Flow Over Time can, in turn, be reduced to Maximum Flow Over Time – i.e., sending a maximum amount of flow within a given time horizon. The latter problem can be solved efficiently using a classic result of Ford and Fulkerson [14]: Compute an appropriate static flow, decompose this flow into (disjoint, in the case of unit capacities) paths, and send one unit of flow along each path as long as it still reaches the sink within the time horizon.

A natural extension of this approach to the case $\Delta > 1$ is computing a quickest flow over time sending $\Delta \cdot d$ units in total, and transform it into a train routing by only sending one unit of flow (i.e., one train) every Δ time units along each path. Because, conversely, any routing of d trains also can be extended to a flow over time of value $\Delta \cdot d$ when allowing an extra period of Δ time units beyond its makespan, we obtain the following approximation guarantee for TMO. Again, a detailed proof can be found in the full version of this article [1, Appendix C].

▶ **Theorem 4.** *There is an algorithm that, given a TMO instance, computes in strongly polynomial time a convoy routing with makespan at most* $\mathrm{OPT} + \Delta$*, where* OPT *is the optimal makespan.*

4 Reduction to MinMaxDP

In this section, we examine the relationship between TMO and MinMaxDP. It is easy to see that any α-approximation for TMO implies an α-approximation for MinMaxDP, by choosing $\Delta = \alpha \cdot \sum_{a \in A} \tau_a$. Proving the converse is more involved. We use a bundle graph (see Figure 2) consisting of a sequence of parallel arcs as a gadget, attached to the source node for the MinMaxDP instance, for our reduction.

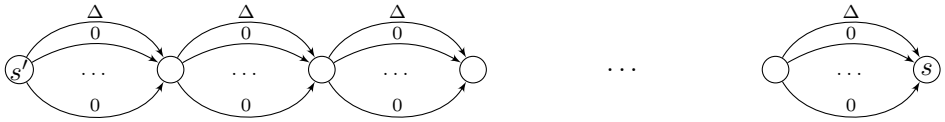


Figure 2 The gadget G_k for solving TMO via MinMaxDP. The gadget consists of a sequence of $(d - k)$ bundles. Each bundle contains an arc with travel time Δ and $k - 1$ arcs with travel time 0.

▶ **Theorem 5.** *Let $\mathcal{A}$ be an α-approximation algorithm for MinMaxDP, and let $\mathcal{R}(m)$ denote its runtime dependent on $m := |A|$. There exists a $\max\{\alpha, (1+\varepsilon)\}$-approximation for TMO with runtime in $\mathcal{O}\left(m \cdot \mathcal{R}\left(m^2(2 + \frac{1}{\varepsilon})\right)\right)$.*

Proof (sketch). We defer a formal proof to the full version of this article [1, Appendix D] and restrict ourselves to providing its intuition for now. First, consider the case $d > m(1 + \frac{1}{\varepsilon})$, i.e., when d is large. Then, OPT is much greater than Δ because some arcs are used by more than $\lceil \frac{1}{\varepsilon} \rceil$ trains, and hence the flow-based approximation from Section 3 yields a $(1 + \varepsilon)$-approximate solution.

Otherwise, when $d \leq m(1 + \frac{1}{\varepsilon})$, we utilise the given α-approximation $\mathcal{A}$ for MinMaxDP. We guess the number $k \in [m]$ of disjoint paths in an optimal convoy routing. For that k, we construct an auxiliary digraph D'_k by adding the gadget G_k described in Figure 2 prior to the source node s of the given digraph D. It is easy to see that the minimum makespan for MinMaxDP on D'_k (when k was guessed correctly) and the minimum makespan for the original TMO-instance coincide. Hence, $\mathcal{A}$'s approximation guarantee carries over to TMO. Note that the above case distinction is necessary to bound the size of D'_k polynomially. ◀

▶ Remark 6. Let $\mathcal{C}$ be a class of digraphs that is closed under adding a bundle graph prior to their source s. If $\mathcal{A}$ is an α-approximation algorithm for MinMaxDP on graph class $\mathcal{C}$, then Theorem 5 yields an approximation algorithm for TMO on graph class $\mathcal{C}$.

Bundle graphs and SePa-graphs are examples of such closed classes. Theorem 5 in conjunction with Theorem 1 from [9] yields the following stronger result for bundle graphs.

▶ **Corollary 7.** *There is a PTAS for TMO on bundle graphs.*

5 Approximation algorithm for MinMaxDP in series-parallel graphs

In this section, we describe and analyse an algorithm for approximating MinMaxDP in series-parallel graphs. Before we state our main result, we first provide a formal definition of series-parallel graphs and the parameter $\varphi(D)$ that is part of the approximation guarantee.

Series-parallel graphs. A *series-parallel digraph* $D = (V, A)$ (*SePa-graph*, for short) has a source s and a sink t and consists of either a single arc or can be obtained by a series or parallel composition of two series-parallel subgraphs D' and D''. The series composition $D = D' \odot_s D''$ merges the sink of D' with the source of D'', such that the source of D is the source of D' and the sink of D is the sink of D''. The parallel composition $D = D' \odot_p D''$ merges the two sources of D' and D'' to a single source of D and merges the sinks of D' and D'' to a single sink of D. Note that it is possible to have parallel arcs between two nodes.

The parameter $\varphi(D)$. Any SePa-graph can be described by a *binary decomposition tree*, where the leaves of the tree correspond to the arcs and the internal vertices correspond to either the series (labeled Ⓢ) or the parallel (labeled Ⓟ) composition of their children; see Figure 3 for an example. We can split the tree into arc-, S- or P-components. Such a component is a maximal connected subset of vertices in the binary decomposition tree with the same label. The *root* of a component C is the (unique) vertex $v \in C$ that is closest to the root of the binary decomposition tree. Moreover, we call the vertices that are not in the component C but whose parents are, the children of the component C. We let $\varphi(D)$ denote the maximum number of traversed S-components on a path from the root to a leaf. Note that, even if the binary decomposition tree of a SePa-graph is not unique, the number $\varphi(D)$ is (see full version [1, Appendix E]). Throughout this section, we assume that we are given a fixed binary decomposition tree of the input graph D (such a tree can be constructed in linear time [2]).

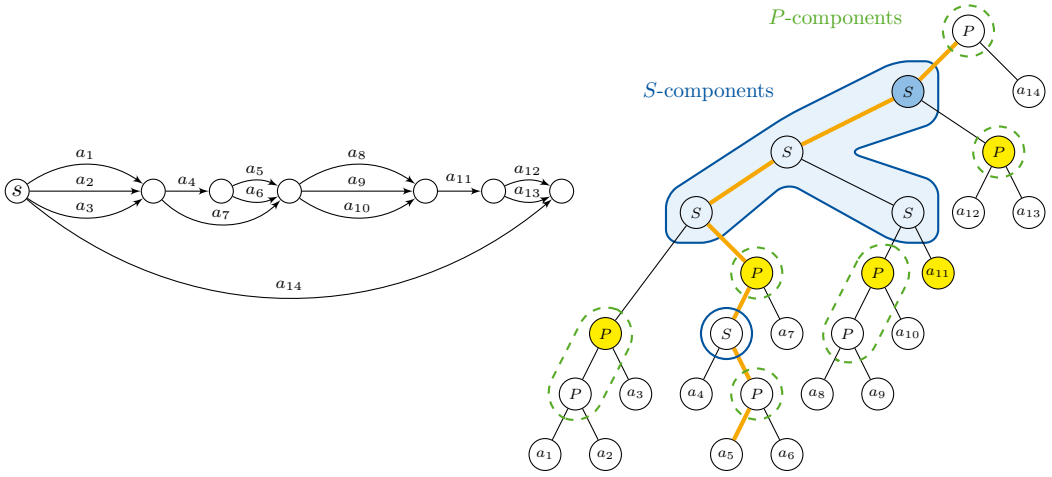


Figure 3 An example of a SePa-graph on the left. A corresponding series-parallel decomposition tree is shown on the right. Consider the large S-component (shaded in light blue), its root is highlighted blue and its children are highlighted yellow. Along the orange (bold) path (from the root to a leaf), we traverse two S-components. Also every other path traverses at most two, so $\varphi(D) = 2$.

Main result and overview. We can now formally state our main result for this section. Note that H_k denotes the k-th harmonic number, i.e., $H_k = \sum_{j=1}^{k} \frac{1}{j} \leq \log_2(k) + 1$.

▶ **Theorem 8.** *For every $\varepsilon > 0$ there is an algorithm that computes a $(\min\{H_k, \varphi(D) + 1\}) \cdot (1+\varepsilon)$-approximate path profile for MinMaxDP and runs in time polynomial in m, k, and $\frac{1}{\varepsilon}$.*

A central component of our algorithm is a *greedy composition* procedure that inductively combines path profiles in subgraphs based on the binary decomposition tree. In Section 5.1, we present this greedy composition procedure and several invariants maintained by the procedure that are crucial to our analysis. In particular, we show that if the invariants are fulfilled by the final solution, they imply the desired approximation ratio.

A remaining challenge is that it is not obvious how to initialise the procedure with solutions that satisfy the invariants at the beginning. In particular, some of the invariants are defined with respect to certain properties of the optimal solution and are thus not easy to verify. In Section 5.2, we show how to solve this issue by devising two dynamic programs, each optimising for one of the two terms whose minimum defines the approximation guarantee. Running these two DPs concurrently then yields the guarantee of the theorem (where ε is an error incurred by rounding so that the DPs run in polynomial time).

5.1 Greedy composition procedure

We use the following notation. We write $D' \sqsubseteq D$ for a series-parallel subgraph D' which corresponds to one of the vertices of the decomposition tree of D, i.e., the decomposition tree of D' is the subtree rooted at that vertex. Now consider any such subgraph $D' \sqsubseteq D$ with arc set A', source s', and sink t'. If an s-t-path P in a path profile $\boldsymbol{P}$ of D uses an arc in A', the series-parallel structure of D implies that $P \cap A'$ is an (s', t')-path in D'. We denote the length of a path P in D' by $\tau(P, D') \coloneqq \sum_{a \in P \cap A'} \tau_a$, and the length of the longest subpath of $\boldsymbol{P}$ in D' by $C_{\max}(\boldsymbol{P}, D') \coloneqq \max_{P \in \boldsymbol{P}} \tau(P, D')$. Moreover, we call $\theta(\boldsymbol{P}, D')$ the total length of arcs of path profile $\boldsymbol{P}$ in component D', that is $\theta(\boldsymbol{P}, D') \coloneqq \sum_{P \in \boldsymbol{P}} \tau(P, D')$.

Greedy composition. Let D' and D'' be two SePa-graphs, and let $\boldsymbol{P'}$ and $\boldsymbol{P''}$ be path profiles of D' and D'' consisting of k' and k'' arc-disjoint paths, respectively. For a series composition $D = D' \odot_s D''$, if $k' = k''$, the *greedy composition* of $\boldsymbol{P}$ and $\boldsymbol{P'}$ is the path profile $\boldsymbol{P} = \boldsymbol{P'} \odot_s \boldsymbol{P''}$ of D that is derived by combining the longest path in $\boldsymbol{P'}$ with the shortest path in $\boldsymbol{P''}$, the second-longest path in $\boldsymbol{P'}$ with the second-shortest path in $\boldsymbol{P''}$, and so on. Note that this construction is only defined for $k' = k''$. For a parallel composition $D = D' \odot_p D''$, the greedy composition is given by $\boldsymbol{P} = \boldsymbol{P'} \odot_p \boldsymbol{P''}$ of D, where $\boldsymbol{P}$ simply consists of the union of the $k' + k''$ paths in $\boldsymbol{P'}$ and $\boldsymbol{P''}$.

We now discuss three properties that are maintained by the greedy composition procedure described above if they are fulfilled by the path profiles that are being combined. To define these properties, we fix an arbitrary optimal path profile $\boldsymbol{P}^*$ in the graph D that defines our MinMaxDP instance. The first property we consider is consistency.

Consistency. We call a path-profile $\boldsymbol{P}$ *consistent* with $\boldsymbol{P}^*$ in $D' \sqsubseteq D$ if $\boldsymbol{P}$ and $\boldsymbol{P}^*$ use the same number of paths in D' and if

$$\theta(\boldsymbol{P}, D') \leq \theta(\boldsymbol{P}^*, D'). \tag{1}$$

Note that consistency is maintained by greedy composition, as the arcs used by the disjoint paths in the composed profile are exactly the union of the paths in the two profiles that are being combined. We now define the second property, balancedness, and show that together with consistency it implies the first of our two approximation bounds.

Balancedness. Let $\boldsymbol{P}$ be a path profile, and let $D' \sqsubseteq D$ be traversed by $k' \leq k$ paths from $\boldsymbol{P}$. Let $p_1 \geq \ldots \geq p_{k'}$ be the lengths of those paths in D'. Then $\boldsymbol{P}$ is *balanced* in D' if

$$\frac{1}{i}\left(\sum_{j=1}^{i} p_j\right) - p_{i+1} \leq C_{\max}(\boldsymbol{P}^*, D') \qquad \text{for all } i \in [k'-1]. \tag{2}$$

We first establish that balancedness – when combined with consistency – is indeed maintained by greedy composition. While this is straightforward for parallel composition, the proof for series composition requires a more involved analysis that carefully combines (1) and (2). The details of this analysis are given in the full version of this article [1, Appendix F.1].

▶ **Lemma 9.** *Let $\odot \in \{\odot_s, \odot_p\}$ and let $D^{(1)} \odot D^{(2)} = D'$. If $\boldsymbol{Q}$ and $\boldsymbol{R}$ are consistent and balanced path profiles in $D^{(1)}$ and $D^{(2)}$, respectively, then the greedy profile $\boldsymbol{P} = \boldsymbol{Q} \odot \boldsymbol{R}$ is a consistent and balanced path profile in D'.*

Crucially, the following lemma reveals that if the final solution is balanced and consistent with $\boldsymbol{P}^*$, it is a logarithmic approximation for the latter. The lemma follows from an averaging argument over (2) for $i \in [k-1]$, which can be found in the full version of this article [1, Appendix F.2].

▶ **Lemma 10.** *Let $\boldsymbol{P}$ be a consistent and balanced path profile in D. Then $C_{\max}(\boldsymbol{P}, D) \leq H_k \cdot C_{\max}(\boldsymbol{P}^*, D)$.*

We now introduce the third property, φ-boundedness, which is likewise maintained by greedy composition and implies the second bound on the approximation guarantee when fulfilled by the final solution.

φ-boundedness. We call a path profile $\boldsymbol{P}$ *φ-bounded* in $D' \sqsubseteq D$ if

$$C_{\max}(\boldsymbol{P}, D') \leq (\varphi(D') + 1) \cdot C_{\max}(\boldsymbol{P}^*, D). \tag{3}$$

Note that in this definition, the left-hand side refers to the length of the longest path of $\boldsymbol{P}$ in the subgraph $D' \sqsubseteq D$, whereas the right-hand side refers to the optimal objective value for MinMaxDP on the entire graph D. The following lemma establishes that consistency and φ-boundedness are maintained by the greedy composition when executing all compositions that correspond to one component of the decomposition tree.

▶ **Lemma 11.** *Let $D' \sqsubseteq D$ be a subgraph corresponding to a root vertex of an S-component or P-component, respectively. Let $D^{(1)}, \ldots, D^{(i)}$ be the graphs corresponding to the children of the component. Further, let $\boldsymbol{P}^{(j)}$ be a consistent and φ-bounded path profile in $D^{(j)}$ for $j \in [i]$. Then, the greedy series, respectively parallel, composition $\boldsymbol{P}$ of $\boldsymbol{P}^{(1)}, \ldots, \boldsymbol{P}^{(i)}$ is a consistent and φ-bounded path profile in D'.*

The proof of this lemma involves an induction over the components of the decomposition tree of D and can be found in the full version of this article [1, Appendix F.3].

5.2 Dynamic programs

The results in the preceding section imply that if we start from path profiles for the leaves of the decomposition tree of D that are consistent with an optimal solution and balanced or φ-bounded, respectively, then iteratively applying greedy composition will yield a path profile for D that is consistent and balanced or φ-bounded, respectively. In particular, such a profile then yields an H_k-approximation or $(\varphi(D)+1)$-approximation.

However, since we do not know the optimal solution, we do not know which path profiles fulfil these requirements. A natural approach seems to be to restrict ourselves to a solution with minimum total length (i.e., an integral minimum-cost s-t-flow of value k). However, there is a difficult trade-off between minimising the length of a longest path or minimising the total length of the solution in a subgraph, as the following example shows.

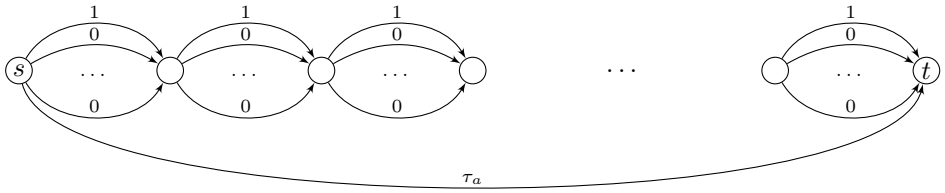


Figure 4 The depicted graph D is built as follows. Let D' be a graph with k parallel arcs (one arc with length 1 and $k-1$ arcs with length 0). We obtain D by connecting k copies of D' in series plus adding an additional arc from s to t with length τ_a.

▶ **Example 12.** Consider the graph D' depicted in Figure 4. If we set $\tau_a := k-1$, the path profile with minimal total length consists of $k-1$ paths of length 0 and one path with length $k-1$, leading to the objective value $k-1$. However, the optimal path profile consists of k paths all having length 1. On the other hand, assume that $\tau_a := 1+\varepsilon$ and create a graph D that consists of k copies of D' composed in series. In this case, solving the MinMaxDP in each subgraph D' results in all paths having length 1 inside each copy of the example, yielding to the objective value k when combining them in series. However, the optimal path profile uses exactly one arc of length $1+\varepsilon$ in each path and otherwise only length-0 arcs, yielding an objective value of $1+\varepsilon$. Thus, an algorithm which locally only considers the length of the longest path in the profile, would be a factor of k apart form the optimal value.

DP framework. In the following, we show how to resolve this issue by means of two dynamic programs. In both cases, our dynamic programming table will have the following attributes per cell:

- a subgraph $D' \sqsubseteq D$,
- the number of paths $k' \in [k]$ used in D',
- the total length $\theta' \in \Theta$ of the path profile in D'.

Recall that the total length of a path profile $\boldsymbol{P}$ in subgraph D' was defined as the sum of the lengths of the paths in D', i.e., $\theta(\boldsymbol{P}, D') := \sum_{P \in \boldsymbol{P}} \sum_{a \in A[D'] \cap P} \tau_a$. We let Θ denote the set containing all possible values that might appear as total costs.

To fill the cells of the dynamic programming table, we start with the subgraphs consisting of single arcs, i.e., $D' = (\{v, w\}, \{a = (v, w)\})$. Here, the non-empty entries of the table corresponding to D' are $(D', 1, \tau_a) = \{(a)\}$ and $(D', 0, 0) = \{\emptyset\}$, while all remaining entries corresponding to D' are empty (which is different from containing an empty path profile).

Given a cell $(D' \odot D'', \tilde{k}, \tilde{\theta})$, we combine all path profiles $\boldsymbol{P'}$ and $\boldsymbol{P''}$ of D' and D'' consisting of k' and k'' paths of total length θ' and θ'', respectively, if and only if $\tilde{\theta} = \theta' + \theta''$ and $\tilde{k} = k' + k''$ (in case of a parallel composition), or $\tilde{k} = k' = k''$ (in case of a series composition). We build path profiles for $D' \odot D''$ according to the greedy composition of the path profiles for D' and D'' as described above.

With this procedure, we obtain multiple candidates for a path profile in table entry $(D' \odot D'', \tilde{k}, \tilde{\theta})$. Below, we analyse two different strategies to pick a *good* candidate that lead to different approximation guarantees. After filling the complete table of the dynamic

program, there is a path profile $\boldsymbol{P}_\theta$ with k paths in D for each possible total-cost value $\theta \in \Theta$. In the last step, the algorithm will choose the path profile for which the longest path is as short as possible, i.e. the algorithm chooses $\arg\min_{\theta\in\Theta} C_{\max}(\boldsymbol{P}_\theta, D)$.

DP strategy for balancedness. Our first strategy for selecting the candidates for the intermediate steps aims for balancedness of the solutions and is formalised in the following lemma, that reveals that the strategy yields an H_k-approximation.

▶ **Lemma 13.** *The dynamic program which always selects the candidate $\boldsymbol{P}$ with k' paths of lengths $p_1 \geq \cdots \geq p_{k'}$ that minimises*

$$\max_{i\in\{1,\dots,k'-1\}} \left\{ \frac{1}{i}\sum_{j=1}^{i} p_j - p_{i+1} \right\}, \tag{4}$$

computes a solution with $DP \leq H_k \cdot \mathrm{OPT}$, where DP is the objective value of the computed solution and OPT *is the objective value of an optimal solution.*

The proof of this lemma is by induction on the table entries of the optimal solution and by using Lemma 9 to show that we always have a balanced candidate. Together with the choice (4), the lemma follows. A formal proof can be found in the full version of this article [1, Appendix F.4].

DP strategy for boundedness. Our second candidate selection strategy aims at φ-boundedness of the intermediate solutions. This is formalised in the following lemma, which shows that this strategy yields a $(\varphi(D)+1)$-approximation.

▶ **Lemma 14.** *The dynamic program that always selects the candidate $\boldsymbol{P}$ with k' paths of lengths $p_1 \geq \cdots \geq p_{k'}$ that minimises*

$$p_1 - p_{k'} \quad \text{in case of a series combination, and} \tag{5}$$
$$p_1 \quad \text{in case of a parallel composition} \tag{6}$$

computes a solution with $DP \leq (\varphi(D)+1)\cdot \mathrm{OPT}$, where DP is the objective value of the computed solution and OPT *is the objective value of an optimal solution.*

Similarly to the proof of Lemma 13 we follow the table entries of an optimal solution. However, here we need to consider a component of the decomposition tree of D at once and show that the entries for the subgraphs corresponding to segment roots are φ-bounded. The formal proof of this lemma can be found in the full version of this article [1, Appendix F.5].

Running time. From the size of the dynamic programming table, we can obtain the bound on its running time formalised in the lemma below. By a standard rounding approach, we can further ensure that the number of possible total-cost values $|\Theta|$ is polynomial in the size of the graph at a loss of a $(1+\varepsilon)$-factor in the approximation ratio, obtaining the desired polynomial running time; see the full version of this article [1, Appendix F.6] for details.

▶ **Lemma 15.** *The dynamic program runs in time $\mathcal{O}(m \cdot k^4 \cdot |\Theta|^2)$.*

6 Conclusion

In this paper, we introduced TMO as a natural optimisation problem that captures some of the fundamental challenges of routing trains with a headway constraint. By showing the existence of optimal train routings with a convoy structure, we linked the approximability of the problem to that of MinMaxDP, for which we then provided new approximation results in SePa-graphs. Several interesting questions arise from our work:

- Is it possible to extend the logarithmic approximation guarantee for MinMaxDP in SePa-graphs to arbitrary DAGs (where it would be best possible due to Theorem 1)?
- Conversely, can the inapproximability result from [10] for DAGs be extended to SePa-graphs (again implying that the greedy approach is best possible in those graphs)?
- Which of the results presented in this paper can be extended to the setting in which trains might have distinct sources and sinks? Note that for general DAGs even finding disjoint paths for multiple source-sink pairs is NP-hard [12] and that this still holds true in undirected SePa-graphs [23]; however, this hardness result does not carry over to directed SePa-graphs, so there is hope that our approximation results for those graphs extend to the multi commodity case.

References

1 Umang Bhaskar, Katharina Eickhoff, Lennart Kauther, Jannik Matuschke, Britta Peis, and Laura Vargas Koch. On the Approximability of Train Routing and the Min-Max Disjoint Paths Problem, 2025. doi:10.48550/arXiv.2507.03687.

2 Hans L. Bodlaender and Babette de Fluiter. Parallel algorithms for series parallel graphs. In Josep Díaz and Maria J. Serna, editors, *Algorithms — ESA '96*, Lecture Notes in Computer Science, pages 277–289, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. doi:10.1007/3-540-61680-2_62.

3 Ralf Borndörfer, Torsten Klug, Thomas Schlechte, Armin Fügenschuh, Thilo Schang, and Hanno Schülldorf. The freight train routing problem for congested railway networks with mixed traffic. *Transportation Science*, 50(2):408–423, 2016. doi:10.1287/trsc.2015.0656.

4 Valentina Cacchiani, Alberto Caprara, and Paolo Toth. Scheduling extra freight trains on railway networks. *Transportation Research Part B: Methodological*, 44(2):215–231, 2010. doi:10.1016/j.trb.2009.07.007.

5 Gabrio Caimi, Leo Kroon, and Christian Liebchen. Models for railway timetable optimization: Applicability and applications in practice. *Journal of Rail Transport Planning & Management*, 6(4):285–312, 2017. doi:10.1016/j.jrtpm.2016.11.002.

6 Malachy Carey. Ex ante heuristic measures of schedule reliability. *Transportation Research Part B: Methodological*, 33(7):473–494, 1999. doi:10.1016/S0191-2615(99)00002-8.

7 Edward G. Coffman, Jr. and Mihalis Yannakakis. Permuting elements within columns of a matrix in order to minimize maximum row sum. *Mathematics of Operations Research*, 9(3):384–390, 1984. doi:10.1287/moor.9.3.384.

8 José Correa, Carolina Osorio, Laura Vargas Koch, David Watling, and Svenja Griesbach. Dynamic traffic models in transportation science (dagstuhl seminar 24281). *Dagstuhl Reports*, 14(7):1–16, 2024. doi:10.4230/DagRep.14.7.1.

9 Syamantak Das and Andreas Wiese. On minimizing the makespan when some jobs cannot be assigned on the same machine. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.31.

10 Trivikram Dokka and Marc Goerigk. Multi-level bottleneck assignment problems: Complexity and sparsity-exploiting formulations. *Computers & Operations Research*, 154:106213, 2023. doi:10.1016/j.cor.2023.106213.

11 Daniel Dressler and Martin Skutella. An FPTAS for flows over time with aggregate arc capacities. In Klaus Jansen and Roberto Solis-Oba, editors, *International Workshop on Approximation and Online Algorithms*, volume 6534 of *Lecture Notes in Computer Science*, pages 106–117. Springer, 2010. doi:10.1007/978-3-642-18318-8_10.

12 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976. doi:10.1137/0205048.

13 Rudolf Fleischer, Qi Ge, Jian Li, and Hong Zhu. Efficient algorithms for k -disjoint paths problems on dags. In Ming-Yang Kao and Xiang-Yang Li, editors, *Algorithmic Aspects in Information and Management*, volume 4508 of *Lecture Notes in Computer Science*, pages 134–143. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-72870-2_13.

14 Lester Randolph Ford, Jr. and Delbert Ray Fulkerson. Constructing maximal dynamic flows from static flows. *Operations research*, 6(3):419–433, 1958. doi:10.1287/opre.6.3.419.

15 T. Godwin, Ram Gopalan, and T. T. Narendran. Freight train routing and scheduling in a passenger rail network: Computational complexity and the stepwise dispatching heuristic. *Asia-Pacific Journal of Operational Research*, 24(4):499–533, 2007. doi:10.1142/S0217595907001358.

16 Rebecca Haehn, Erika Ábrahám, and Nils Nießen. Freight train scheduling in railway systems. In Holger Hermanns, editor, *Measurement, Modelling and Evaluation of Computing Systems*, volume 12040, pages 225–241. Springer, 2020. doi:10.1007/978-3-030-43024-5_14.

17 Wen-Lian Hsu. Approximation algorithms for the assembly line crew scheduling problem. *Mathematics of Operations Research*, 9(3):376–383, 1984. doi:10.1287/moor.9.3.376.

18 Fahimeh Khoshniyat. *Optimization-Based Methods for Revising Train Timetables with Focus on Robustness*. Linköping University Electronic Press, 2016. doi:10.3384/lic.diva-132920.

19 Leo G. Kroon, H. Edwin Romeijn, and Peter J. Zwaneveld. Routing trains through railway stations: complexity issues. *European Journal of Operational Research*, 98(3):485–498, 1997. doi:10.1016/S0377-2217(95)00342-8.

20 Chung-Lun Li, S. Thomas McCormick, and David Simchi-Levi. The complexity of finding two disjoint paths with min-max objective function. *Discrete Applied Mathematics*, 26(1):105–115, 1990. doi:10.1016/0166-218X(90)90024-7.

21 Richard M. Lusby, Jesper Larsen, Matthias Ehrgott, and David Ryan. Railway track allocation: models and methods. *OR Spectrum*, 33(4):843–883, 2011. doi:10.1007/s00291-009-0189-0.

22 Shi Mu and Maged Dessouky. Scheduling freight trains traveling on complex networks. *Transportation Research Part B: Methodological*, 45(7):1103–1123, 2011. doi:10.1016/j.trb.2011.05.021.

23 Takao Nishizeki, Jens Vygen, and Xiao Zhou. The edge-disjoint paths problem is NP-complete for series–parallel graphs. *Discrete Applied Mathematics*, 115(1):177–186, 2001. doi:10.1016/S0166-218X(01)00223-2.

24 Bianca Pascariu, Marcella Samà, Paola Pellegrini, Andrea d'Ariano, Joaquin Rodriguez, and Dario Pacciarelli. Formulation of train routing selection problem for different real-time traffic management objectives. *Journal of Rail Transport Planning & Management*, 31:100460, 2024. doi:10.1016/j.jrtpm.2024.100460.

25 Thomas Rothvoss, Laura Sanità, and Robert Weismantel. Combinatorial optimization. Technical Report 50/2024, Mathematisches Forschungsinstitut Oberwolfach, 2024. doi:10.14760/OWR-2024-50.

26 Sedigheh Sadeghhosseini. *Time headway and platooning characteristics of vehicles on interstate highways*. University of Illinois at Urbana-Champaign, 2002. URL: https://hdl.handle.net/2142/83185.

27 Masahide Saho and Maiko Shigeno. Cancel-and-tighten algorithm for quickest flow problems. *Networks*, 69(2):179–188, 2017. doi:10.1002/net.21726.

28 Martin Skutella. An introduction to network flows over time. In William Cook, László Lovász, and Jens Vygen, editors, *Research Trends in Combinatorial Optimization: Bonn 2008*, pages 451–482. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-540-76796-1_21.

The Geodesic Fréchet Distance Between Two Curves Bounding a Simple Polygon

Thijs van der Horst
Department of Information and Computing Sciences, Utrecht University, The Netherlands
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Marc van Kreveld
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Tim Ophelders
Department of Information and Computing Sciences, Utrecht University, The Netherlands
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

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

Abstract

The Fréchet distance is a popular similarity measure that is well-understood for polygonal curves in $\mathbb{R}^d$: near-quadratic time algorithms exist, and conditional lower bounds suggest that these results cannot be improved significantly, even in one dimension and when approximating with a factor less than three. We consider the special case where the curves bound a simple polygon and distances are measured via geodesics inside this simple polygon. Here the conditional lower bounds do not apply; Efrat et al. (2002) were able to give a near-linear time 2-approximation algorithm.

In this paper, we significantly improve upon their result: we present a $(1+\varepsilon)$-approximation algorithm, for any $\varepsilon > 0$, that runs in $\mathcal{O}(\frac{1}{\varepsilon}(n + m \log n) \log nm \log \frac{1}{\varepsilon})$ time for a simple polygon bounded by two curves with n and m vertices, respectively. To do so, we show how to compute the reachability of specific groups of points in the free space at once, by interpreting the free space as one between separated one-dimensional curves. We solve this one-dimensional problem in near-linear time, generalizing a result by Bringmann and Künnemann (2015). Finally, we give a linear time exact algorithm if the two curves bound a convex polygon.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fréchet distance, approximation, geodesic, simple polygon

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.35

Related Version *Full Version*: `https://arxiv.org/abs/2501.03834`

Funding *Tim Ophelders*: partially supported by the Dutch Research Council (NWO) under project no. VI.Veni.212.260.

1 Introduction

The Fréchet distance is a well-studied similarity measure for curves in a metric space. Most results so far concern the Fréchet distance between two polygonal curves R and B in $\mathbb{R}^d$ with n and m vertices, respectively. Then the Fréchet distance between two such curves can be computed in $\tilde{\mathcal{O}}(nm)$ time (see e.g. [1, 5]). There is a closely matching conditional lower bound: If the Fréchet distance between polygonal curves can be computed in $\mathcal{O}((nm)^{1-\varepsilon})$ time (for any constant $\varepsilon > 0$), then the Strong Exponential Time Hypothesis fails [3]. This lower bound extends to curves in one dimension, and holds even when approximating to a factor less than three [6].

Because it is unlikely that exact strongly subquadratic algorithms exist, approximation algorithms have been developed [9, 15, 8]. The authors [15] were the first to present an algorithms which results in an arbitrarily small polynomial approximation factor (n^{ε} for any


© Thijs van der Horst, Marc van Kreveld, Tim Ophelders, and Bettina Speckmann;
licensed under Creative Commons License CC-BY 4.0



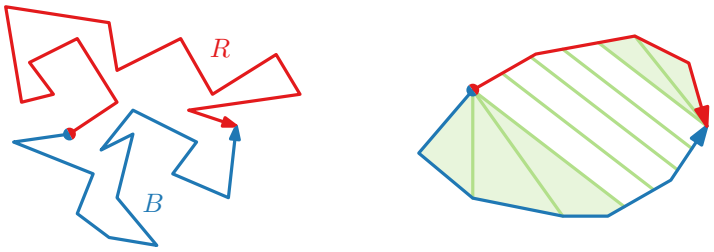


Figure 1 (left) Curves R and B bound a simple polygon. (right) A maximally-parallel matching.

$\varepsilon \in (0, 1]$) in strongly subquadratic time ($\tilde{\mathcal{O}}(n^{2-\varepsilon})$). Very recently, Cheng et al. [8] gave the first (randomized) constant-factor approximation algorithm with a strongly subquadratic running time. Specifically, it computes a $(7+\varepsilon)$-approximation in $\mathcal{O}(nm^{0.99}\log(n/\varepsilon))$ time.

For certain families of "realistic" curves, the SETH lower bound does not apply. For example, when the curves are *c-packed*, Driemel et al. [11] gave an $(1+\varepsilon)$-approximation algorithm, for any $\varepsilon \in (0, 1)$, that runs in $\tilde{\mathcal{O}}(cn/\varepsilon)$ time. Bringmann and Künnemann [4] improved the running time to $\tilde{\mathcal{O}}(cn/\sqrt{\varepsilon})$ time.

For curves in one dimension with an imbalanced number of vertices, the Fréchet distance can be computed in strongly subquadratic time without making extra assumptions about the shape of the curves. This was recently established by Blank and Driemel [2], who give an $\tilde{\mathcal{O}}(n^{2\alpha}+n)$-time algorithm when $m = n^\alpha$ for some $\alpha \in (0, 1)$.

If the two polygonal curves R and B lie inside a simple polygon P with k vertices and we measure distances by the geodesic distance inside P, then neither the upper nor the conditional lower bound change in a fundamental way. Specifically, Cook and Wenk [10] show how to compute the Fréchet distance in this setting in $\mathcal{O}(k + N^2 \log kN \log N)$ time, with $N = \max\{n, m\}$. For more general polygonal obstacles, Chambers et al. [7] give an algorithm that computes the *homotopic* Fréchet distance in $\mathcal{O}(N^9 \log N)$ time, where $N = m + n + k$ is the total number of vertices on the curves and obstacles.

Har-Peled et al. [13] investigate the setting where R and B are simple, interior-disjoint curves on the boundary of a triangulated topological disk. If the disk has k faces, their algorithm computes a $\mathcal{O}(\log k)$-approximation to the homotopic Fréchet distance in $\mathcal{O}(k^6 \log k)$ time. Efrat et al. [12] consider a more geometric setting, where R and B bound a simple polygon (see Figure 1 (left)). Here, the SETH lower bound does not apply; a 2-approximation to the geodesic Fréchet distance can be computed in $\mathcal{O}((n+m)\log nm)$ time [12]. Moreover, the authors [14] recently gave an $\mathcal{O}((n+m)\log^4 nm)$-time exact algorithm for a similar setting, where distances are measured under the L_1-geodesic distance. Their result implies a $\sqrt{2}$-approximation algorithm for the geodesic Fréchet distance.

Organization and results. In this paper, we significantly improve upon the results of Efrat et al. [12] and the authors [14]: we present a $(1+\varepsilon)$-approximation algorithm for the geodesic Fréchet distance, for any $\varepsilon \in (0, 1]$, that runs in $\mathcal{O}(\frac{1}{\varepsilon}(n + m\log n)\log nm \log \frac{1}{\varepsilon})$ time when R and B together bound a simple polygon. We give an overview of our algorithm in Section 2. Our algorithm relies on an interesting connection between matchings and nearest neighbors and is described in Section 3. There we also explain how to transform the decision problem for *far* points on B (those who are not a nearest neighbor of any point on R) into a problem between separated one-dimensional curves. This problem is about computing, given a set of "entrances" and "potential exits" (points anywhere in the parameter space), which potential exits are δ-reachable from entrances. Bringmann and Künnemann [4] study a

restricted variant of this problem in the context of Fréchet distance between c-packed curves. In addition to giving a near-linear time algorithm for the general problem, we also compute the Fréchet distance between two separated one-dimensional curves in linear time.

In the full version of this paper, we additionally describe a simple linear-time algorithm for when P is a convex polygon. We show that in this setting there is a Fréchet matching with a specific structure: a *maximally-parallel matching* (see Figure 1 (right)). We compute the orientation of the parallel part from up to $O(n+m)$ different tangents, which we find using "rotating calipers."

Preliminaries. A (polygonal) *curve* R with n *vertices* is a continuous map $R\colon [1,n] \to \mathbb{R}^d$ that is linear on each interval $[i, i+1]$ where i is an integer. Its vertices are the points $R(i)$. If the vertices lie in the plane, then we say R is two-dimensional.[1] A one-dimensional curve is defined analogously.

We denote by $R[x,x']$ the subcurve of R over the domain $[x,x']$, and abuse notation slightly to let $R[r,r']$ to also denote this subcurve when $r = R(x)$ and $r' = R(x')$. The *edges* of R are the directed line segments $R[i,i+1]$ for integers $i \in [1,n-1]$. We write $|R|$ to denote the number of vertices of R. In this work, we consider two simple, interior-disjoint curves $R\colon [1,n] \to \mathbb{R}^2$ and $B\colon [1,m] \to \mathbb{R}^2$ with $R(1) = B(1)$ and $R(n) = B(m)$, and say that they bound a simple polygon P.

A *reparameterization* of $[1,n]$ is a non-decreasing surjection $f\colon [0,1] \to [1,n]$. Two reparameterizations f and g of $[1,n]$ and $[1,m]$, describe a *matching* (f,g) between two curves R and B with n and m vertices, respectively, where any point $R(f(t))$ is matched to $B(g(t))$. The matching (f,g) is said to have *cost*

$$\max_t \; d(R(f(t)), B(g(t))),$$

where $d(\cdot,\cdot)$ is the geodesic distance between points in P. A matching with cost at most δ is called a *δ-matching*. The (continuous) *geodesic Fréchet distance* $d_{\mathrm{F}}(R,B)$ between R and B is the minimum cost over all matchings. The corresponding matching is a *Fréchet matching*.

The *parameter space* of R and B is the axis-aligned rectangle $[1,n] \times [1,m]$. Any point (x,y) in the parameter space corresponds to the pair of points $R(x)$ and $B(y)$ on the two curves. A point (x,y) in the parameter space is *δ-close* for some $\delta \geq 0$ if $d(R(x), B(y)) \leq \delta$. The *$\delta$-free space* $\mathcal{F}_\delta(R,B)$ of R and B is the set of δ-close points. Alt and Godau [1] observed that there is a one-to-one correspondence between δ-matchings between subcurves $R[x,x']$ and $B[y,y']$, and bimonotone paths from (x,y) to (x',y') through $\mathcal{F}_\delta(R,B)$. We abuse terminology slightly and refer to such paths as δ-matchings as well.

A point $q = (x',y') \in \mathcal{F}_\delta(R,B)$ is *δ-reachable* from a point $p = (x,y)$ if $x \leq x'$ and $y \leq y'$, and there exists a bimonotone (i.e., monotone in both coordinates) path in $\mathcal{F}_\delta(R,B)$ from p to q. That is, if the subcurves $R[x,x']$ and $B[y,y']$ have Fréchet distance at most δ between them. Points that are δ-reachable from $(1,1)$ are simply called δ-reachable points.

Let $\varepsilon \in (0,1]$ be a parameter. A $(1+\varepsilon)$-*approximate decision algorithm* for our problem takes a decision parameter $\delta \geq 0$, and outputs either that $d_{\mathrm{F}}(R,B) \leq (1+\varepsilon)\delta$ or that $d_{\mathrm{F}}(R,B) > \delta$. It may report either answer if $\delta < d_{\mathrm{F}}(R,B) \leq (1+\varepsilon)\delta$.

[1] Curves are inherently one-dimensional objects. We abuse terminology slightly to refer to the ambient dimension as the dimension of a curve.

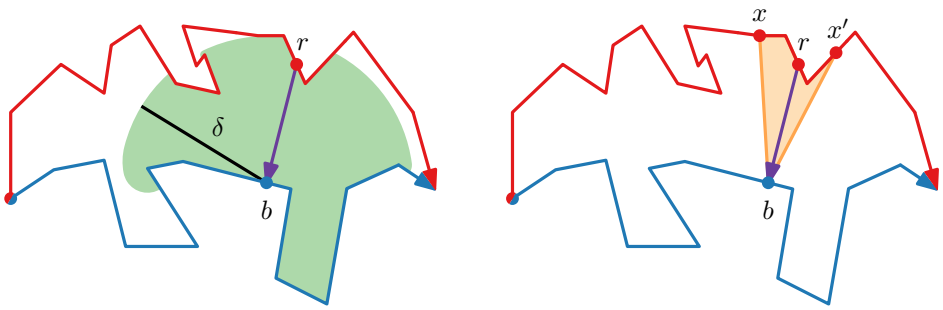


Figure 2 (left) Points r and b with $b \in NN(r)$. The region shaded in green consists of all points within geodesic distance δ of b. (right) The (r, b, δ)-nearest neighbor fan (orange).

2 Algorithmic outline

In this section we sketch the major parts of our algorithm that approximates the geodesic Fréchet distance $\delta_F := d_F(R, B)$ between curves R and B. We approximate δ_F using a $(1+\varepsilon)$-approximate decision algorithm, and use binary search to find the correct decision parameter. In Section 3.4 we relate the Fréchet distance to the geodesic Hausdorff distance δ_H, which allows us to bound the number of iterations of the binary search. In particular, we show that δ_F lies in the range $[\delta_H, 3\delta_H]$. The Hausdorff distance δ_H can be computed in $\mathcal{O}((n+m)\log nm)$ time [10, Theorem 7.1]. A binary search over $[\delta_H, 3\delta_H]$ results in a $(1+\varepsilon)$-approximation of the Fréchet distance after $\mathcal{O}(\log \frac{1}{\varepsilon})$ calls to the decision algorithm. Theorem 1 follows.

▶ **Theorem 1.** *For any $\varepsilon \in (0, 1]$, we can compute a $(1+\varepsilon)$-approximation to $d_F(R, B)$ in $\mathcal{O}(\frac{1}{\varepsilon}(n + m\log n)\log nm \log \frac{1}{\varepsilon})$ time.*

The approximate decision algorithm. In the remainder of this section we outline the approximate decision algorithm, which is presented in detail in Section 3. At its heart lies a useful connection between matchings and nearest neighbors. A *nearest neighbor* of a point r on R is any point b on B that among all points on B is closest to r. We denote the nearest neighbor(s) of r by $NN(r)$. We prove in Section 3.1 that any δ-matching matches each nearest neighbor b of r relatively close to r. Specifically, b must be matched to only points r' for which the entire subcurve of R between r and r' is within distance δ of b. We introduce the concept of a (r, b, δ)-nearest neighbor fan to capture the candidate locations for r'.

The *(r, b, δ)-nearest neighbor fan* $F_{r,b}(\delta)$ contains the point b and the maximal subcurve $R[x, x']$ that contains r and is within geodesic distance δ of b; it is the union of geodesics between b and points on $R[x, x']$, see Figure 2. We call b the *apex* of $F_{r,b}(\delta)$ and $R[x, x']$ the *leaf* of the fan. We prove in Section 3.1 that any δ-matching must match the apex b to a point in the leaf $R[x, x']$.

As r moves forward along R, so do its nearest neighbors b along B. Their nearest neighbor fans $F_{r,b}(\delta)$ move monotonically through the polygon P bounded by R and B. However, while r moves continuously along R, the points b and their nearest neighbor fans might jump discontinuously. Such discontinuities occur due to points b that are not a nearest neighbor of any point on R, and thus at points that are not the apex of a nearest neighbor fan. We classify the points on B accordingly: we call a point b on B a *near* point if it is a nearest neighbor of at least one point on R, and call b a *far* point otherwise.

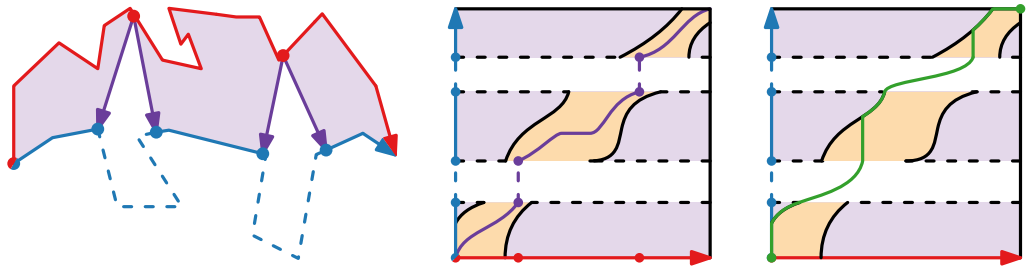


Figure 3 (left) The mappings (purple) from points on R to their nearest neighbor(s) on B. (middle) The partition of the parameter space based on near and far points on B. The partly-dashed purple curve indicates the nearest neighbor(s) on B of points on R. The beige regions correspond to the (r, b, δ)-nearest neighbor fans. (right) A δ-matching that is greedy on B inside the regions.

The distinction between near and far points induces a partition $\mathcal{H}$ of the parameter space into horizontal slabs. We consider these slabs from bottom to top, and iteratively construct a δ-matching (provided that one exists). Recall that a δ-matching is a bimonotone path in $\mathcal{F}_\delta(R, B)$ from the bottom-left corner of the parameter space to the top-right corner.

Inside a slab $H_{\text{near}} \in \mathcal{H}$ corresponding to a subcurve of B with only near points, the nearest neighbor fans correspond to a connected, x-monotone and y-monotone region $\mathcal{R}$ spanning the entire height of H_{near}. These regions are illustrated in Figure 3. The intersection between any δ-matching and H_{near} is contained in $\mathcal{R}$. The structure of $\mathcal{R}$ implies that if a δ-matching between R and B exists, there exists one which moves vertically up inside $\mathcal{R}$ whenever this is possible. Geometrically, this corresponds to greedily traversing the near points on B, and traversing parts of R only when necessary. We formalize this in Section 3.2.

Slabs whose corresponding subcurves of B have only far points pose the greatest technical challenge for our algorithm; we show how to match far points in an approximate manner in Section 3.3. Specifically, let $H_{\text{far}} \in \mathcal{H}$ correspond to a subcurve $B[b, b']$ with only far points on its interior. We compute a suitable subcurve of R that can be $(1+\varepsilon)\delta$-matched to $B[b, b']$ in the following manner. First we argue that $d(b, b') \leq 2\delta_F$. In other words, the geodesic from b to b' is short and separates R from the subcurve $B[b, b']$. We use this separating geodesic to transform the problem of creating a matching for far points into $K = \mathcal{O}(1/\varepsilon)$ one-dimensional problems.

Specifically, we discretize the separator with K points, which we call *anchors*, and ensure that consecutive anchors are at most $\varepsilon\delta$ apart. We snap our geodesics to these anchors (see Figure 4), which incurs a small approximation error. Based on which anchor a geodesic

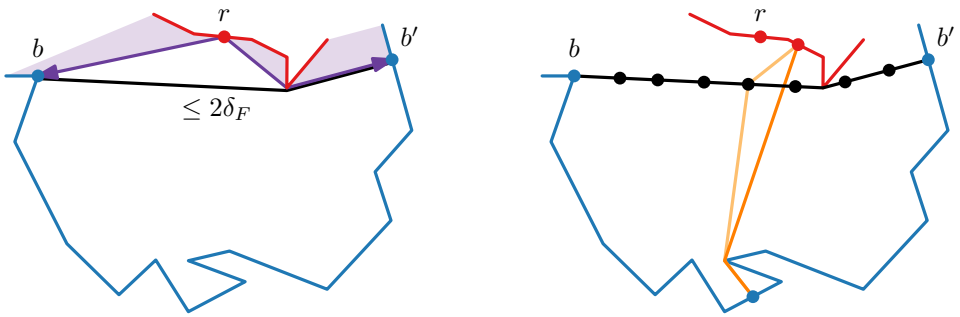


Figure 4 (left) The points b and b' are both nearest neighbor of some point r, implying a short separator. (right) Adding anchor points to the separator and snapping the orange geodesic (between arbitrary points on R and $B[b, b']$) to one.

snaps to, we partition the parameter space of R and $B[b, b']$ into regions, one for each anchor. For each anchor point, the lengths of the geodesics snapped to it can be expressed as distances between points on two separated one-dimensional curves; this results in a one-dimensional problem which we can solve exactly. In Section 3.3 we present the transformation into one-dimensional curves, and in Section 4 we present an efficient exact algorithm for the resulting one-dimensional problem.

3 Approximate geodesic Fréchet distance

3.1 Nearest neighbor fans and partitioning the parameter space

We first present useful properties of nearest neighbor fans and their relation to matchings. In Lemma 3 we give a crucial property of nearest neighbor fans, namely that *any* δ-matching between R and B matches b to only points in the leaf of the fan. For the proof, we make use of the following auxiliary lemma:

▶ **Lemma 2.** *Let $r \in R$ and $b \in NN(r)$. For any points $r' \in R$ and $b' \in B$ on opposite sides of the geodesic $\pi(r, b)$ between r and b, we have $d(r', b) \leq d(r', b')$.*

▶ **Lemma 3.** *Let $r \in R$ and $b \in NN(r)$. For any $\delta \geq 0$, every δ-matching between R and B matches b to only points in the leaf of $F_{r,b}(\delta)$.*

Proof. Fix a δ-matching (f, g) and let r' be a point matched to b by this matching. Assume without loss of generality that r' comes before r along R. Let $\hat{r}$ be a point between r' and r. The δ-matching (f, g) matches $\hat{r}$ to at least one point $\hat{b}$ after b. Thus we obtain from Lemma 2 that $d(\hat{r}, b) \leq d(\hat{r}, \hat{b}) \leq \delta$. This proves that all points between r' and r are included in the leaf of $F_{r,b}(\delta)$. ◀

We partition the parameter space into (closed) maximal horizontal slabs, such that for each slab $[1, n] \times [y, y']$, either the subcurve $B[y, y']$ contains only near points, or its interior contains only far points. Let $\mathcal{H}$ be the resulting partition. Each slab $H \in \mathcal{H}$ has two horizontal line segments, one on its bottom and one on its top side, that correspond to nearest neighbor fans. We refer to these line segments as the *entrance* and *exit intervals* of H, and refer to points on them as the *entrances* and *exits* of H. A consequence of Lemma 3 is that any δ-matching enters and leaves H through an entrance and an exit, respectively. We compute *δ-safe* entrances and exits: δ-reachable points from which (n, m) is δ-reachable. Each such point is used by a δ-matching, and is used to iteratively determine if such a matching exists.

It proves sufficient to consider only a discrete set of entrances and exits for each slab. For each slab, we identify (implicitly) a set of $\mathcal{O}(n)$ entrances and exits that contain δ-safe entrances and exits (if any exist at all). We define these entrances and exits using locally closest points and will call them transit points.

A point r on R is *locally closest* to a point b on B if perturbing r infinitesimally while staying on R increases its distance to b. The *transit* entrances and exits are those entrances and exits (x, y) where $R(x)$ is either a vertex or locally closest to $B(y)$. We show that it is sufficient to consider only transit entrances and exits:

▶ **Lemma 4.** *If there exists a δ-matching, then there exists one that enters and leaves each slab through a transit entrance and exit.*

Our algorithm computes a δ-safe transit exit for each slab. To do so, it requires the explicit entrance and exit intervals. We compute these using a geodesic Voronoi diagram.

▶ **Lemma 5.** *The partition $\mathcal{H}$ consists of $\mathcal{O}(m)$ slabs. We can compute $\mathcal{H}$, together with the entrance and exit interval of each slab, in $\mathcal{O}((n+m)\log nm)$ time in total.*

3.2 Slabs of near points

Let $H_{\text{near}} \in \mathcal{H}$ be a slab corresponding to a subcurve $\hat{B}$ of B with only near points. We use properties of nearest neighbor fans to determine a δ-safe transit exit of H_{near}. A crucial property is that the nearest neighbor fans behave monotonically with respect to their apexes, if δ is large enough. Specifically, this is the case if $\delta \geq \delta_H$, the geodesic Hausdorff distance between R and B. This is the maximum distance between a point on $R \cup B$ and its closest point on the other curve.

▶ **Lemma 6.** *Suppose $\delta \geq \delta_H$. Let b and b' be near points on B and let $R[x_1, x_1']$ and $R[x_2, x_2']$ be the leaves of their respective nearest neighbor fans. If b comes before b', then $x_1 \leq x_2$ and $x_1' \leq x_2'$.*

The monotonicity of the nearest neighbor fans, together with the fact that each point on $\hat{B}$ corresponds to such a fan, ensures that we can determine such an exit in logarithmic time (see Lemma 8). We make use of the following data structure that reports transit exits:

▶ **Lemma 7.** *Given the exit interval $[x, x'] \times \{y\}$ of H_{near}, we can report the at most three transit exits on a horizontal line segment $[i, i+1] \times \{y\}$, for any integer $1 \leq i < n$, in $\mathcal{O}(\log nm)$ time, after $\mathcal{O}(n+m)$ preprocessing time.*

▶ **Lemma 8.** *Suppose $\delta \geq \delta_H$. Let $H_{\text{near}} \in \mathcal{H}$ be a slab corresponding to a subcurve of B with only near points. Given the exit interval of H_{near}, together with a δ-safe transit entrance, we can compute a δ-safe transit exit in $\mathcal{O}(\log nm)$ time, after $\mathcal{O}(n+m)$ preprocessing time.*

Proof (sketch). Given a δ-safe transit entrance p_{enter}, the leftmost transit exit to the right of p_{enter} is δ-safe. We report it with the data structure of Lemma 7. ◀

3.3 Slabs of far points

Next we give an algorithm for computing a δ-safe transit exit of a given slab $H_{\text{far}} \in \mathcal{H}$ that corresponds to a subcurve of B with only far points on its interior. Our algorithm is approximate: given $\varepsilon \in (0, 1]$, it computes an (ε, δ)-*safe* transit exit (if one exists). This is a $(1+\varepsilon)\delta$-reachable point from which (n, m) is δ-reachable. Such a transit exit behaves like a δ-safe transit exit for the purpose of iteratively constructing a matching.

To compute an (ε, δ)-safe transit exit, we make use of an approximate decision algorithm that uses the fact that $\hat{B}$ has only far points on its interior. We present this algorithm in Section 3.3.1. In Section 3.3.2 we then apply this approximate decision algorithm in a search procedure, where we search over the $\mathcal{O}(n)$ transit exits to compute an (ε, δ)-safe one.

3.3.1 Approximate decision algorithm for far points

Let $\hat{B}$ be the subcurve of B that corresponds to H_{far}, so its interior has only far points. Let $\hat{R}$ be an arbitrary subcurve of R, for which we seek to approximately determine whether $d_{\text{F}}(\hat{R}, \hat{B}) \leq \delta$. We report either that $d_{\text{F}}(\hat{R}, \hat{B}) \leq (1+\varepsilon)\delta$, or that $d_{\text{F}}(\hat{R}, \hat{B}) > \delta$.

For our algorithm, we first discretize the space of geodesics between points on $\hat{R}$ and points on $\hat{B}$, by grouping the geodesics into few ($\mathcal{O}(1/\varepsilon)$) groups and rerouting the geodesics in a group through some representative point. Let $\hat{b}$ and $\hat{b}'$ be the first, respectively last,

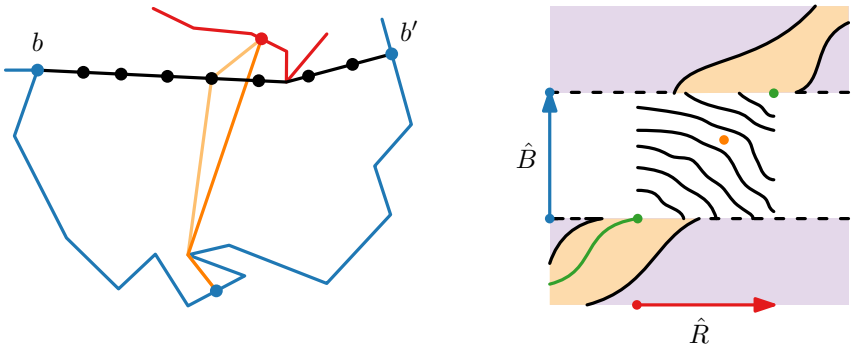


Figure 5 (left) Snapping a geodesic (orange) to an anchor. (right) The eight regions in the parameter space of $\hat{R}$ and $\hat{B}$ corresponding to the first eight (out of nine) anchors. The orange geodesic lies in region $\mathcal{R}_5$.

endpoints of $\hat{B}$. There is a point r on R with $NN(r) = \{\hat{b}, \hat{b}'\}$. We observe that this implies the geodesic $\pi(\hat{b}, \hat{b}')$ that connects $\hat{b}$ to $\hat{b}'$ is short with respect to the Fréchet distance, and thus with respect to any relevant value for δ:

▶ **Lemma 9.** $d(\hat{b}, \hat{b}') \leq 2d_{\mathrm{F}}(\hat{R}, \hat{B})$.

We assume for the remainder that $d(\hat{b}, \hat{b}') \leq 2\delta$; if this is not the case, we immediately report that $d_{\mathrm{F}}(\hat{R}, \hat{B}) > \delta$. This assumption means that the geodesic $\pi(\hat{b}, \hat{b}')$ is a short separator between $\hat{R}$ and $\hat{B}$. That is, any geodesic between a point on $\hat{R}$ and a point on $\hat{B}$ intersects $\pi(\hat{b}, \hat{b}')$. For clarity, we denote by π_{sep} the separator $\pi(\hat{b}, \hat{b}')$. We use the short separator to formulate the (exact) reachability problem as $\mathcal{O}(1/\varepsilon)$ subproblems involving one-dimensional curves. This is where we incur a small approximation error.

We discretize π_{sep} with $K + 1 = \mathcal{O}(1/\varepsilon)$ points $\hat{b} = a_1, \ldots, a_{K+1} = \hat{b}'$, which we call *anchors*, and ensure that consecutive anchors at most a distance $\varepsilon\delta$ apart, see Figure 5 (left). We assume that no anchor coincides with a vertex of $\hat{R} \cup \hat{B}$ (except for a_1 and a_{K+1}).

We route geodesics between points on $\hat{R}$ and points on $\hat{B}$ through these anchors. Specifically, for points $\hat{r} \in \hat{R}$ and $\hat{b} \in \hat{B}$, if $\pi(\hat{r}, \hat{b})$ intersects π_{sep} between consecutive anchors a_k and a_{k+1}, then we "snap" $\pi(\hat{r}, \hat{b})$ to a_k; that is, we replace it by the union of $\pi(\hat{r}, a_k)$ and $\pi(a_k, \hat{b})$ (see Figure 5 (right)). This creates a new path between $\hat{r}$ and $\hat{b}$ that goes through a_k and has length at most $d(\hat{r}, \hat{b}) + \varepsilon\delta$.

We associate an anchor a_k with the points (x, y) in the parameter space for which the geodesic $\pi(\hat{R}(x), \hat{B}(y))$ is snapped to a_k. These points form a region $\mathcal{R}_k$ that is connected and monotone in both coordinates (see Figure 5 (right)). We iteratively compute, for each region $\mathcal{R}_k$, a set of $(1 + \varepsilon)\delta$-reachable points on its boundary, such that the decision question can be answered by checking if the last set contains $(|\hat{R}|, |\hat{B}|)$. We later formulate these subproblems in terms of pairs of one-dimensional curves that are separated by a point.

We first discretize the problem. For this, we identify sets of points on the shared boundaries between the pairs of adjacent regions, such that there exists a δ-matching that enters and exits each region through such a set. Specifically, for $k = 2, \ldots, K$, we let the set S_k contain those points $(x, y) \in \mathcal{R}_{k-1} \cap \mathcal{R}_k$ for which one of $R(x)$ and $B(y)$ is a vertex or locally closest to a_k (so perturbing the point infinitesimally along its curve increases its distance to a_k). We set $S_1 = \{(1, 1)\}$. In the following lemmas, we show that these sets suit our needs and are efficiently constructable.

▶ **Lemma 10.** *If there exists a δ-matching between $\hat{R}$ and $\hat{B}$, then there exists one that goes through a point in S_k for every anchor a_k.*

▶ **Lemma 11.** *Each set S_k contains $\mathcal{O}(|\hat{R}| + |\hat{B}|)$ points and can be constructed in $\mathcal{O}((|\hat{R}| + |\hat{B}|) \log nm)$ time, after $\mathcal{O}(n + m)$ preprocessing time.*

Having constructed the sets S_k for all anchors in $\mathcal{O}(\frac{1}{\varepsilon}(|\hat{R}| + |\hat{B}|) \log nm)$ time altogether, we move to computing subsets $S_k^* \subseteq S_k$ containing all δ-reachable points, and only points that are $(1+\varepsilon)\delta$-reachable. We proceed iteratively, constructing S_{k+1}^* from S_k^*. For this, observe that for any point (x, y) in the interior of $\mathcal{R}_k$, the geodesic $\pi(\hat{R}(x), \hat{B}(y))$ was snapped to a_k. We use this fact to construct a pair of one-dimensional curves that approximately describe the lengths of these geodesics.

Let $\bar{R}_k \colon [1, |\hat{R}|] \to \mathbb{R}$ and $\bar{B}_k \colon [1, |\hat{B}|] \to \mathbb{R}$ be one-dimensional curves, where we set $\bar{R}_k(x) = -d(\hat{R}(x), a_k)$ and $\bar{B}_k(y) = d(\hat{B}(y), a_k)$ (note the difference in sign). These curves encode the distances between points on $\hat{R}$ and $\hat{B}$ when snapping geodesics to a_k. That is, $|\bar{R}_k(x) - \bar{B}_k(y)|$ is the length of $\pi(\hat{R}(x), \hat{B}(y))$ after snapping to a_k. Hence, for any pair of points $p \in S_k^*$ and $q \in S_{k+1}$, we have the following relations:

- If q is δ-reachable from p in the parameter space of $\hat{R}$ and $\hat{B}$, then it is $(1+\varepsilon)\delta$-reachable in the parameter space of $\bar{R}_k$ and $\bar{B}_k$.
- Conversely, if q is $(1+\varepsilon)\delta$-reachable from p in the parameter space of $\bar{R}_k$ and $\bar{B}_k$, then it is $(1+\varepsilon)\delta$-reachable in the parameter space of $\hat{R}$ and $\hat{B}$.

We define S_{k+1}^* as the points in S_{k+1} that are $(1+\varepsilon)\delta$-reachable from points in S_k^*, in the parameter space of $\bar{R}_k$ and $\bar{B}_k$. Computing these points is the problem involving one-dimensional curves that we alluded to earlier.

▶ **Lemma 12.** *Given S_k^*, we can compute S_{k+1}^* in $\mathcal{O}((|\hat{R}|+|\hat{B}|) \log nm)$ time, after $\mathcal{O}(n+m)$ preprocessing time.*

We apply the above procedure iteratively, computing S_k^* for each anchor a_k. These sets take a total of $\mathcal{O}(\frac{1}{\varepsilon}(|\hat{B}| + |\hat{R}|) \log nm)$ time to construct. Afterwards, if $(|\hat{R}|, |\hat{B}|) \in S_K^*$, we report that $d_{\mathrm{F}}(\hat{R}, \hat{B}) \leq (1+\varepsilon)\delta$. Otherwise, we report that $d_{\mathrm{F}}(\hat{R}, \hat{B}) > \delta$. We obtain:

▶ **Lemma 13.** *Let $\hat{R}$ be an arbitrary subcurve of R, and let $\hat{B}$ be a maximal subcurve of B with only far points on its interior. We can decide whether $d_{\mathrm{F}}(\hat{R}, \hat{B}) \leq (1+\varepsilon)\delta$ or $d_{\mathrm{F}}(\hat{R}, \hat{B}) > \delta$ in $\mathcal{O}(\frac{1}{\varepsilon}(|\hat{R}| + |\hat{B}|) \log nm)$ time, after $\mathcal{O}(n+m)$ preprocessing time.*

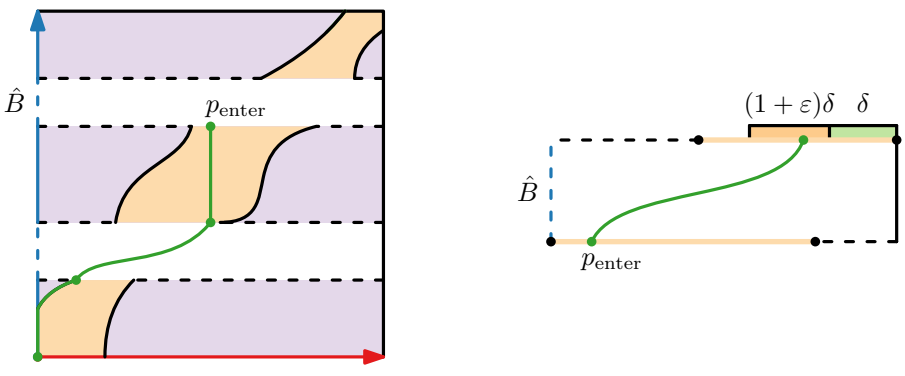


Figure 6 The subproblem of matching far points. The exit interval on the right is divided into three regions, based on reachability of points. The aim is to compute a $(1+\varepsilon)\delta$-reachable transit exit to the left of all δ-reachable transit exits.

3.3.2 Computing a good exit

Recall that we set out to compute an (ε, δ)-safe transit exit of H_{far}. We assume we are given an (ε, δ)-safe entrance $p_{\text{enter}} = (x, y)$. Since the entire exit interval of H_{far} lies in $\mathcal{F}_\delta(R, B)$, it suffices to compute a transit exit q_{exit} that is $(1+\varepsilon)\delta$-reachable from p_{enter} and that lies to the left of all transit exits that are δ-reachable from p_{enter}, see Figure 6. We compute such a transit exit q_{exit} through a search procedure, combined with the decision algorithm.

There are $\mathcal{O}(n)$ transit exits of H_{far}. To avoid running the decision algorithm for each of these, we use exponential search. The choice for exponential search over, e.g., binary search comes from the fact that the running time of the decision algorithm depends on the location of the transit exit, with transit exits lying further to the right in the exit interval of H_{far} needing more time for the decision algorithm. Exponential search ensures that we do not consider transit exits that are much more to the right than needed.

▶ **Lemma 14.** *Let $H_{\text{far}} \in \mathcal{H}$ be a slab corresponding to a subcurve $\hat{B}$ of B with only far points on its interior. Given an (ε, δ)-safe transit entrance $p = (x, y)$ of H_{far}, we can compute an (ε, δ)-safe transit exit $q = (x', y')$ in $\mathcal{O}(\frac{1}{\varepsilon}(|R[x, x']| + |\hat{B}|) \log n \log nm)$ time.*

3.4 The approximate optimization algorithm

We combine the algorithms of Sections 3.2 and 3.3 to obtain a $(1+\varepsilon)$-approximate decision algorithm, which we then turn into an algorithm that computes a $(1+\varepsilon)$-approximation of the geodesic Fréchet distance between R and B. Given $\delta \geq 0$ and $\varepsilon \in (0, 1]$, the approximate decision algorithm reports that $d_{\text{F}}(R, B) \leq (1+\varepsilon)\delta$ or $d_{\text{F}}(R, B) > \delta$.

Let δ_H be the geodesic Hausdorff distance between R and B. This distance, which is the maximum distance between a point on $R \cup B$ to its closest point on the other curve, is a natural lower bound on the geodesic Fréchet distance. If $\delta < \delta_H$, we therefore immediately return that $d_{\text{F}}(R, B) > \delta$. We can compute δ_H in $\mathcal{O}((n+m) \log nm)$ time [10].

Next suppose $\delta \geq \delta_H$. For our approximate decision algorithm, we first compute the partition $\mathcal{H}$ and the entrance and exit intervals of each of its slabs. By Lemma 5, this takes $\mathcal{O}((n+m) \log nm)$ time. We iterate over the $\mathcal{O}(m)$ slabs of $\mathcal{H}$ from bottom to top. Once we consider a slab $H \in \mathcal{H}$, we have computed an (ε, δ)-safe transit entrance $p_{\text{enter}} = (x, y)$ (except if H is the bottom slab, in which case we set $p_{\text{enter}} = (1, 1)$).

Let $\hat{B}$ be the subcurve corresponding to H. If $\hat{B}$ contains only near points, we compute a (ε, δ)-safe transit exit $q_{\text{exit}} = (x', y')$ of H in $\mathcal{O}(\log nm)$ time with the algorithm of Section 3.2. Otherwise, we use the algorithm of Section 3.3, which takes $\mathcal{O}(\frac{1}{\varepsilon}(|R[x, x']| + |\hat{B}|) \log n \log nm)$ time. Both algorithms require $\mathcal{O}(n+m)$ preprocessing time. Taken over all slabs in $\mathcal{H}$, the total complexity of the subcurves $\hat{B}$ is $\mathcal{O}(m)$. This gives the following result:

▶ **Theorem 15.** *For any $\varepsilon \in (0, 1]$, there is a $(1+\varepsilon)$-approximate decision algorithm running in $\mathcal{O}(\frac{1}{\varepsilon}(n+m) \log n \log nm)$ time.*

We turn the decision algorithm into an approximate optimization algorithm with a simple binary search. For this, we show that the geodesic Fréchet distance is not much greater than δ_H. This gives an accurate "guess" of the actual geodesic Fréchet distance.

▶ **Lemma 16.** $\delta_H \leq d_{\text{F}}(R, B) \leq 3\delta_H$.

For our approximate optimization algorithm, we perform binary search over the values $\delta_H, (1+\varepsilon)\delta_H, \ldots, 3\delta_H$ and run our approximate decision algorithm with each encountered parameter. This procedure yields a $(1+\varepsilon)^2$-approximation to the geodesic Fréchet distance. Since $(1+\varepsilon)^2 \leq 1 + 3\varepsilon$ for $\varepsilon \in (0, 1]$, scaling ε by a factor of $1/3$ gives our main result:

▶ **Theorem 1.** *For any $\varepsilon \in (0, 1]$, we can compute a $(1+\varepsilon)$-approximation to $d_{\text{F}}(R, B)$ in $\mathcal{O}(\frac{1}{\varepsilon}(n + m \log n) \log nm \log \frac{1}{\varepsilon})$ time.*

4 Separated one-dimensional curves and propagating reachability

In this section we consider the following problem: Let $\bar{R}$ and $\bar{B}$ be two one-dimensional curves with n and m vertices, respectively, where $\bar{R}$ lies left of the point 0 and $\bar{B}$ right of it. We are given a set $S \subseteq \mathcal{F}_\delta(\bar{R}, \bar{B})$ of $\mathcal{O}(n+m)$ "entrances," for some $\delta \geq 0$. Also, we are given a set $E \subseteq \mathcal{F}_\delta(\bar{R}, \bar{B})$ of $\mathcal{O}(n+m)$ "potential exits." We wish to compute the subset of potential exits that are δ-reachable from an entrance. We call this procedure *propagating reachability information* from S to E. We assume that the points in S and E correspond to pairs of vertices of $\bar{R}$ and $\bar{B}$. This assumption can be met by introducing $\mathcal{O}(n+m)$ vertices, which does not increase our asymptotic running times. Additionally, we may assume that all vertices of $\bar{R}$ and $\bar{B}$ have unique values, for example by a symbolic perturbation.

The problem of propagating reachability information has already been studied by Bringmann and Künnemann [4]. In case S lies on the left and bottom sides of the parameter space and E lies on the top and right sides, they give an $\mathcal{O}((n+m)\log nm)$ time algorithm. We are interested in a more general case however, where S and E may lie anywhere in the parameter space. We make heavy use of the concept of *prefix-minima* to develop an algorithm for our more general setting that has the same running time as the one described by Bringmann and Künnemann [4]. Furthermore, our algorithm is able to actually compute a Fréchet matching between $\bar{R}$ and $\bar{B}$ in linear time (see the full version of this paper), whereas Bringmann and Künnemann require near-linear time for only the decision version.

As mentioned above, we use prefix-minima extensively for our results in this section. Prefix-minima are those vertices that are closest to the separator 0 among those points before them on the curves. In the full version of this paper, we prove that a Fréchet matching exists that matches subcurves between consecutive prefix-minima to prefix-minima of the other curve, see Figure 7 for an illustration. We call these matchings *prefix-minima matchings*. This matching will end in a bichromatic closest pair of points, and so we can compose the matching with a symmetric matching for the reversed curves.

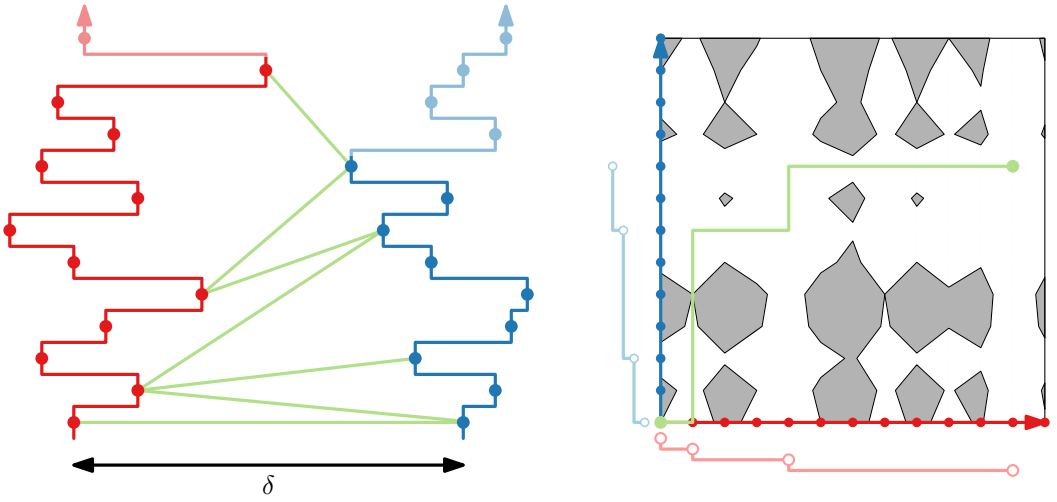


Figure 7 (left) A pair of separated, one-dimensional curves $\bar{R}$ and $\bar{B}$, drawn stretched vertically for clarity, together with a prefix-minima matching. (right) The path in $\mathcal{F}_\delta(\bar{R}, \bar{B})$ corresponding to the matching.

In Section 4.1 we introduce two geometric forests in $\mathcal{F}_\delta(\bar{R}, \bar{B})$, with leaves at S, that capture multiple prefix-minima matchings at once. It is based on *horizontal-greedy* and *vertical-greedy* matchings. We show that these forests have linear complexity and can be computed efficiently.

In Section 4.2 we do not only go forward from points in S, but also backwards from points in E using *suffix-minima*. Again we have horizontal-greedy and vertical-greedy versions. Intersections between the two prefix-minima forests and the two suffix minima forests show the existence of a δ-free path of the corresponding points in S and E, so the problem reduces to a bichromatic intersection algorithm.

4.1 Greedy paths in the free space

We wish to construct a set of canonical prefix-minima δ-matchings in the free space from which we can deduce which points in E are reachable. Naturally, we want to avoid constructing a path between every point in S and every point in E. Therefore, we investigate certain classes of prefix-minima δ-matchings that allows us to infer reachability information with just two paths per point in S and two paths per point in E. Furthermore, these paths have a combined $\mathcal{O}(n+m)$ description complexity.

We first introduce one of the greedy matchings and prove a useful property. A *horizontal-greedy δ-matching* π_{hor} is a prefix-minima δ-matching starting at a point $s = (i, j)$ that satisfies the following property: Let (i', j') be a point on π_{hor} where $\bar{R}(i')$ and $\bar{B}(j')$ are prefix-minima of $\bar{R}[i, n]$ and $\bar{B}[j, m]$. If there exists a prefix-minimum $\bar{R}(\hat{i})$ of $\bar{R}[i, n]$ after $\bar{R}(i')$, and the horizontal line segment $[i', \hat{i}] \times \{j'\}$ lies in $\mathcal{F}_\delta(\bar{R}, \bar{B})$, then either π_{hor} traverses this line segment, or π_{hor} terminates in (i', j').

For an entrance $s \in S$, let $\pi_{\text{hor}}(s)$ be the maximal horizontal-greedy δ-matching. See Figure 8 for an illustration. The path $\pi_{\text{hor}}(s)$ serves as a canonical prefix-minima δ-matching, in the sense that any point t that is reachable from s by a prefix-minima δ-matching is reachable from a point on $\pi_{\text{hor}}(s)$ through a single vertical segment:

▶ **Lemma 17.** *Let $s \in S$ and let t be a point that is reachable by a prefix-minima δ-matching from s. A point $\hat{t} \in \pi_{\text{hor}}(s)$ vertically below t exists for which the segment $\overline{\hat{t}t}$ lies in $\mathcal{F}_\delta(\bar{R}, \bar{B})$.*

A single path $\pi_{\text{hor}}(s)$ may have $\mathcal{O}(n+m)$ complexity. We would like to construct the paths for all entrances, but this would result in a combined complexity of $\mathcal{O}((n+m)^2)$. However, due to the definition of the paths, if two paths $\pi_{\text{hor}}(s)$ and $\pi_{\text{hor}}(s')$ have a point (i, j) in common, then the paths are identical from (i, j) onwards. Thus, rather than explicitly describing the paths, we instead describe their union. Specifically, the set $\bigcup_{s \in S} \pi_{\text{hor}}(s)$ forms a geometric forest $\mathcal{T}_{\text{hor}}(S)$ whose leaves are the points in S, see Figure 8. This forest has low complexity and can be constructed efficiently:

▶ **Lemma 18.** *The forest $\mathcal{T}_{\text{hor}}(S)$ has $\mathcal{O}(n+m)$ vertices.*

▶ **Lemma 19.** *We can construct a geometric graph for $\mathcal{T}_{\text{hor}}(S)$ in $\mathcal{O}((n+m)\log nm)$ time.*

4.2 Propagating reachability

Next we give an algorithm for propagating reachability information. For the algorithm, we consider three more δ-matchings that are symmetric in definition to the horizontal-greedy δ-matchings. The first is the maximal *vertical-greedy* δ-matching $\pi_{\text{ver}}(s)$, which, as the name suggests, is the maximal prefix-minima δ-matching starting at an entrance

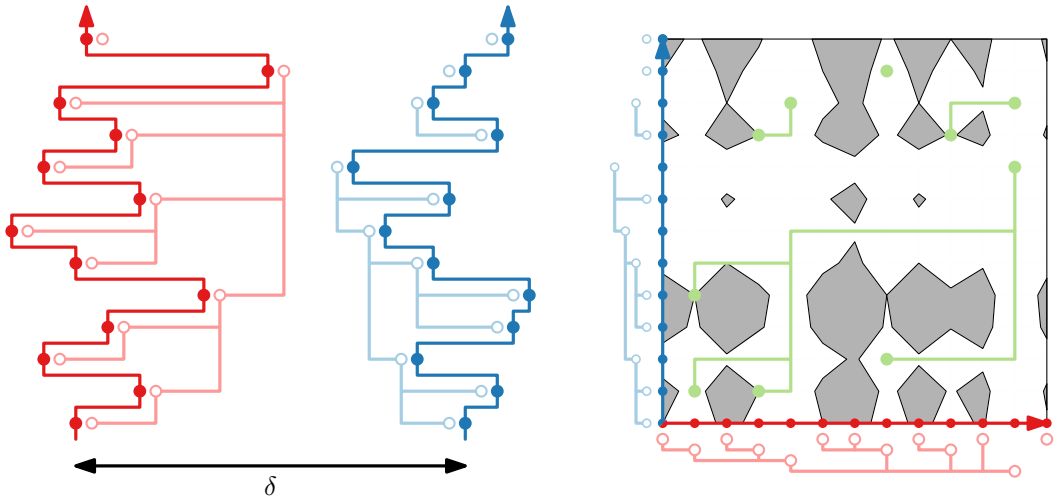


Figure 8 (left) For every vertex, its next prefix-minimum is depicted as its parent in the respective tree. (right) The horizontal-greedy δ-matchings. Paths move monotonically to the right and up.

$s \in S$ that prioritizes vertical movement over horizontal movement. The other two require a symmetric definition to prefix-minima, namely *suffix-minima*. These are the vertices closest to 0 compared to the suffix of the curve after the vertex. The maximal *reverse* horizontal- and vertical-greedy δ-matchings $\bar{\pi}_{\text{hor}}(t)$ and $\bar{\pi}_{\text{ver}}(t)$, starting at a potential exit $t \in E$, are symmetric in definition to the maximal horizontal- and vertical-greedy δ-matching, except that they move backwards, to the left and down, and their vertices correspond to suffix-minima of the curves (see Figure 9).

Consider a point $s = (i, j) \in S$ and let $t = (i', j') \in E$ be δ-reachable from s. Let $\bar{R}(i^*)$ and $\bar{B}(j^*)$ form a bichromatic closest pair of $\bar{R}[i, i']$ and $\bar{B}[j, j']$. Note that these points are unique, by our general position assumption. We prove in the full version of this paper that (i^*, j^*) is δ-reachable from s, and that t is δ-reachable from (i^*, j^*).

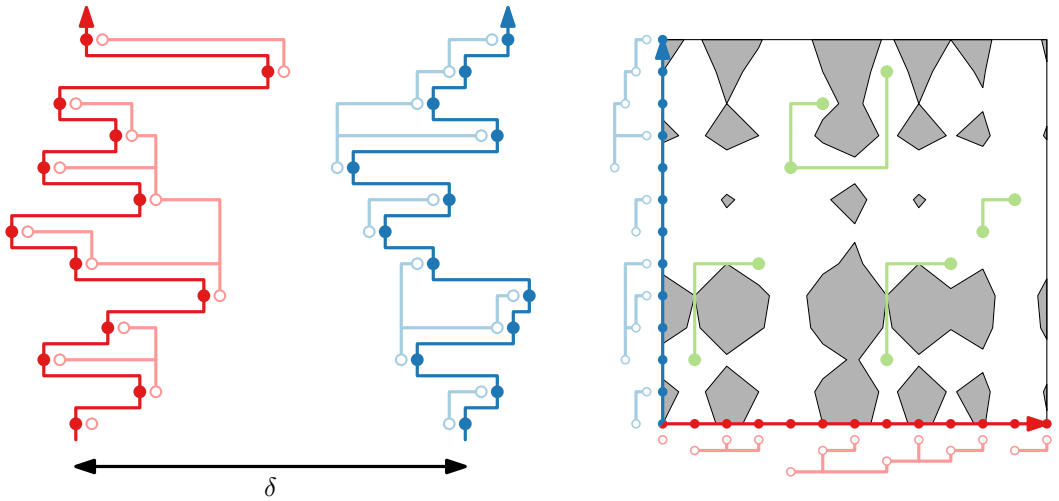


Figure 9 (left) For every vertex, its previous suffix-minimum is shown as its parent in the tree. (right) The reverse horizontal-greedy δ-matchings. Paths move monotonically to the left and down.

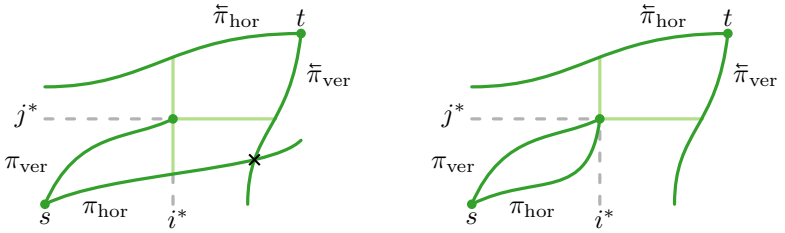


Figure 10 Two possible situations following from Lemmas 17 and 20. The paths starting at s or t are the four greedy matchings. The horizontal and vertical light green segments lie in $\mathcal{F}_\delta(R, B)$. On the right, the extensions of $\pi_{\text{hor}}(s)$ and $\pi_{\text{ver}}(s)$ respectively intersect $\overleftarrow{\pi}_{\text{ver}}(t)$ and $\overleftarrow{\pi}_{\text{hor}}(t)$.

From Lemma 17 we have that $\pi_{\text{hor}}(s)$ has points vertically below (i^*, j^*), and the vertical segment between $\pi_{\text{hor}}(s)$ and (i^*, j^*) lies in $\mathcal{F}_\delta(\bar{R}, \bar{B})$. We extend the property to somewhat predict the movement of $\pi_{\text{hor}}(s)$ near t:

▶ **Lemma 20.** *Either $\pi_{\text{hor}}(s)$ terminates in (i^*, j^*), or it contains a point vertically below t or horizontally left of t.*

Based on Lemmas 17 and 20 and their symmetric counterparts, $\pi_{\text{hor}}(s)$ and $\pi_{\text{ver}}(s)$ satisfy the properties below, and $\overleftarrow{\pi}_{\text{hor}}(t)$ and $\overleftarrow{\pi}_{\text{ver}}(t)$ satisfy symmetric properties, see Figure 10.

- $\pi_{\text{hor}}(s)$ has a point vertically below (i^*, j^*), and the vertical segment between $\pi_{\text{hor}}(s)$ and (i^*, j^*) lies in $\mathcal{F}_\delta(\bar{R}, \bar{B})$.
- $\pi_{\text{ver}}(s)$ has a point horizontally left of (i^*, j^*), and the horizontal segment between $\pi_{\text{ver}}(s)$ and (i^*, j^*) lies in $\mathcal{F}_\delta(\bar{R}, \bar{B})$.
- $\pi_{\text{hor}}(s)$ and $\pi_{\text{ver}}(s)$ both either terminate in (i^*, j^*), or contain a point vertically below t or horizontally left of t.

These properties mean that either $\pi_{\text{hor}}(s) \cup \pi_{\text{ver}}(s)$ intersects $\overleftarrow{\pi}_{\text{hor}}(t) \cup \overleftarrow{\pi}_{\text{ver}}(t)$, or the following extensions do: Let $\pi^+_{\text{hor}}(s)$ be the path obtained by extending $\pi_{\text{hor}}(s)$ with the maximum horizontal line segment in $\mathcal{F}_\delta(\bar{R}, \bar{B})$ whose left endpoint is the end of $\pi_{\text{hor}}(s)$. Define $\pi^+_{\text{ver}}(s)$ symmetrically, by extending $\pi_{\text{ver}}(s)$ with a vertical segment. Also define $\overleftarrow{\pi}^+_{\text{hor}}(s)$ and $\overleftarrow{\pi}^+_{\text{ver}}(s)$ analogously. By Lemma 17, $\pi^+_{\text{hor}}(s)$ or $\pi^+_{\text{ver}}(s)$ *must* intersect $\overleftarrow{\pi}^+_{\text{hor}}(t)$ or $\overleftarrow{\pi}^+_{\text{ver}}(t)$. Furthermore, if $\pi^+_{\text{hor}}(s)$ or $\pi^+_{\text{ver}}(s)$ intersects $\overleftarrow{\pi}^+_{\text{hor}}(t')$ or $\overleftarrow{\pi}^+_{\text{ver}}(t')$ for some potential exit $t' \in E$, then the bimonotonicity of the paths implies that t' is δ-reachable from s. Thus:

▶ **Lemma 21.** *A point $t \in E$ is δ-reachable from a point $s \in S$ if and only if $\pi^+_{\text{hor}}(s) \cup \pi^+_{\text{ver}}(s)$ intersects $\overleftarrow{\pi}^+_{\text{hor}}(t') \cup \overleftarrow{\pi}^+_{\text{ver}}(t')$.*

Recall that $\mathcal{T}_{\text{hor}}(S)$ represents all paths $\pi_{\text{hor}}(s)$. We augment $\mathcal{T}_{\text{hor}}(S)$ to represent all paths $\pi^+_{\text{hor}}(s)$. For this, we take each root vertex p and compute the maximal horizontal segment $\overline{pq} \subseteq \mathcal{F}_\delta(\bar{R}, \bar{B})$ that has p as its left endpoint. We compute this segment in $\mathcal{O}(\log n)$ time after $\mathcal{O}(n \log n)$ time preprocessing (for details, refer to the full version of this paper). We then add q as a vertex to $\mathcal{T}_{\text{hor}}(S)$, and add an edge from p to q.

Let $\mathcal{T}^+_{\text{hor}}(S)$ be the augmented graph. We define the graphs $\mathcal{T}^+_{\text{ver}}(S)$, $\overleftarrow{\mathcal{T}}^+_{\text{hor}}(E)$ and $\overleftarrow{\mathcal{T}}^+_{\text{ver}}(E)$ analogously. The four graphs have a combined complexity of $\mathcal{O}(n+m)$ and can be constructed in $\mathcal{O}((n+m)\log nm)$ time. Our algorithm computes the edges of $\overleftarrow{\mathcal{T}}^+_{\text{hor}}(E)$ and $\overleftarrow{\mathcal{T}}^+_{\text{ver}}(E)$ that intersect an edge of $\mathcal{T}^+_{\text{hor}}(S)$ or $\mathcal{T}^+_{\text{ver}}(S)$. We do so with a standard sweepline algorithm:

▶ **Lemma 22.** *Given sets of n "red" and m "blue," axis-aligned line segments in $\mathbb{R}^2$, we can report all segments that intersect a segment of the other color in $\mathcal{O}((n+m)\log nm)$ time.*

Suppose we have computed the set of edges $\mathcal{E}$ of $\overleftarrow{\mathcal{T}}^{+}_{\text{hor}}(E)$ and $\overleftarrow{\mathcal{T}}^{+}_{\text{ver}}(E)$ that intersect an edge of $\mathcal{T}^{+}_{\text{hor}}(S)$ or $\mathcal{T}^{+}_{\text{ver}}(S)$. We store $\mathcal{E}$ in a red-black tree, so that we can efficiently retrieve and remove edges from this set. Let $e \in \mathcal{E}$ and suppose e is an edge of $\overleftarrow{\mathcal{T}}^{+}_{\text{hor}}(E)$. Let μ be the top-right vertex of e. All potential exits of E that are stored in the subtree of μ are reachable from a point in S. We traverse the entire subtree of μ, deleting every edge we find from $\mathcal{E}$. Every point in E we find is marked as reachable. In this manner, we obtain:

▶ **Theorem 23.** *Let $\bar{R}$ and $\bar{B}$ be two separated one-dimensional curves with n and m vertices. Let $\delta \geq 0$, and let $S, E \subseteq \mathcal{F}_\delta(\bar{R}, \bar{B})$ be sets of $\mathcal{O}(n+m)$ points. We can compute the set of all points in E that are δ-reachable from points in S in $\mathcal{O}((n+m)\log nm)$ time.*

5 Conclusion

We studied computing the approximate geodesic Fréchet distance of two curves R and B that bound a simple polygon P, one clockwise and one counterclockwise, whose endpoints meet. Our algorithm is approximate, though the only approximate parts are for matching the far points and turning the decision algorithm into an optimization algorithm. Doing so exactly and in strongly subquadratic time remains an interesting open problem.

Our algorithm extends to the case where R and B do not cover the complete boundary of the polygon. In other words, the start and endpoints of R and B need not coincide. Geodesics between points on R and B must stay inside P. In this case, $k = |P|$ can be much greater than $n + m - 2$, which influences the preprocessing and query times of various data structures we use. The running time then becomes: $\mathcal{O}(k + \frac{1}{\varepsilon}(n + m\log n)\log k \log \frac{1}{\varepsilon})$.

References

1 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

2 Lotte Blank and Anne Driemel. A faster algorithm for the Fréchet distance in 1D for the imbalanced case. In *Proc. 32nd Annual European Symposium on Algorithms (ESA)*, volume 308 of *LIPIcs*, pages 28:1–28:15, 2024. doi:10.4230/LIPICS.ESA.2024.28.

3 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proc. 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

4 Karl Bringmann and Marvin Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *International Journal of Computational Geometry & Applications*, 27(1-2):85–120, 2017. doi:10.1142/S0218195917600056.

5 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017. doi:10.1007/s00454-017-9878-7.

6 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proc. 30th Annual Symposium on Discrete Algorithms (SODA)*, pages 2887–2901, 2019. doi:10.1137/1.9781611975482.179.

7 Erin W. Chambers, Éric Colin de Verdière, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Computational Geometry*, 43(3):295–311, 2010. doi:10.1016/J.COMGEO.2009.02.008.

8 Siu-Wing Cheng, Haoqiang Huang, and Shuo Zhang. Constant approximation of Fréchet distance in strongly subquadratic time. *CoRR*, abs/2503.12746, 2025. doi:10.48550/arXiv.2503.12746.

9 Connor Colombe and Kyle Fox. Approximating the (continuous) Fréchet distance. In *Proc. 37th International Symposium on Computational Geometry (SoCG)*, pages 26:1–26:14, 2021. doi:10.4230/LIPIcs.SoCG.2021.26.

10 Atlas F. Cook IV and Carola Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Transactions on Algorithms*, 7(1):9:1–9:19, 2010. doi:10.1145/1868237.1868247.

11 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012. doi:10.1007/s00454-012-9402-z.

12 Alon Efrat, Leonidas J. Guibas, Sariel Har-Peled, Joseph S. B. Mitchell, and T. M. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete & Computational Geometry*, 28(4):535–569, 2002. doi:10.1007/S00454-002-2886-1.

13 Sariel Har-Peled, Amir Nayyeri, Mohammad R. Salavatipour, and Anastasios Sidiropoulos. How to walk your dog in the mountains with no magic leash. *Discrete & Computational Geometry*, 55(1):39–73, 2016. doi:10.1007/S00454-015-9737-3.

14 Thijs van der Horst, Marc van Kreveld, Tim Ophelders, and Bettina Speckmann. A near-linear time exact algorithm for the l_1-geodesic Fréchet distance between two curves on the boundary of a simple polygon. To appear at the Algorithms and Data Structures Symposium (WADS), 2025.

15 Thijs van der Horst, Marc J. van Kreveld, Tim Ophelders, and Bettina Speckmann. A subquadratic n^{ε}-approximation for the continuous Fréchet distance. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1759–1776, 2023. doi:10.1137/1.9781611977554.ch67.

Efficient Contractions of Dynamic Graphs – With Applications

Monika Henzinger
Institute of Science and Technology, Klosterneuburg, Austria

Evangelos Kosinas
Institute of Science and Technology, Klosterneuburg, Austria

Robin Münk
Technical University of Munich, Germany

Harald Räcke
Technical University of Munich, Germany

Abstract

A non-trivial minimum cut (NMC) sparsifier is a multigraph $\hat{G}$ that preserves all non-trivial minimum cuts of a given undirected graph G. We introduce a flexible data structure for fully dynamic graphs that can efficiently provide an NMC sparsifier upon request at any point during the sequence of updates. We employ simple dynamic forest data structures to achieve a fast from-scratch construction of the sparsifier at query time. Based on the strength of the adversary and desired type of time bounds, the data structure comes with different guarantees. Specifically, let G be a fully dynamic simple graph with n vertices and minimum degree δ. Then our data structure supports an insertion/deletion of an edge to/from G in $n^{o(1)}$ worst-case time. Furthermore, upon request, it can return w.h.p. an NMC sparsifier of G that has $O(n/\delta)$ vertices and $O(n)$ edges, in $\hat{O}(n)$ time. The probabilistic guarantees hold against an adaptive adversary. Alternatively, the update and query times can be improved to $\tilde{O}(1)$ and $\tilde{O}(n)$ respectively, if amortized-time guarantees are sufficient, or if the adversary is oblivious. Throughout the paper, we use $\tilde{O}$ to hide polylogarithmic factors and $\hat{O}$ to hide subpolynomial (i.e., $n^{o(1)}$) factors.

We discuss two applications of our new data structure. First, it can be used to efficiently report a cactus representation of all minimum cuts of a fully dynamic simple graph. Building this cactus for the NMC sparsifier instead of the original graph allows for a construction time that is sublinear in the number of edges. Against an adaptive adversary, we can with high probability output the cactus representation in worst-case $\hat{O}(n)$ time. Second, our data structure allows us to efficiently compute the maximal k-edge-connected subgraphs of undirected simple graphs, by repeatedly applying a minimum cut algorithm on the NMC sparsifier. Specifically, we can compute with high probability the maximal k-edge-connected subgraphs of a simple graph with n vertices and m edges in $\tilde{O}(m + n^2/k)$ time. This improves the best known time bounds for $k = \Omega(n^{1/8})$ and naturally extends to the case of fully dynamic graphs.

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

Keywords and phrases Graph Algorithms, Cut Sparsifiers, Dynamic Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.36

Related Version *Full Version*: `https://arxiv.org/abs/2509.05157`

Funding *Monika Henzinger and Evangelos Kosinas*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (MoDynStruct, No. 101019564) and the Austrian Science Fund (FWF) grant DOI 10.55776/Z422 and grant DOI 10.55776/I5982.
Harald Räcke and Robin Münk: This project has received funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 498605858.



© Monika Henzinger, Evangelos Kosinas, Robin Münk, and Harald Räcke;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Graph sparsification is an algorithmic technique that replaces an input graph G by another graph $\hat{G}$, which has fewer edges and/or vertices than G, but preserves (or approximately preserves) a desired graph property. Specifically, for connectivity-based and flow-based problems, a variety of static sparsifiers exist, that approximately maintain cut- or flow-values in G [2, 3, 4, 8, 20, 26, 27].

Let n denote the number of vertices, m the number of edges, and δ the minimum degree of the input graph. In an undirected simple graph it is possible to reduce the number of vertices to $O(n/\delta)$ and the number of edges to $O(n)$ both in randomized and deterministic time $\tilde{O}(m)$ while preserving the value of all non-trivial minimum cuts *exactly* [10, 19, 21]. A minimum cut is considered *trivial* if one of its sides consists of a single vertex and we call the resulting multigraph a *non-trivial minimum cut sparsifier (NMC sparsifier)*.

Most sparsification algorithms assume a static graph. Maintaining an NMC sparsifier in a fully dynamic setting, where a sequence of edge insertions and deletions can be arbitrarily interleaved with requests to output the NMC sparsifier, was only recently studied: Goranci, Henzinger, Nanongkai, Saranurak, Thorup, and Wulff-Nilsen [11] show how to maintain an NMC sparsifier in a fully dynamic graph w.h.p. in $\tilde{O}(n)$ worst-case update and query time (as a consequence of Theorem 3.7 in [11]), under the assumption of an oblivious adversary. Additionally, Theorem 4.5 in [11] gives a deterministic algorithm that outputs an NMC sparsifier of size $\tilde{O}(m/\delta)$ in $\tilde{O}(m/\delta)$ worst-case query time with $\tilde{O}(\delta^3)$ amortized update time.

1.1 Our Results

In this paper, we present the first data structure for providing an NMC sparsifier of a fully dynamic graph that supports sublinear *worst-case* update and query time and works against an *adaptive* adversary. As a first application, we give an improved fully dynamic algorithm that outputs a cactus representation of *all* minimum cuts of the current graph upon request. Additionally, we use our data structure to compute the maximal k-edge connected subgraphs of an undirected simple graph with an improvement in running time for large values of k.

In more detail, we provide a data structure for a fully dynamic graph that can be updated in *worst-case* time $\hat{O}(1)$ and that allows (at any point during the sequence of edge updates) to construct an NMC sparsifier in worst-case time $\hat{O}(n)$. The probabilistic guarantees work against an *adaptive* adversary. If the update time is relaxed to be *amortized* or if the adversary is *oblivious*, the update time can be improved to $\tilde{O}(1)$ and the query time to $\tilde{O}(n)$.

Our basic approach is to maintain suitable data structures in a dynamically changing graph that allow the execution of the *static* NMC sparsifier algorithm based on random 2-out contractions proposed by Ghaffari, Nowicki, and Thorup [10] in time $\hat{O}(n)$ instead of $\tilde{O}(m)$. Our main insight is that this speedup can be achieved by maintaining

1. a dynamic spanning forest data structure (DSF) of the input graph G and
2. a dynamic cut set data structure (DCS), where the user determines which edges belong to a (not necessarily spanning) forest F of G, and, given a vertex v, the data structure returns an edge that leaves the tree of F that contains v (if it exists).

We show that these two data structures suffice to construct an NMC sparsifier of the current graph in the desired time bound. Put differently, we can avoid processing all edges of G in order to build the NMC sparsifier, and only spend time that is roughly proportional to the size of the sparsifier.

Note that the NMC sparsifier is computed from scratch every time it is requested and no information of a previously computed sparsifier is maintained throughout the dynamic updates of the underlying graph. This ensures the probabilistic guarantees hold against an adaptive adversary if the guarantees of the chosen DSF and DCS data structures do.

Our main result is the following theorem.

▶ **Theorem 1.** *Let G be a fully dynamic simple graph that currently has n nodes and minimum degree $\delta > 0$. There is a data structure that outputs an NMC sparsifier of G that has $O(n/\delta)$ vertices and $O(n)$ edges w.h.p. upon request. Each update and query takes either*

1. *worst-case $\hat{O}(1)$ and $\hat{O}(n)$ time respectively w.h.p., assuming an* adaptive *adversary, or*
2. *amortized $\tilde{O}(1)$ and $\tilde{O}(n)$ time respectively w.h.p., assuming an* adaptive *adversary, or*
3. *worst-case $\tilde{O}(1)$ and $\tilde{O}(n)$ time respectively w.h.p., assuming an* oblivious *adversary.*

Recall that $\tilde{O}$ hides polylogarithmic factors and $\hat{O}$ hides subpolynomial (i.e., $n^{o(1)}$) factors. The three cases of Theorem 1 result from using different data structures to realize our required DSF and DCS data structures. We show that with minimal overhead both data structures can be reduced to a dynamic minimum spanning forest data structure, for which many constructions have been proposed in the literature. For Case 1, we make use of the fully dynamic minimum spanning forest algorithm of Chuzhoy, Gao, Li, Nanongkai, Peng, and Saranurak [6] – the only spanning forest data structure known so far that can provide worst-case time guarantees against an adaptive adversary. Case 2 is the result of substituting the deterministic data structure of Holm, de Lichtenberg, and Thorup [15] and case 3 results from using the randomized data structure of Kapron, King, and Mountjoy [16] instead.

As a first application we can efficiently provide a cactus representation of the minimum cuts of a simple graph upon request in the fully dynamic setting. The cactus representation of all minimum cuts of a static graph is known to be computable in near linear time $\tilde{O}(m)$ using randomized algorithms [12, 18, 21]. With deterministic algorithms, the best known time bound is $m^{1+o(1)}$ [12]. We are not aware of any previous work on providing the cactus for fully dynamic graphs in sublinear time per query. Specifically, we show the following:

▶ **Theorem 2.** *Let G be a fully dynamic simple graph with n vertices. There is a data structure that w.h.p. reports a cactus representation of all minimum cuts of G. Each update and query takes either*

1. *worst-case $\hat{O}(1)$ and $\hat{O}(n)$ time respectively w.h.p., assuming an* adaptive *adversary, or*
2. *amortized $\tilde{O}(1)$ and $\tilde{O}(n)$ time respectively w.h.p., assuming an* adaptive *adversary, or*
3. *worst-case $\tilde{O}(1)$ and $\tilde{O}(n)$ time respectively w.h.p., assuming an* oblivious *adversary.*

Theorem 2 provides an improvement over one of the main technical components in [11]. Specifically, Goranci et al. [11], provide a method to efficiently maintain an NMC sparsifier of a dynamic simple graph, based on the 2-out contractions of Ghaffari et al. [10]. However, their main technical result (Theorem 3.7 in [11]) has several drawbacks. First, it needs to know an estimated upper bound $\hat{\delta}$ on the minimum degree of any graph that occurs during the sequence of updates of G. Second, they try to *maintain* the sparsifier during the updates. This results in an update time $\tilde{O}(\hat{\delta})$, and forces them to "hide" their sparsifier from an adversary, i.e., they can expose their sparsifier only if they work against an oblivious adversary. Thus, to make their minimum cut algorithm work against an adaptive adversary, they return only the *value* of the minimum cut. In contrast, our algorithm computes a sparsifier from scratch upon request, and can therefore provide a cactus representation of *all* minimum cuts, even against an adaptive adversary.

Notice that we provide a different trade-off in reporting the minimum cut: We have an update time of $\hat{O}(1)$ and a query time of $\hat{O}(n)$, whereas Theorem 1.1 of Goranci et al. [11] has an update time of $\tilde{O}(n)$ and query time $O(1)$. This trade-off is never worse (modulo

Table 1 Best known time bounds for computing the maximal k-edge-connected subgraphs in undirected graphs in the static setting. The $\tilde{O}$ expression hides polylogarithmic factors.

Algorithm	Time	Type	Range of k
Chechik et al., Forster et al. [5, 7]	$\tilde{O}(m + k^{O(k)}n^{3/2})$	Det.	$k \in \mathbb{N}$
Forster et al. [7]	$\tilde{O}(m + k^3 n^{3/2})$	Las Vegas Rnd.	$k \in \mathbb{N}$
Henzinger et al. [13]	$\tilde{O}(n^2)$	Det.	$k \in \mathbb{N}$
Thorup, Georgiadis et al. [28, 9]	$\tilde{O}(m + k^8 n^{3/2})$	Det.	$k = \log^{O(1)} n$
Saranurak and Yuan [25]	$O(m + kn^{1+o(1)})$	Det.	$k = \log^{o(1)} n$
Nalam and Saranurak [24]	$\tilde{O}(m \cdot \min\{m^{3/4}, n^{4/5}\})$	Monte Carlo Rnd.	$k \in \mathbb{N}$
This paper	$\tilde{O}(m + n^2/k)$	Monte Carlo Rnd.	$k \in \mathbb{N}$

the subpolynomial factors) if one caches the result of a query and answers queries from the cache if the graph did not change. Furthermore, upon request, we can provide a minimum cut explicitly, and not just its value.

As a second application of our main result, we improve the time bounds for computing the (vertex sets of the) maximal k-edge-connected subgraphs in a simple undirected graph. Specifically, we have the following:

▶ **Theorem 3.** *Let G be a simple graph with n vertices and m edges, and let k be a positive integer. We can compute the maximal k-edge-connected subgraphs of G w.h.p. in $\tilde{O}(m+n^2/k)$ time w.h.p.*

For comparison, Table 1 gives an overview of the best known algorithms for computing the maximal k-edge-connected subgraphs in undirected graphs. Thorup [28] does not deal with the computation of the maximal k-edge-connected subgraphs, but the result is a consequence of his fully dynamic minimum cut algorithm as observed in [9]. The algorithm in [24] is the only one that holds for weighted graphs (with integer weights), and it has no dependency on k in the time bound. Notice that our algorithm improves over all prior results for $k = \Omega(n^{1/8})$ and $m = \Omega(n^{8/7})$.

With a reduction to simple graphs, this implies the following bounds for computing the maximal k-edge-connected subgraphs of multigraphs.

▶ **Corollary 4.** *Let G be a multigraph with n vertices and m edges, and let k be a positive integer. We can compute the maximal k-edge-connected subgraphs of G w.h.p. in $\tilde{O}(m + k^2n + kn^2)$ time w.h.p.*

Notice that Corollary 4 provides an improvement compared to the other algorithms in Table 1, depending on k and the density of the graph. For example, if δ and ϵ are two parameters satisfying $1/4 \leq \delta \leq 1$, $16/15 \leq \epsilon \leq 2$ and $\delta \leq \epsilon - 6/5$, then we have an improvement in the regime where $m = \Theta(n^\epsilon)$ and $k = \Theta(n^\delta)$ against all previous algorithms in Table 1 that work for multigraphs (i.e., except for Henzinger et al. [13], which works only for simple graphs).

Finally, the method that we use in Theorem 3 for computing the maximal k-edge-connected subgraphs in static graphs extends to the case of dynamic graphs.

▶ **Theorem 5.** *Let G be a fully dynamic simple graph with n vertices. There is a data structure that can provide the maximal k-edge-connected subgraphs of G at any point in time, with high probability, for any integer $k < n$. Each update and query takes either*

1. *worst-case* $\hat{O}(1)$ *and* $\hat{O}(n^2/k)$ *time respectively w.h.p, assuming an* adaptive *adversary,*
2. *amortized* $\tilde{O}(1)$ *and* $\tilde{O}(n^2/k)$ *time respectively w.h.p, assuming an* adaptive *adversary, or*
3. *worst-case* $\tilde{O}(1)$ *and* $\tilde{O}(n^2/k)$ *time respectively w.h.p, assuming an* oblivious *adversary.*

The results by Aamand et al. [1] and Saranurak and Yuan [25] provide algorithms for maintaining the maximal k-edge-connected subgraphs in *decremental* graphs. Georgiadis et al. [9] provide a fully dynamic algorithm that given two vertices determines whether they belong to the same k-edge connected subgraph in time $O(1)$. Their worst-case update time is $\tilde{O}(T(n,k))$, where $T(n,k)$ is the running time of any algorithm for static graphs that is used internally by the algorithm (see values in the time column of Table 1 with the additive term m removed). Thus, as in the case for dynamic minimum cut, our algorithm provides a different trade-off, with fast updates and slower query time. Notice that Theorem 5 provides an improvement over [9] (for any function $T(n,k)$ corresponding to an algorithm from Table 1) when k is sufficiently large (i.e., when $k = \Omega(n^{1/8})$).

Due to the space constraints, more details on related work, all omitted proofs, and some further details can be found in the full version of this paper.

2 Preliminaries

In this paper, we consider only undirected, unweighted graphs. A graph is called *simple* if it contains no parallel edges, conversely it is a multigraph if it does. We use common graph terminology and notation that can be found e.g. in [23]. Let $G = (V, E)$ be a graph. Throughout, we use n and m to denote the number of vertices and edges of G, respectively. We use $V(G)$ and $E(G)$ to denote the set of vertices and edges, respectively, of G; that is, $V = V(G)$ and $E = E(G)$. A *subgraph* of G is a graph of the form (V', E'), where $V' \subseteq V$ and $E' \subseteq E$. A *spanning* subgraph of G is a subgraph of the form (V, E') that contains at least one incident edge to every vertex of G that has degree > 0. If X is a subset of vertices of G, we let $G[X]$ denote the induced subgraph of G on the vertex set X. Thus, $V(G[X]) := X$, and the edge set of $G[X]$ is $\{e \in E \mid \text{ both endpoints of } e \text{ are in } X\}$. If E' is a set of edges of G, we let $G[E'] := (V, E')$.

A connected component of G is a maximal connected subgraph of G. A set of edges of G whose removal increases the number of connected components of G is called a *cut* of G. The size $|C|$ is called the *value* of the cut. A cut C with minimum value in a connected graph G is called a *minimum cut* of G. In this case, $G \setminus C = (V, E \setminus C)$ consists of two connected components. If one of them is a single vertex, then C is called a *trivial minimum cut*.

An *NMC sparsifier* of G is a multigraph H on the same vertex set as G that preserves all non-trivial minimum cuts of G, in the sense that the number of edges leaving any non-trivial minimum cut is the same in H as it is in G.

Contractions of graphs. Let $E' \subseteq E$ be a set of edges of a graph $G = (V, E)$, and let $C_1, \ldots, C_t$ be the connected components of the graph $G' = (V, E')$. The *contraction* $\hat{G}$ induced by E' is the graph derived from G by contracting each C_i, for $i \in \{1, \ldots, t\}$, into a single node. We ignore possible self-loops, but we maintain distinct edges of G that have their endpoints in different connected components of G'. Hence, $\hat{G}$ may be a multigraph even though G is a simple graph. Furthermore, there is a natural injective correspondence from the edges of $\hat{G}$ to the edges of G. We say that an edge of G is *preserved* in $\hat{G}$ if its endpoints are in different connected components of G' (it corresponds to an edge of $\hat{G}$).

A *random 2-out contraction* of G is a contraction of G that is induced by sampling from every vertex v of G two edges incident to it (independently, with repetition allowed) and setting E' to be the edges sampled in this way. Thus, E' satisfies $|E'| \leq 2|V(G)|$. The importance of considering 2-out contractions is demonstrated in the following theorem of Ghaffari et al. [10].

▶ **Theorem 6** (rephrased weaker version of Theorem 2.3 in [10]). *A random 2-out contraction of a simple graph G with n vertices and minimum degree δ has $O(n/\delta)$ vertices, with high probability, and preserves any fixed non-trivial minimum cut of G with constant probability.*

Preserving small cuts via forest decompositions. Nagamochi and Ibaraki [22] have shown the existence of a sparse subgraph that maintains all cuts of the original graph with value up to k, where k is an integer ≥ 1. Specifically, given a graph $G = (V, E)$ with n vertices and m edges, there is a spanning subgraph $H = (V, E')$ of G with $|E'| \leq k(n-1)$ such that every set of edges $C \subseteq E$ with $|C| < k$ is a cut of G if and only if C is a cut of H. A graph H with this property is given by a *k-forest decomposition* of G, which is defined as follows. First, we let F_1 be a spanning forest of G. Then, for every $i \in \{2, \ldots, k\}$, we recursively let F_i be a spanning forest of $G \setminus (F_1 \cup \cdots \cup F_{i-1})$. Then we simply take the union of the forests $H = F_1 \cup \cdots \cup F_k$. A naive implementation of this idea takes time $O(k(n+m))$. However, this construction can be completed in linear time by computing a maximum adjacency ordering of G [23].

Maximal k-edge-connected subgraphs. Given a graph G and a positive integer k, a *maximal k-edge-connected subgraph* of G is a subgraph of the form $G[S]$, where: (1) $G[S]$ is connected, (2) the minimum cuts of $G[S]$ have value at least k, and (3) S is maximal with this property. The first two properties can be summarized by saying that $G[S]$ is k-edge-connected, which equivalently means that we have to remove at least k edges in order to disconnect $G[S]$. It is easy to see that the vertex sets $S_1, \ldots, S_t$ that induce the maximal k-edge-connected subgraphs of G form a partition of V.

A note on probabilistic guarantees. Throughout the paper we use the abbreviation w.h.p. (with high probability) to mean a probability of success at least $1 - O(\frac{1}{n^c})$, where c is a fixed constant chosen by the user and n denotes the number of vertices of the graph. The choice of c only affects the values of two parameters q and r that are used internally by the algorithm of Ghaffari et al. [10], which our result is based on. Note that the specification "w.h.p" may be applied either to the running time or to the correctness (or to both).

3 Outline of Our Approach

Our main contribution is a new data structure that outputs an NMC sparsifier of a fully dynamic graph upon request. The idea is to compute the NMC sparsifier from scratch every time it is requested. For this we adapt the construction by Ghaffari et al. [10] to compute a random 2-out contraction of the graph at the current time. We can achieve a speedup of the construction time by maintaining just two data structures for dynamic forests throughout the updates of the graph.

3.1 Updates: Data Structures for Dynamic Forests

As our fully dynamic graph G changes, we rely on efficient data structures for the following two problems, which we call the "dynamic spanning forest" problem (DSF) and the "dynamic cutset" problem (DCS). In DSF, the goal is to maintain a spanning forest F of a dynamic graph G. Specifically, the DSF data structure supports the following operations.

- `insert`(e). Inserts a new edge e to G. If e has endpoints in different trees of F, then it is automatically included in F, and this event is reported to the user.
- `delete`(e). Deletes the edge e from G. If e was a tree edge in F, then it is also removed from F, and a new replacement edge is selected among the remaining edges of G in order to reconnect the trees of F that got disconnected. If a replacement edge is found, it automatically becomes a tree edge in F, and it is output to the user.

The DCS problem is a more demanding variant of DSF. Here the goal is to maintain a (not necessarily spanning) forest F of the graph, but we must also be able to provide an edge that connects two different trees if needed. Specifically, the DCS data structure supports the following operations.

- $\texttt{insert}_G(e)$. Inserts a new edge e to G.
- $\texttt{insert}_F(e)$. Inserts an edge e to F, if e is an edge of G that has endpoints in different trees of F. Otherwise, F stays the same.
- $\texttt{delete}_G(e)$. Deletes the edge e from G. If e is a tree edge in F, it is also deleted from F.
- $\texttt{delete}_F(e)$. Deletes the edge e from F (or reports that e is not an edge of F).
- `find_cutedge`(v). Returns an edge of G (if it exists) that has one endpoint in the tree of F that contains v, and the other endpoint outside of that tree.

The main difference between DSF and DCS is that in DCS the user has absolute control over the edges of the forest F. In both problems, the most challenging part is finding an edge that connects two distinct trees of F. In DCS this becomes more difficult because the data structure must work with the forest that the user has created, whereas in DSF the spanning forest is managed internally by the data structure. Both of these problems can be solved by reducing them to the dynamic minimum spanning forest (MSF) problem, as shown in the following Lemma 7.

▶ **Lemma 7.** *The DSF problem can be reduced to the DCS problem within the same time bounds. The DCS problem can be reduced to dynamic MSF with an additive $O(\log n)$ overhead for every operation.*

To realize these two data structures deterministically with worst-case time guarantees, the only available solution at the moment is to make use of the reduction to the dynamic MSF problem given in Lemma 7 and then employ the dynamic MSF data structure of Chuzhoy et. al. [6], which supports every update operation in worst-case $\hat{O}(1)$ time. Alternatively, one can solve both DSF and DCS deterministically with *amortized* time guarantees using the data structures of Holm et. al. [15]. In that case, every update for DSF can be performed in amortized $O(\log^2 n)$ time, and every operation for DCS can be performed in amortized $O(\log^4 n)$ time, by reduction to dynamic MSF. If one is willing to allow randomness, one can solve both DSF and DCS in worst-case polylogarithmic time per operation, under the assumption of an oblivious adversary with the data structure by Kapron et al. [16].

These different choices for realizing the dynamic MSF data structure give the following lemma.

▶ **Lemma 8.** *The DSF and DCS data structures can be realized in either*
1. *deterministic worst-case* $\hat{O}(1)$ *update time, or*
2. *deterministic amortized* $\tilde{O}(1)$ *update time, or*
3. *worst-case* $\tilde{O}(1)$ *update time, assuming an* oblivious *adversary.*

To handle the updates on G, any insertion or deletion of an edge is also applied in both the DSF and the DCS data structure. In particular, for DCS we only use the $\texttt{insert}_G$ and $\texttt{delete}_G$ operations and keep its forest empty. At query time, we will build the DCS forest from scratch and then fully delete it again. In order for this to be efficient, DCS needs to have already processed all edges of G.

3.2 Queries: Efficiently Contracting a Dynamic Graph

The NMC sparsifier that we output for each query is a random 2-out contraction of the graph at the current time. For this construction, we use our maintained forest data structures and build upon the algorithm of Ghaffari et al. [10], who prove the following.

▶ **Theorem 9** (Weaker version of Theorem 2.1 of Ghaffari et al. [10])**.** *Let G be a simple graph with n vertices, m edges, and minimum degree δ. In $O(m \log n)$ time we can create a contraction $\hat{G}$ of G that has $O(n/\delta)$ vertices and $O(n)$ edges w.h.p., and preserves all non-trivial minimum cuts of G w.h.p.*

Note that in particular, the contracted graph $\hat{G}$ created by the above theorem is an NMC sparsifier, as desired. We first sketch the algorithm that yields Theorem 9 as described by Ghaffari et al. [10]. Then we outline how we can adapt and improve this construction for dynamic graphs, making use of the DSF and DCS data structures. Specifically, to answer a query we show that we can build $\hat{G}$ in time proportional to the size of $\hat{G}$, which may be much lower than $O(m \log n)$. The details and a formal analysis can be found in Section 4.

Random 2-out Contractions of Static Graphs

Ghaffari et al.'s algorithm [10] works as follows (see also Algorithm 1). First, it creates a collection of $q = O(\log n)$ random 2-out contractions $G_1, \ldots, G_q$ of G, where every G_i is created with independent random variables. Now, according to Theorem 2.4 in [10], each G_i for $i \in \{1, \ldots, q\}$, has $O(n/\delta)$ vertices w.h.p., and preserves any fixed non-trivial minimum cut of G with constant probability.

In a second step, they compute a $(\delta + 1)$-forest decomposition $\hat{G}_i$ of G_i, for every $i \in \{1, \ldots, q\}$, in order to ensure that $\hat{G}_i$ has $O(\delta \cdot (n/\delta)) = O(n)$ edges w.h.p. Because G has minimum degree δ, every non-trivial minimum cut has value at most δ. Hence, each $\hat{G}_i$ still maintains every fixed non-trivial minimum cut with constant probability.

Finally, they select a subset of edges $E_{\text{con}} \subseteq E(G)$ to contract by a careful *"voting"* process. Specifically, for every edge e of G, they check if it is an edge of at least r graphs from $\hat{G}_1, \ldots, \hat{G}_q$, where r is a carefully chosen parameter. If e does not satisfy this property, then it is included in E_{con}, the set of edges to be contracted. In the end, $\hat{G}$ is given by contracting all edges from E_{con}.

An Improved Construction using Dynamic Forests

We now give an overview of our algorithm for efficiently constructing the NMC sparsifier $\hat{G}$. It crucially relies on having the DSF and DCS data structures initialized to contain the edges of the current graph G. For a fully dynamic graph, this is naturally ensured at query

Algorithm 1 Construction of the contracted graph $\hat{G}$ in Theorem 9.

```
input: a simple graph G in adjacency list representation
choose parameters q, r = Θ(log n) according to [10]
compute the minimum degree δ of G
for i ← 1 to q do
    construct a 2-out contraction G_i of G
foreach i ∈ {1, ..., q} do
    construct a (δ + 1)-forest decomposition Ĝ_i of G_i
let E_con ← ∅ // a set of edges of G to contract
foreach edge e of G do
    if e is preserved in less than r graphs from Ĝ_1, ..., Ĝ_q then
        E_con ← E_con ∪ {e}
return the graph obtained from G by contracting the edges in E_con
```

time by maintaining the two forest data structures throughout the updates, as described in Section 3.1. Since the goal is to output a graph $\hat{G}$ with $O(n/\delta)$ vertices and $O(n)$ edges w.h.p., we aim for an algorithm that takes roughly $O(n)$ time. The process is broken up into three parts.

Part 1. First, we compute the 2-out contractions $G_1, \ldots, G_q$ for $q = O(\log n)$. Each G_i can be computed by sampling, for every vertex v of G, two random edges incident to v (independently, with repetition allowed). Since the graph G is dynamic, the adjacency lists of the vertices also change dynamically, and therefore this is not an entirely trivial problem. However, in the full version of this paper we provide an efficient solution that relies on elementary data structures. Specifically, we can support every graph update in worst-case constant time, and every query for a random incident edge to a vertex in worst-case $\tilde{O}(1)$ time. Notice that each G_i, for $i \in \{1, \ldots, q\}$, is given by contracting a set E_i of $O(n)$ edges of G.

Part 2. Next, we separately compute a $(\delta + 1)$-forest decomposition $\hat{G}_i$ of G_i, for every $i \in \{1, \ldots, q\}$. Each $\hat{G}_i$ is computed by only accessing the edges of E_i plus the edges of the output (which are $O(n)$ w.h.p.). For this, we rely on the DCS data structure. Since in DCS we have complete control over the maintained forest F, we can construct it in such a way, that every connected component of $G[E_i]$ induces a subtree of F. Notice that the connected components of $G[E_i]$ correspond to the vertices of G_i. This process of modifying F takes $O(|E_i|) = O(n)$ update operations on the DCS data structure. Then, we have established the property that the tree edges that connect vertices of different connected components of $G[E_i]$ correspond to a spanning tree of G_i. Afterwards, we just repeatedly remove the spanning tree edges, and find replacements using the DCS data structure. These replacements constitute a new spanning forest of G_i. Thus, we just have to repeat this process $\delta + 1$ times to construct the desired $(\delta + 1)$-forest decomposition. Note that every graph $\hat{G}_i$ is constructed in time roughly proportional to its size (which is $O(n)$ w.h.p.). Any overhead comes from the use of the DCS data structure.

Part 3. Finally, we construct the graph $\hat{G}$ by contracting the edges $E_{<r}$ of G that appear in less than r graphs from $\hat{G}_1, \ldots, \hat{G}_q$. From Ghaffari et al. ([10], Theorem 2.4) it follows that $|E \setminus E_{<r}| = |E_{\geq r}| = O(qn) = O(n \log n)$. In the following we provide an algorithm for constructing $\hat{G}$ with only $O(n + |E_{\geq r}|)$ operations of the DSF data structure.

We now rely on the spanning forest F of G that is maintained by the DSF data structure. We pick the edges of F one by one, and check for every edge whether it is contained in $E_{<r}$ (it is easy to check for membership in $E_{<r}$ in $O(\log n)$ time). If it is not contained in $E_{<r}$, then we store it as a *candidate edge*, i.e., an edge that possibly is in $\hat{G}$. In this case, we also remove it from F, and the DSF data structure will attempt to fix the spanning forest by finding a replacement edge. Otherwise, if $e \in E_{<r}$, then we "fix" it in F, and do not process it again.

In the end all "fixed" edges of F form a spanning forest of the connected components of $G[E_{<r}]$. Note that the algorithm makes continuous progress: in each step it either identifies an edge of $E_{\geq r}$, or it "fixes" an edge of F (which can happen at most $n-1$ times). Thus, it performs $O(n + |E_{\geq r}|) = \tilde{O}(n)$ DSF operations. Since we have arrived at a spanning forest of $G[E_{<r}]$, we can build $\hat{G}$ by contracting the candidate edges stored during this process.

4 Analysis

Our main technical tools are Propositions 10 and 11. Proposition 10 allows us to create a forest decomposition of a contracted graph without accessing all the edges of the original graph, but roughly only those that are included in the output, and those that we had to contract in order to get the contracted graph. Proposition 11 has a somewhat reverse guarantee: it allows us to create a contracted graph by accessing only the edges that are preserved during the contraction (plus only a small number of additional edges).

In this section we assume that G is a fully dynamic graph that currently has n vertices. We also assume that we have maintained a DCS and a DSF data structure on G, which support every operation in U_{CS} and U_{SF} time, respectively (cf. Section 3.1). The proofs of all theorems and propositions in this section can be found in the full version of this paper.

4.1 A k-Forest-Decomposition of the Contraction

Given a set of edges to contract, the following proposition shows how to compute a k forest decomposition of the induced contracted graph. Crucially, the contracted graph H does not have to be given explicitly, but it is sufficient to know only a set of edges whose contraction yields H. Thus, the running time of the construction is independent of the number of edges of H. The whole procedure is shown in Algorithm 2.

▶ **Proposition 10.** *Let H be a contraction of G that results from contracting a given set E_{con} of edges in G. Let further k be a positive integer and n_H be the number of vertices in H. Then we can construct a k-forest decomposition of H in time $O\big((n + kn_H) \cdot U_{\text{CS}} + |E_{con}| \log n\big)$.*

Note that the number of DCS operations depends only on (1) the number n of vertices of G, (2) the number n_H of vertices of H, and (3) the number k of forests that we want to build.

4.2 Building the Contracted Graph

A contracted graph can naturally be computed from its defining set of contraction edges E_{con} in time proportional to the size of this set $|E_{\text{con}}|$. Recall that in our case E_{con} is the result of the "voting" procedure across all generated δ-forest decompositions (cf. Section 3.2), which is rather expensive to compute. We hence use a different construction that does not need to know E_{con} explicitly. Instead, it relies on an efficient oracle to certify that a given edge is not contained in E_{con}.

Algorithm 2 Compute a k-forest decomposition of the graph that is formed by contracting a set of edges E' of G.

```
let F be the empty DCS forest
foreach edge e ∈ E' do
    if the endpoints of e belong to different trees of F then
        DCS.insert_F(e)
let V be a set that consists of one vertex from every tree of F
set S ← ∅
for i ← 1 to k do
    set L ← ∅ // L will contain the edges of the current spanning forest
    foreach v ∈ V do
        let e ← DCS.find_cutedge(v)
        while e ≠ ⊥ do
            DCS.insert_F(e), and append e to L
            e ← DCS.find_cutedge(v)
    foreach e ∈ L do
        DCS.delete_G(e), and append e to S
use DCS.delete_F to remove all the edges from F
foreach e ∈ S do
    DCS.insert_G(e) // restore DCS to its original state
return S
```

▶ **Proposition 11.** *Let H be a contraction of G that results from contracting a set E_{con} of edges in G and let E_{pre} be the set of edges of G that are preserved in H. Suppose that there is a set of edges E_{check} with $E_{pre} \subseteq E_{check}$ and $E_{check} \cap E_{con} = \emptyset$, for which we can check membership in time μ. Then we can construct H in time $O(n\mu + |E_{check}| \cdot (U_{\mathrm{SF}} + \mu))$.*

Note that the number of DSF operations is proportional to the number of edges in E_{check} (the set E_{con} is not required as input to the algorithm). Thus, this algorithm becomes more efficient if only few edges are contained in $E(G) \setminus E_{\mathrm{con}}$ (since $E_{\mathrm{check}} \subseteq E(G) \setminus E_{\mathrm{con}}$). In our application, we will have $\mu = O(\log n)$.

4.3 Constructing an NMC sparsifier

We can now state our result for computing an NMC sparsifier of a simple graph using Propositions 10 and 11. Compare this to Theorem 9 and note how Theorem 12 requires initialized DSF and DCS data structures but has a running time that is independent of the number of edges in G. An outline of this procedure is given in Section 3.2.

▶ **Theorem 12.** *Let G be a simple graph with n vertices that has minimum degree $\delta > 0$ and is maintained in a DSF and a DCS data structure. Then, with high probability we can construct an NMC sparsifier of G that has $O(n/\delta)$ vertices and $O(n)$ edges. W.h.p this takes $\tilde{O}(n \cdot (U_{\mathrm{SF}} + U_{\mathrm{CS}}))$ time.*

4.4 Fully Dynamic Graphs

The result of Theorem 12 can easily be extended to fully dynamic simple graphs by maintaining the DSF and DCS data structures throughout the updates of the graph. These forest data structures can be realized in different ways, as described in Section 3.1. Depending on this choice we get a different result, and this is how we derive Theorem 1.

If G is disconnected, all minimum cuts have value 0 and an NMC sparsifier H is by definition only required to have no edges between different connected components of G. Crucially, this implies that there is no guarantee that any information of the minimum cuts within each connected component of G is preserved in H. In this case, however, we can easily strengthen the result by applying Theorem 12 to each connected component of G individually.

5 Applications

5.1 A Cactus Representation of All Minimum cuts in Dynamic Graphs

It is well-known that a graph with n vertices has $O(n^2)$ minimum cuts, all of which can be represented compactly with a data structure of $O(n)$ size that has the form of a cactus graph (for more details, see [23]). As a first immediate application of our main Theorem 1, we show how the NMC sparsifier $\hat{G}$ of any fully dynamic simple graph G can be used to quickly compute this cactus representation. This result is summarized in Theorem 2; see the full version of this paper for a detailed construction.

To obtain just a single minimum cut of G, one can apply the deterministic minimum cut algorithms of [14, 19] on $\hat{G}$, which yields a minimum cut C of $\hat{G}$ in time $\tilde{O}(|\hat{G}|) = \tilde{O}(n)$. To transform C into a minimum cut of G, we compare its size with the minimum degree δ of any node in G: If $|C| \leq \delta$, then we get a minimum cut of G by simply mapping the edges from C back to the corresponding edges of G. Otherwise, a minimum cut of G is given by any vertex of G that has degree δ (which is easy to maintain throughout the updates).

5.2 Computing the Maximal k-Edge-Connected Subgraphs

The data structure of Theorem 1 can also be used to improve the time bounds for computing the maximal k-edge-connected subgraphs of a simple graph, in for cases where k is a sufficiently large polynomial of n. Specifically, we get an improvement for $k = \Omega(n^{1/8})$, c.f. Section 1.

A general strategy for computing these subgraphs is the following. Let G be a simple graph with n vertices and m edges, and let k be a positive integer. The basic idea is to repeatedly find and remove cuts with value less than k from G. First, as long as there are vertices with degree less than k, we repeatedly remove them from the graph. Now we are left with a (possibly disconnected) graph where every non-trivial connected component has minimum degree at least k. If we perform a minimum cut computation on a non-trivial connected component S of G, there are two possibilities: either the minimum cut is at least k, or we will have a minimum cut C with value less than k. In the first case, S is confirmed to be a maximal k-edge-connected subgraph of G. In the second case, we remove C from S, and thereby split it into two connected components S_1 and S_2. Then we recurse on both S_1 and S_2. Since the graph is simple and S has minimum degree at least k, it is a crucial observation that both S_1 and S_2 contain at least k vertices (see e.g. [19]). Therefore the number of nodes decreases by at least k with every iteration and hence the total recursion depth is $O(n/k)$.

The minimum cut computation takes time $T_{\text{mc}} = \tilde{O}(m)$ [17], hence the worst-case running time of this approach is $\Theta(n/k \cdot T_{\text{mc}}) = \tilde{O}(mn/k)$. We can use Theorem 1 to bring the time down to $\tilde{O}(m + n^2/k)$ w.h.p.

▶ **Theorem 3.** *Let G be a simple graph with n vertices and m edges, and let k be a positive integer. We can compute the maximal k-edge-connected subgraphs of G w.h.p. in $\tilde{O}(m+n^2/k)$ time w.h.p.*

Through a reduction to simple graphs, Theorem 3 implies Corollary 4, which is a similar result with slightly worse time bounds for undirected *multigraphs*. Finally, the method that establishes Theorem 3 naturally extends to the case of fully dynamic simple graphs, which yields Theorem 5.

References

1 Anders Aamand, Adam Karczmarz, Jakub Lacki, Nikos Parotsidis, Peter M. R. Rasmussen, and Mikkel Thorup. Optimal decremental connectivity in non-sparse graphs. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 6:1–6:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.6.

2 Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012. doi:10.1137/090772873.

3 András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 47–55. ACM, 1996. doi:10.1145/237814.237827.

4 András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015. doi:10.1137/070705970.

5 Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F. Italiano, Veronika Loitzenbauer, and Nikos Parotsidis. Faster algorithms for computing maximal 2-connected subgraphs in sparse directed graphs. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1900–1918. SIAM, 2017. doi:10.1137/1.9781611974782.124.

6 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1158–1167. IEEE, 2020. doi:10.1109/FOCS46700.2020.00111.

7 Sebastian Forster, Danupon Nanongkai, Liu Yang, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2046–2065. SIAM, 2020. doi:10.1137/1.9781611975994.126.

8 Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. *SIAM J. Comput.*, 48(4):1196–1223, 2019. doi:10.1137/16M1091666.

9 Loukas Georgiadis, Giuseppe F. Italiano, Evangelos Kosinas, and Debasish Pattanayak. On maximal 3-edge-connected subgraphs of undirected graphs. *CoRR*, abs/2211.06521, 2022. doi:10.48550/arXiv.2211.06521.

10 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1260–1279. SIAM, 2020. doi:10.1137/1.9781611975994.77.

11 Gramoz Goranci, Monika Henzinger, Danupon Nanongkai, Thatchaphol Saranurak, Mikkel Thorup, and Christian Wulff-Nilsen. Fully dynamic exact edge connectivity in sublinear time. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 70–86. SIAM, 2023. doi:10.1137/1.9781611977554.ch3.

12 Zhongtian He, Shang-En Huang, and Thatchaphol Saranurak. Cactus representation of minimum cuts: Derandomize and speed up. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 1503–1541. SIAM, 2024. doi:10.1137/1.9781611977912.61.

13 Monika Henzinger, Sebastian Krinninger, and Veronika Loitzenbauer. Finding 2-edge and 2-vertex strongly connected components in quadratic time. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 713–724. Springer, 2015. doi:10.1007/978-3-662-47672-7_58.

14 Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. *SIAM J. Comput.*, 49(1):1–36, 2020. doi:10.1137/18M1180335.

15 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

16 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142. SIAM, 2013. doi:10.1137/1.9781611973105.81.

17 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. doi:10.1145/331605.331608.

18 David R. Karger and Debmalya Panigrahi. A near-linear time algorithm for constructing a cactus representation of minimum cuts. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 246–255. SIAM, 2009. doi:10.1137/1.9781611973068.28.

19 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019. doi:10.1145/3274663.

20 Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. *SIAM Journal on Computing*, 47(6):2315–2336, 2018. doi:10.1137/16M1061850.

21 On-Hei Solomon Lo, Jens M. Schmidt, and Mikkel Thorup. Compact cactus representations of all non-trivial min-cuts. *Discret. Appl. Math.*, 303:296–304, 2021. doi:10.1016/J.DAM.2020.03.046.

22 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. doi:10.1007/BF01758778.

23 Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic Aspects of Graph Connectivity*, volume 123 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2008. doi:10.1017/CBO9780511721649.

24 Chaitanya Nalam and Thatchaphol Saranurak. Maximal k-edge-connected subgraphs in weighted graphs via local random contraction. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 183–211. SIAM, 2023. doi:10.1137/1.9781611977554.ch8.

25 Thatchaphol Saranurak and Wuwei Yuan. Maximal k-edge-connected subgraphs in almost-linear time for small k. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 92:1–92:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.92.

26 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. doi:10.1137/080734029.

27 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011. doi:10.1137/08074489X.

28 Mikkel Thorup. Fully-dynamic min-cut. *Comb.*, 27(1):91–127, 2007. doi:10.1007/s00493-007-0045-2.

Hardness of Computation of Quantum Invariants on 3-Manifolds with Restricted Topology

Henrique Ennes
Université Côte d'Azur, Centre INRIA, Sophia Antipolis, France

Clément Maria
Université Côte d'Azur, Centre INRIA, Sophia Antipolis, France

Abstract

Quantum invariants in low-dimensional topology offer a wide variety of valuable invariants about knots and 3-manifolds, presented by explicit formulas that are readily computable. Their computational complexity has been actively studied and is tightly connected to topological quantum computing. In this article, we prove that for any 3-manifold quantum invariant in the Reshetikhin-Turaev model, there is a deterministic polynomial time algorithm that, given as input an arbitrary closed 3-manifold M, outputs a closed 3-manifold M' with the same quantum invariant, such that M' is hyperbolic, contains no low genus embedded incompressible surface, and is presented by a strongly irreducible Heegaard diagram. Our construction relies on properties of Heegaard splittings and the Hempel distance. At the level of computational complexity, this proves that the hardness of computing a given quantum invariant of 3-manifolds is preserved even when severely restricting the topology and the combinatorics of the input. This positively answers a question raised by Samperton [44].

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Geometric topology; Theory of computation → Problems, reductions and completeness

Keywords and phrases 3-manifold, Heegaard splitting, Hempel distance, Quantum invariant, polynomial time reduction

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.37

Related Version *Full Version*: `https://arxiv.org/abs/2503.02814`

Funding This work has been partially supported by the ANR project ANR-20-CE48-0007 (AlgoKnot) and the project ANR-15-IDEX-0001 (UCA JEDI). It has also been supported by the French government, through the France 2030 investment plan managed by the Agence Nationale de la Recherche, as part of the "UCA DS4H" project, reference ANR-17-EURE-0004.

Acknowledgements We are grateful to Lukas Woike, Alex He, and Nicolas Nisse for helpful discussions on Vafa's theorem, triangulations of Heegaard splittings, and general complexity theory, respectively. We also thank the anonymous reviewers for their thoughtful comments and constructive feedback, which greatly contributed to improving the clarity, rigor, and quality of this paper.

1 Introduction

Quantum invariants are topological invariants defined using tools from physics, explicitly, from topological quantum field theories (TQFTs). These invariants have become of interest for modeling phenomena in condensed matter physics [5, 19], topological quantum computing [29], and experimental mathematics [14, 38], where many deep conjectures remain open. Thanks to their diversity and discriminating power to distinguish between non-equivalent topologies, they have also played an important role in the constitution of censuses of knots and 3-manifolds [12]. The invariants are constructed from the data of a fixed algebraic object, called a *modular category*, and a topological support, and take the form of a partition function, whose value depends solely on the topological type of the support and not on its combinatorial


© Henrique Ennes and Clément Maria;
licensed under Creative Commons License CC-BY 4.0

presentation. Generally, quantum invariants are defined for presentations of either knots or 3-manifolds, although there are types of invariants, such as the Reshetikhin-Turaev, whose definitions naturally encompass both objects.

The complexity of the exact and approximate computation of these invariants has attracted much interest, particularly in connection with quantum complexity classes. Most non-trivial quantum invariants turn out to be #P-hard to compute and, sometimes, even #P-hard to approximate [3, 20, 30] within reasonable precision. This is the case of the Jones polynomial for knots [20, 30] and the Turaev-Viro invariant of 3-manifolds associated with the Fibonacci category [3, 20]. On the positive side, polynomial time *quantum* algorithms exist for computing weak forms of approximations [2, 4, 20] and efficient *parameterized* algorithms have been designed [11, 13, 34, 36, 39] leading to polynomial time algorithms on certain families of instances. For the latter, the topology of the input knot or 3-manifold plays a central role in measuring the computational complexity of the problem, either directly with running times depending strongly on some topological parameter [15, 39], or indirectly where *simple topologies* guarantee the existence of simple combinatorial representations [23, 25, 37] that can be, in turn, algorithmically exploited.

There are other instances, however, where the hardness of computing the invariants is preserved even if the topology and combinatorics of the input are restricted. In [30], Kuperberg shows that, for certain quantum invariants that are *hard to approximate* on *links*, the hardness is preserved when restricted to *knots*. In a follow-up work, Samperton [44] proves that if computing a quantum invariant is hard for all input diagrams of any knot, then the computation remains hard when restricting the input to *hyperbolic* knots given by diagrams with a minimal number of crossings. In this article, we follow a similar path to both [30] and [44], this time for quantum invariants of 3-manifolds, by proving the hardness of computing invariants of irreducible presentations and hyperbolic manifolds.

We follow the strategy of Samperton [44], which consists of using Vafa's theorem [54] to efficiently complicate the topological structure of the input without changing the invariant. Nonetheless, while Samperton's process involves adding extra crossings to the knot diagrams, we increase the *Hempel distance* of a *Heegaard diagram* of some 3-manifold. Although there exists an extensive catalog of algorithms to increase Hempel distances [17, 22, 26, 28, 33, 42, 56], to the best of our knowledge, our work is the first to 1. explicitly compute the involved complexities, ensuring polynomial time; and, 2. keep some 3-manifold invariant constant throughout the process.

The main result is expressed in Theorem 1, whose precise statement can be found in Section 4. Here and throughout the paper, we denote by Σ_g the *closed surface of genus* g (unique up to homeomorphism) with some fixed orientation and assume $g \geq 2$. When referring to a general compact surface, potentially with boundary, we use Σ.

▶ **Theorem 1.** *Let $\mathcal{C}$ be a modular category and M a closed 3-manifold represented by a Heegaard diagram $(\Sigma_g, \alpha, \beta)$ of complexity m. There is a deterministic algorithm that constructs, in time $O(\mathrm{poly}(m, g))$ and uniformly on the choice of $\mathcal{C}$, a strongly irreducible Heegaard diagram $(\Sigma_{g+1}, \alpha', \beta')$ representing a hyperbolic 3-manifold M' that shares with M the Reshetikhin-Turaev invariant over $\mathcal{C}$. Moreover, for a fixed choice of $k \in \mathbb{N}$, M' has no embedded orientable and incompressible surface of genus at most $2k$.*

▶ Remark 2. Hyperbolicity has historically been used as both a simplifying structure and an intermediate step for algorithms on 3-manifold, see for example [31, 50]. Similarly to [44] for knots, our result proves that hyperbolicity is of no help for the computational complexity of the quantum invariant. On the other hand, when producing hard instances of 3-manifolds in computational topology – e.g., in complexity reduction [1, 6] or the construction of

combinatorially involved manifolds [24] – it is common to produce *Haken* 3-manifolds with low genus incompressible surfaces (generally, tori). It is an important open question to understand the hardness of computation for non-Haken 3-manifolds. Our result shows that the computational complexity of quantum invariants is preserved even when getting rid of low-genus incompressible surfaces.

The paper is divided into four parts: a review of background material (Section 2), the demonstration of some auxiliary algorithms (Section 3), the proof our main result (Section 4), and some illustration of its computational consequences for the hardness of computing quantum invariants (Section 5). Due to the conference's constraints, some of the demonstrations were omitted from this current article, but can be found in the extended version of the paper[1].

2 Background material

In the following review, we assume acquaintance with the basic ideas from geometric topology, such as boundaries, compactness, homeomorphisms, (free) homotopies, isotopies, and manifolds. For these topics, we refer the reader to [49] and [51]. Moreover, we shall use, without properly defining, some well-known concepts belonging to the theories of curves and surfaces, which can be found in [18].

2.1 Curves in surfaces

An *arc* in a surface Σ is the image of a proper embedding of the interval in Σ (i.e., its endpoints lie both on $\partial\Sigma$). Similarly, a *simple closed curve* is the image of a proper embedding $S^1 \hookrightarrow \Sigma$. We will often refer to a simple closed curve by only *closed curve* or just *curve*. A *multicurve* is a finite collection of disjoint properly embedded simple curves in Σ. We denote by $\#\gamma$ its number of connected components. Whenever possible, we distinguish simple curves from multicurves by using Greek letters to represent the latter.

A curve in the surface will be called *essential* if not homotopic to a point (or, equivalently, if it does not bound a disk on the surface), a puncture, or a boundary component. A multicurve is essential if all its components are essential. Unless otherwise stated, all curves and multicurves will be assumed essential.

We will often be interested not in a curve s, but in the equivalence classes of s up to *isotopies* in Σ_g, $[s]$. Being essential is preserved under isotopies, so we naturally extend the definition of essential curves to their isotopy classes. For two curves s, t in some surface Σ, their *geometric intersection number* is defined as the minimal number of their intersection points up to isotopy, that is $i(s,t) = \min\{|s' \cap t'| : s' \in [s], t' \in [t]\}$. Two curves in a surface Σ_g are isotopic if and only if they are (free) homotopic [18].

The *curve graph* of a closed surface Σ_g, $C(\Sigma_g)$, is the graph whose vertices are isotopy classes of essential curves and any two vertices $[s]$ and $[t]$ are connected by an edge if and only if $i(s,t) = 0$. The usual graph distance d defines a metric on the vertices of $C(\Sigma_g)$ that is interpreted as $d([s],[t]) = n$ implying the existence a sequence of essential curves $r_0, r_1, \ldots, r_n$ with $r_0 \in [s]$, $r_n \in [t]$, and $r_i \cap r_{i+1} = \emptyset$ for $0 \leq i < n$. The definitions related to the curve graph can be naturally extended to curves by identifying the elements of an isotopy class to the same vertex; in particular, $d(r,s) = 0$ is equivalent to r and s being isotopic.

[1] Available at `https://arxiv.org/abs/2503.02814`.

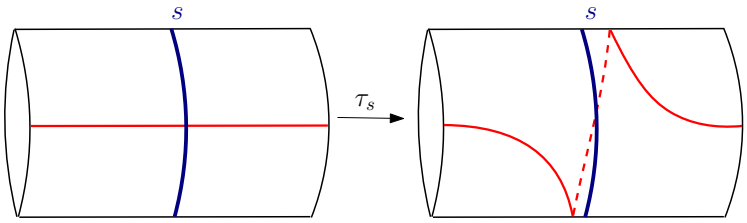


Figure 1 Illustration of the action of a Dehn twist about a curve s (blue) on some curve transversal to it (red). The homeomorphism is only different from the identity on a regular neighborhood of s.

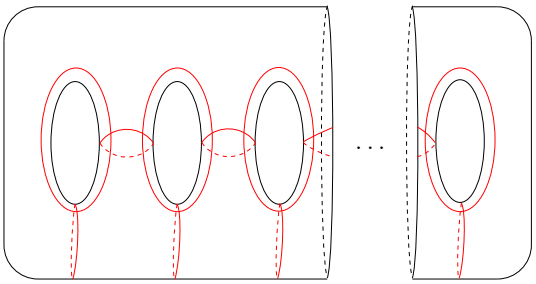


Figure 2 Diagram representing the Lickorish curves in a closed surface $\Sigma_g, g \geq 2$.

We recall that for each surface Σ_g, its *mapping class group*, $\mathrm{Mod}(\Sigma_g)$, is the group of orientation-preserving homeomorphisms $\Sigma_g \to \Sigma_g$ up to isotopies. Its canonical action on the surface conserves the geometric intersection number between curves [18], therefore acting isometrically on $(C(\Sigma_g), d)$ by the induced map $\phi \cdot [s] = \phi_*([s])$. The mapping class groups always contain the (isotopy classes of) *Dehn twists*, homeomorphisms $\tau_s : \Sigma_g \to \Sigma_g$ defined by cutting off a local neighborhood of the (multi)curve s of Σ_g and gluing it back with a 2π counterclockwise twist, determined by the orientation of the surface (Figure 1). In particular, for each $g \geq 2$, $\mathrm{Mod}(\Sigma_g)$ is generated as a group by (isotopy classes of) Dehn twists about the $3g-1$ recursively defined *Lickorish curves* in Σ_g (Figure 2) [32]. We can, consequently, always assume that an element of $\mathrm{Mod}(\Sigma_g)$ is of form $\phi = \tau_{s_r}^{n_r} \circ \cdots \circ \tau_{s_1}^{n_1}$, where $n_i \in \mathbb{Z}$ and s_i are Lickorish curves for $1 \leq i \leq r$.

2.2 Curves on handlebodies and Heegaard splittings

An essential multicurve $\gamma \subset C(\Sigma_g)$ is a *(full) system* if no two components are isotopic to each other and $\Sigma_g - \gamma$ is a union of $\#\gamma - g + 1$ punctured spheres (here the minus sign indicates cutting the surface along γ). The minimum number of connected components that a full system may have is g, in which case $\Sigma_g - \gamma$ is a $2g$-punctured sphere and γ is called a *minimum* system. On the other extreme, a system γ is *maximum* or a *pants decomposition* when $\#\gamma = 3g-3$ and $\Sigma_g - \gamma$ is the union of $2g-2$ thrice punctured spheres, also known as *pairs of pants* (see Figure 4). When every connected component of a system γ is isotopic to a connected component of another system γ', we say that γ is *contained* in γ' and denote that by $\gamma \subseteq \gamma'$. A minimum system γ can always be extended to a pants decomposition $\rho \supseteq \gamma$, see Theorem 16.

The genus g *handlebody* defined for a full system γ in the surface Σ_g is the 3-manifold

$$V_\gamma = \Sigma_g \times [0,1] \cup_{\gamma \times \{0\}} \text{2-handles} \cup \text{3-handles}$$

built by attaching the 2-handles along the curves γ in $\Sigma \times \{0\}$ and then filling any resulting S^2 boundary component with 3-handles. By construction $\partial V_\gamma = \Sigma_g$. Because the curves in

the system γ are assumed to be essential, each of its components will bound (non-trivial) compression disks in V_γ: they will be *meridians* of the handlebody. There are, however, many other meridians in V_γ, as it will be implied by the following definition.

▶ **Definition 3** (Disk graph and equivalent systems). *Let V_γ be a handlebody constructed over Σ_g. Then the disk graph of V_γ, K_γ, is the subgraph of $C(\Sigma_g)$ whose vertices represent (isotopy classes of) meridians of V_γ. We say that two full systems, γ and γ', (potentially with $\#\gamma \neq \#\gamma'$) are equivalent if $\gamma' \subset K_\gamma$ and $\gamma \subset K_{\gamma'}$.*

Note that γ is equivalent to γ' if and only if they define the same handlebody. In particular, if $\gamma \subset \rho$, γ and ρ are equivalent.

In a seminal work, Hempel [22] studied the metric d of the disk graph canonically inherited from $C(\Sigma_g)$. This inspires the next definition. Here and throughout, whenever $A, B \subset C(\Sigma)$ and r is a curve in Σ, we let $d(r, A) = \min\{d(r, s) : s \in A\}$ and $d(A, B) = \min\{d(s, t) : s \in A, t \in B\}$. We assume a curve s on a handlebody V_γ to be fully contained in ∂V_γ.

▶ **Definition 4** (Diskbusting curves). *An essential curve s on a handlebody V_γ is said to be diskbusting if $d(s, K_\gamma) \geq 2$, that is, if s intersects all meridians of V_γ.*

[52] provides a combinatorial condition to verify if a curve on a handlebody is diskbusting, which we quote using the language of [56]. Before, however, we will need a definition.

▶ **Definition 5** (Seams and seamed curves). *An arc in a pair of pants P is called a* seam *if it has endpoints on two distinct components of ∂P. A curve s in a surface Σ_g with a pants decomposition ρ is said to be* seamed *for ρ if, for every component P of $\Sigma_g - \rho$, $s \cap P$ has at least one copy of each of the three types (up to isotopy) of seams in P.*

▶ **Theorem 6** (Theorems 1 of [52], Theorem 4.11 of [56]). *Let s be a curve on the handlebody V_γ. Then s is diskbusting in V_γ if and only if there is a pants decomposition ρ equivalent to γ such that s is seamed for ρ.*

A closed 3-manifold M is said to have a *Heegaard splitting* if it is the union of two handlebodies intersecting only at their common boundary. It is well-known [45] that there exists, for every closed 3-manifold M, a tuple $(\Sigma_g, \alpha, \beta)$ called a *Heegaard diagram (of genus g)*, where α and β are two full systems with the same cardinality in Σ_g, and a Heegaard splitting $M = V_\alpha \cup_{\Sigma_g} V_\beta$. Note, on the other hand, that neither the Heegaard diagram nor the splitting is unique: for example, isotopies to α or β yield the same splitting, whereas given a diagram $(\Sigma_g, \alpha, \beta)$, one can define another splitting for the same manifold, this time of diagram $(\Sigma_{g+1}, \alpha \cup \{c\}, \beta \cup \{c'\})$, where c and c' are curves with $i(c, c') = 1$ fully contained in the extra handle. This last process, known as *stabilization*, is topologically equivalent to directly summing a copy of S^3 to the original manifold M.

Some properties of 3-manifolds can be read straight-up from their Heegaard splittings. For example, a splitting $(\Sigma_g, \alpha, \beta)$ is called *irreducible* if V_α and V_β do not share a meridian. Haken's lemma [49, Theorem 6.3.5] implies that reducible closed 3-manifolds cannot have irreducible Heegaard splittings. Similarly, a splitting $(\Sigma_g, \alpha, \beta)$ is *strongly irreducible* if there are no two essential disjoint curves a and b in Σ_g such that a is a meridian of V_α and b is a meridian of V_β. Every strongly irreducible splitting is irreducible, but the converse is not true (Haken manifolds provide a list of counterexamples [49]).

Given two handlebodies V_α and V_β, we define their *Hempel distance* by $d(K_\alpha, K_\beta)$. In [22], Hempel argued that this distance could be seen as a measure of the complexity of a Heegaard splitting $V_\alpha \sqcup_{\Sigma_g} V_\beta$, which is translated into the following theorem, whose proof (a simple application of results by others) can be found in the full version of the paper.

▶ **Theorem 7.** *Let $(\Sigma_g, \alpha, \beta)$ be a Heegaard diagram of distance $d(K_\alpha, K_\beta) = k$. Then*
- *$k \geq 1$ if and only if $(\Sigma_g, \alpha, \beta)$ is irreducible;*
- *$k \geq 2$ if and only if $(\Sigma_g, \alpha, \beta)$ is strongly irreducible;*
- *if $k \geq 3$, then M is hyperbolic;*
- *M has no orientable and incompressible embedded surface of genus smaller than $2k$.*

▶ Remark 8. Scharlemann and Tomova [48] proved that, if $V_{\alpha'} \cup_{\Sigma_{g'}} V_{\beta'}$ is Heegaard splitting of genus smaller than $k/2$, then $V_\alpha \cup_{\Sigma_g} V_\beta$ is *isotopic* to $V_{\alpha'} \cup_{\Sigma_{g'}} V_{\beta'}$, potentially after finitely many stabilizations. In particular, if $k > 2g + 2$, the splitting is of minimum genus. Unfortunately, as we will see in the proof of Theorem 1, our algorithm is not polynomial time as function of k, which means that it does not imply an efficient reduction to a minimal genus splitting for every input.

Our proof of Theorem 1 will mainly consist of increasing the Hempel distance so that the hypotheses of Theorem 7 are satisfied. For such, we will extensively use the next two theorems due to Yoshizawa.

▶ **Theorem 9** (Theorem 5.8 of [56])**.** *Consider the full systems of curves α and β in Σ_g and $n = \max\{1, d(K_\alpha, K_\beta)\}$. Let $d_i = d(K_i, s)$ for $i = \alpha, \beta$, assume $d_i \geq 2$ and $d_\alpha + d_\beta - 2 > n$. Then, for any $k \in \mathbb{Z}^+$,*

$$\min(k, d_\alpha + d_\beta - 2) \leq d(K_\alpha, K_{\tau_s^{k+n+2}(\beta)}) \leq d_\alpha + d_\beta. \tag{1}$$

▶ **Theorem 10** (Theorem 6.2 of [56])**.** *Let $\gamma = \{c_1, \ldots, c_g\}$ be full in Σ_g and ρ a pants decomposition containing γ. Suppose s is seamed for ρ and define the multicurve $\tau_s^2(\gamma)$ of components $d_1, \ldots, d_g$. Then $d(K_\gamma, \tau_{d_g}^2 \circ \cdots \circ \tau_{d_1}^2(c_1)) \geq 3$.*

2.3 Quantum invariants for 3-manifolds

We will not review the technical construction of TQFTs here, referring the interested reader to [53]. For our purposes, it is enough to know that, for a fixed choice of modular category $\mathcal{C}$ (again, refer to [53] for the definition), the TQFT associates to every closed 3-manifold M a complex scalar known as its *Reshetikhin-Turaev* (RT) invariant. The RT invariant can be given as a function of a Heegaard diagram $(\Sigma_g, \alpha, \beta)$ of M and is denoted by $\langle M \rangle_{\mathcal{C}}^{RT}$ or $\langle (\Sigma_g, \alpha, \beta) \rangle_{\mathcal{C}}^{RT}$, depending on whether we want to emphasize the manifold or the diagram.

The algebraic structure imposed by the modular category sets some constraints on the quantum invariants. The next theorem, for example, is already somewhat folklore in the literature (see [41]) since it implies that non-homeomorphic 3-manifolds may share RT invariants. It can be deduced from Theorem 5.1 of [16] and [43, 53].

▶ **Theorem 11** (Vafa's theorem for 3-manifold TQFTs)**.** *Let $\mathcal{C}$ be a modular category. Then there is an $N \in \mathbb{Z}^+$, depending only on $\mathcal{C}$, such that, for all $k \in \mathbb{Z}$ and every curve s in Σ_g, $\langle (\Sigma_g, \alpha, \beta) \rangle_{\mathcal{C}}^{RT} = \langle (\Sigma_g, \alpha, \tau_s^{kN}(\beta)) \rangle_{\mathcal{C}}^{RT}$.*

For a fixed modular category, we call the integer N the category's *Vafa's constant*.

2.4 Relevant data structure

An embedded graph $G = (V, E)$ in a surface Σ defines a *cellular complex* for Σ if $\Sigma - G$ is a union of open disks, which we call *faces*. Note that this implies that any boundary component of Σ fully lies within some set of edges in E. The *dual graph* of G is another graph embedded in Σ defined by assigning a vertex to each face of $\Sigma - G$ and an edge between the vertices

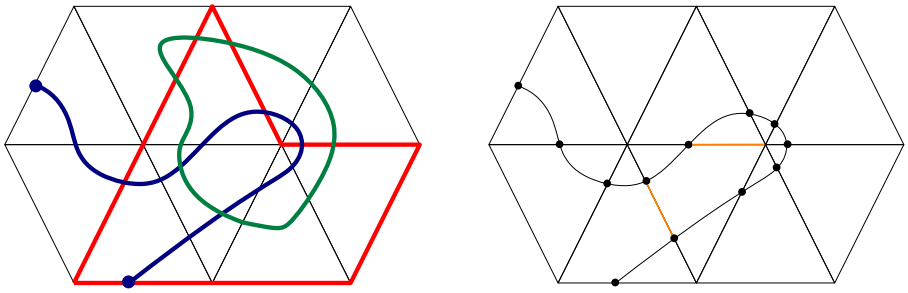


Figure 3 A triangulation of the disk with a curve represented by edge list (red), a standard curve (green), and a normal arc (blue).

if and only if the corresponding faces are separated by an edge in E. As a data structure, we represent a cellular embedding by the lists of faces, their incident edges, and vertices, allowing us to reconstruct Σ by gluing the appropriate pieces. Using this structure, one can compute the dual graph of the cellular embedding in linear time on the number of faces.

A *(generalized) triangulation* $T = (V, E)$ *of a surface* Σ is a cellular complex where all faces are bounded by exactly 3 edges. We denote the number of triangles in a triangulation by $|T|$; note that $|T| = O(|E|)$. We will most often consider *oriented triangulations* by giving an orientation to each triangle consistent with the orientation of the surface; we orient a triangle by imposing an order to its vertices as in the right-hand rule. We say that a triangulation T' of a surface Σ is a *subtriangulation* of another triangulation T of Σ, denoting $T \leq T'$, if the graph of T is embedded on T' (i.e., each vertex of T is a vertex of T' and each edge of T is a union of edges in T'). 'If T and T' are oriented, we also require the induced order of the vertices to be the same. Every face of T' is naturally contained in a face of T.

Using a triangulation $T = (V, E)$ of a surface Σ, we can describe an arc, curve, or multicurve s in Σ lying fully within the edges E by the list of the edges in $E \cap s$ (the red curve in Figure 3). We will call this list an *edge list representation* of the curve s, denote it by $E_T(s)$, and say that the number of edges in $E \cap s$ is the *edge complexity*, $\|E_T(s)\|$. We can always assume that $\|E_T(s)\| \leq |T|$.

For any fixed triangulation T of the surface Σ, there are always, however, isotopy classes of curves in Σ not representable as a subset of edges of T. To deal with this hindrance, we say, for a fixed oriented triangulation $T = (V, E)$ of the surface Σ, that a curve s is *standard* (to T) if it intersects T only transversely and at edges (green curve of Figure 3). If s is standard, we may represent it as an *intersection word* $I_T(s)$, taking E as an alphabet and traversing s along some arbitrary direction and, whenever we meet an edge $e \in E$, we append e to $I_T(s)$ if e is crossed according to the orientation of Σ and e^{-1} otherwise. While the edge representation is well-defined at the curve level, intersection words are defined only up to isotopies inside the triangulation's faces. Furthermore, isotopy classes of $I_T(s)$ are closed under cyclic permutations and taking inverses. The *complexity of an intersection word* $I_T(s)$, $\|I_T(s)\|$, is its length (i.e., the number of edges of T intersected by s, counted with multiplicity). If γ is a multicurve, we let $I_T(\gamma)$ be the set of $\#\gamma$ intersection words representing each component. Similarly, the complexity of a Heegaard diagram $(\Sigma_g, \alpha, \beta)$, where α and β are standard multicurves, equals $\|I_T(\alpha)\| + \|I_T(\beta)\|$. Standard arcs are treated accordingly.

A standard curve, multicurve or arc s is *normal* if no intersection word $I_T(s)$ contains a substring of form ee^{-1} or $e^{-1}e$ where $e \in E$ (the blue curve in Figure 3). When normal, the intersections of s and the faces of the triangulation are arcs connecting distinct edges

of each triangle, which we call *fundamental arcs*. Normal curves are more convenient for tracing than their more general standard counterparts (refer to Proposition 14); moreover, any standard curve s can be made normal in time $O(\|I_T(s)\|)$, see the proof of Theorem 15.

▶ Remark 12. When a curve is normal for a triangulation, its isotopy class is fixed by the number of times it intersects each labeled edge of E [46], meaning that it can be described through a vector in $\mathbb{N}^{|E|}$ called the curve's *normal coordinates* [8, 9, 46, 47], but whose introduction is unnecessary to the results of Section 4.

3 Algorithms for curves in surfaces and handlebodies

3.1 Converting between representations of curves

In Section 2.4, we saw two representations of curves in a surface: edge lists and intersection words. While some topological operations such as cutting along a curve are easier to implement using edge list representations (one needs only to delete the edges crossing $E_T(s)$ from the dual graph of T to cut along s), others, such as doing Dehn twists (Theorem 15) are more suited to curves intersecting the triangulation transversely. Therefore, it will be convenient to have procedures to convert from one representation to the other.

First, we describe an algorithm to transform curves represented by edge lists into intersection words. For an edge list $E_T(s)$ not in a boundary of an oriented surface Σ, there are two choices of normal curves isotopic to s created by slightly displacing it either to the left or to the right of the edges (with respect to the orientation of Σ and some arbitrary orientation of s); we call them *twins born from* s. Given a choice of twin for s, say left, one can compute its intersection word by traversing s and appending the letters representing adjacent edges coming from the left side of the triangulation graph when embedded in Σ. If, however, s lies either partially or fully on the boundary of Σ, we can only consistently displace it to one side, which can be determined in time $O(\|E_T(s)\|)$. This algorithm does not change the triangulation. For convenience, we state this argument as the following proposition.

▶ **Proposition 13.** *Suppose $E_T(s)$ is an edge list representation of an arc, curve, or multicurve s in Σ. Then there is an algorithm to compute an intersection word $I_T(s)$ of complexity $O(|T|)$ in time $O(|T|)$.*

The next result, whose proof is in the paper's full version, describes an algorithm for the other direction, that is, for going from an intersection word to an edge list representation.

▶ **Proposition 14.** *Suppose that s is a normal arc, curve, or multicurve to some triangulation T, with $\|I_T(s)\| = m$. Then, there exists an algorithm that constructs, in time $O(m + |T|)$, a new triangulation $T' \geq T$ of size $O(m + |T|)$ with s in the edges of T'.*

3.2 Basic operations on curves, handlebodies, and Heegaard splittings

Before proceeding with the main results, we will establish algorithms for some basic operations. The first of these is only a minor modification of [47], whereas the second, in the context of curves in surfaces, is mostly due to [55, Theorem 7.1]. The proofs of both theorems are in the paper's full version.

▶ **Theorem 15.** *Suppose t and s are normal (multi)curves for some fixed triangulation T of Σ_g, given through intersection words $I_T(s)$ and $I_T(t)$ with $m = \max\{\|I_T(s)\|, \|I_T(t)\|\}$. Then an intersection word for $\tau_s^k(t)$ for all $k \in \mathbb{Z}$, with $\|I_T(\tau_s^k(t))\| \leq |k|m^3$, can be computed in time $O(|k|m^3)$.*

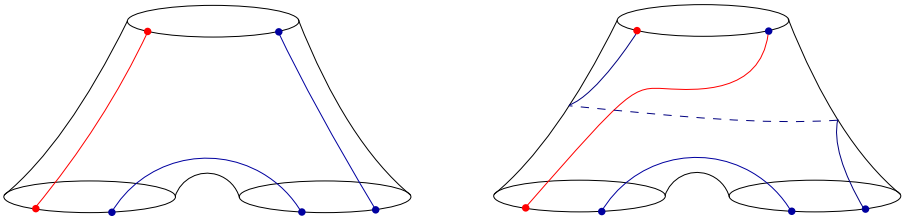


Figure 4 Left: the three types of seams, up to isotopy, on a pair of pants. Two connected components (a red and a blue one) of the multicurve σ are highlighted. Right: a surgery to connect the two distinct components.

▶ **Theorem 16.** *Suppose γ is a minimal system of edge curves in Σ_g. Then one can compute, in time $O(g|T|)$, a triangulation $T' \geq T$ of size $O(m|T|)$ and a pants decomposition ρ in the edges of T', with $\|E_{T'}(\rho)\| = O(g|T|)$, such that ρ contains γ.*

▶ **Theorem 17.** *Suppose γ is a normal minimal system in Σ_g with respect to a triangulation T, with $\|I_T(\gamma)\| = m$. Then there is an algorithm that outputs, in time $O((m + |T|)g^2)$, a disksbuting curve s for the disk graph K_γ, which is normal to T and is represented by intersection word of complexity $O(g|T|)$.*

Proof. Use Proposition 14 to get an edge list representation of γ in a new triangulation T' of complexity $O(m + |T|)$ and then Theorem 16 for an equivalent pants decomposition $\rho \supseteq \gamma$ of complexity $O(g|T'|)$ as an edge list in another triangulation T'' of Σ_g. We will construct a multicurve σ, seamed for the pants decomposition ρ, intersecting each component twice.

For each component r of ρ, select two edges contained in $E_T(r)$ to be points of intersection with σ. For every connected component P of $\Sigma_g - \rho$ (i.e., P is a pair of pants), we connect each intersection point of a boundary component of P to other intersection points in the two other components of ∂P. To avoid self-intersections of σ, we separately draw in each pair of pants a seam at a time, using breadth-first search in the dual graph, recording the intersection words, and changing the dual graph so that no seam can cross a previously traced seam (see the proof of Theorem 16). Even though we change the dual graph, making it more complicated at each drawing of a seam, because the seams are local within pairs of pants, each application of breadth-first search considers only $O(|T''|)$ nodes and edges. At the end of the process, we have a multicurve σ that, although seamed for the pants decomposition ρ, may have up to $g + 1$ connected components. It therefore remains to modify σ so it has just a single component.

Whenever there are still disconnected components in σ, there exists a pair of pants intersected by at least two distinct components of σ, refer to Figure 4. We can then do the surgery on the right side of Figure 4 to connect the two components. Using breadth-first, this takes a total time of $O(g|T''|)$ and yields a connected curve s of intersection word $I_{T''}(s)$. Because, by construction, s is seamed for each pair of pants from $\Sigma_g - \rho$, by Theorem 6, s is diskbusting for γ. Finally, we compute an intersection word of s with respect to T in time $O(|T''|)$ by deleting the edges in $T''\backslash T$. Note that s is already normal for T as, by construction, s is standard to $T'' \geq T' \geq T$ and, since all paths are shortest, no cyclic reductions are possible. ◀

The proof of the following (slightly technical) result is also carried in the paper's extended version.

▶ **Lemma 18.** *Suppose $(\Sigma_g, \alpha, \beta)$ is a Heegaard diagram, with α and β given as intersection words in a triangulation T of Σ_g, where we $m = \max\{\|I_T(\alpha)\|, \|I_T(\beta)\|\}$. One can find, in time $O(m)$, a triangulation T' and new multicurves α', β' which represent a stabilization $(\Sigma_{g+1}, \alpha', \beta')$ of the original diagram, with $O(|T'|) = O(|T|)$ and $\|I_{T'}(\alpha')\|, \|I_{T'}(\beta')\| = O(m)$.*

4 Restricting the topology of 3-manifolds

Given a Heegaard diagram $(\Sigma_g, \alpha, \beta)$ and a fixed $k \in \mathbb{Z}^+$, the algorithm of Theorem 1 uses Theorems 9 and 11 to construct a new diagram $(\Sigma_{g+1}, \alpha, \tau_s^n(\beta))$ with Hempel distance at least k, without altering the associated RT-invariant in the process, where n is some integer multiple of the associated Vafa's constant and s is a curve distant enough from both K_α and K_β. The main challenge of the algorithm comes, however, from building such a curve s. We address the problem by first computing, through Proposition 20, two curves, say s_α and s_β, of distance at least k from K_α and K_β, respectively. We then use these two curves in Proposition 22 to construct a single curve s, whose distances to K_α and K_β satisfy the hypotheses of Theorem 9.

We establish Proposition 20 incrementally, first using Theorem 17 to find a curve s diskbusting to K_γ, increasing $d(K_\gamma, s)$ to at least 3 using Lemma 19, and finally making the distance bigger than k.

▶ **Lemma 19.** *Let γ be a minimal system in Σ_g, normal to a triangulation T of Σ_g, and given by an intersection word $I_T(\gamma)$ with $\|I_T(\gamma)\| = m$. One can compute, in time $O((gm|T|)^9)$, an intersection word of a normal curve s', of complexity $O((gm|T|)^9)$, with $d(K_\gamma, s) \geq 3$.*

This result is simply an application of Theorem 17 followed by Theorem 10; the details are carried out in the extended version.

▶ **Proposition 20.** *Let γ be a minimal system in Σ_g, normal with respect to some triangulation T. Then, for a fixed $3 \leq k \in \mathbb{Z}^+$, one can compute, in time $c_2^{O(k^{c_1})} \log k$, where $c_1 = \log 3$ and $c_2 = O(gm|T|k)$, the intersection word of a curve s, normal to T and of complexity $c_1^{O(k^{c_2})}$, for which $d(K_\gamma, s) > k$.*

Proof. Start using Lemma 19 to find an intersection word of a curve s_0 with $d(K_\gamma, s_0) \geq 3$. If $k = 3$ we are done, so assume otherwise. Define $M = \|I_T(s_0)\|$; recall that $M = O((gm|T|)^9)$. Let c be any connected component of γ and recursively define $s_{i+1} = \tau_{s_i}^{k_i+3}(c)$, where $k_i = 2^{i+1} + 2$. We claim that, for any $\ell \in \mathbb{Z}^+$, $d(K_\gamma, s_\ell) \geq k_{\ell-1}$. We prove this inducting on ℓ: for $\ell = 1$, $k_0 = 4$ and, by Theorem 9 with $\alpha = \beta = \gamma$ and $n = \min\{d(K_\gamma, K_\gamma), 1\} = 1$, we have that

$$d(K_\gamma, s_1) = d(K_\gamma, \tau_{s_0}^{k_0+n+2}(c)) \geq d(K_\gamma, K_{\tau_{s_0}^{k_0+n+2}(\gamma)}) \geq 4.$$

Now assume $d(K_\gamma, s_\ell) \geq 2^\ell + 2$. Again, by Theorem 9,

$$d(K_\gamma, s_{\ell+1}) = d(K_\gamma, \tau_{s_\ell}^{k_\ell+3}(c)) \geq d(K_\gamma, K_{\tau_{s_\ell}^{k_\ell+3}(\gamma)}) \geq 2d(K_\gamma, s_\ell) - 2 \geq 2(2^\ell + 2) - 2 = 2^{\ell+1} + 2,$$

so, by induction, $d(K_\gamma, s_\ell) \geq k_{\ell-1}$. Setting the total number of iterations at $\ell = \lceil \log(k-2) \rceil$ and $s = s_\ell$, we have that $d(K_\gamma, s) \geq k$.

We now estimate the total computational time and the output's complexity of the algorithm. First, we note that, by Theorem 15, $\|I_T(s_{i+1})\| \leq (k_i + 3)\|I_T(s_i)\|^3$. Recursively, this gives

$$\begin{aligned}
\|I_T(s)\| &\leq M^{3^\ell}(k_0+3)^{3^{\ell-1}}(k_1+3)^{3^{\ell-2}}\dots(k_{\ell-1}+3)\\
&\leq M^{3^\ell}(2^1+5)^{3^{\ell-1}}(2^2+5)^{3^{\ell-2}}\dots(2^\ell+5)\\
&\leq M^{3^\ell}(2^\ell+5)^{3^{\ell-1}}(2^\ell+5)^{3^{\ell-2}}\dots(2^\ell+5)\\
&\leq M^{3^\ell}(2^\ell+5)^{3^{\ell-1}+3^{\ell-2}+\dots+3^0}\\
&\leq M^{3^\ell}(2^\ell+5)^{\frac{1}{2}\times(3^\ell-1)}\\
&\leq M^{3^{\log(k-2)+2}}(2^{\log(k-2)+2}+5)^{\frac{1}{2}\times(3^{\log(k-2)+2}-1)}\\
&\leq M^{9\times 3^{\log(k-2)}}(4k-3)^{9/2\times 3^{\log(k-2)}}\\
&\leq M^{9(k-2)^{\log 3}}(4k)^{5(k-2)^{\log 3}}\\
&\leq M^{9k^{c_1}}(4k)^{5k^{c_1}}
\end{aligned}$$

where we used the geometric series in the fifth line, $\lceil \log(k-2)\rceil \leq \log(k-2)+2$ in the sixth line, the relation $a^{\log b} = 2^{\log a \log b} = b^{\log a}$ for any real $a, b > 1$ in the eighth line, and $c_1 = \log 3$ in the last line. The time complexity for computing $I_T(s)$ can be (very coarsely) estimated at $O(\ell \times (k_{\ell-1}+3)\|I_T(s_{\ell-1})\|^3) = O(\log k \times \|I_T(s)\|) = O((gm|T|)^{81k^{c_1}}(4k)^{5k^{c_1}}\log k)$ by noting that $\|I_T(s_i)\| \leq \|I_T(s_{\ell-1})\|$ and $k_i \leq k_{\ell-1} = k$ for all $1 \leq i \leq n$. ◀

We now show Proposition 22, starting with the following technical lemma, whose proof (a simple application of the triangle inequality) is carried out in the paper's full version.

▶ **Lemma 21.** *Consider some full systems γ and γ' in the surface Σ_g with $d(K_\gamma, K_{\gamma'}) \geq 4$ and a curve s with $d(K_\gamma, s) < 2$. Then $d(K_{\gamma'}, s) \geq 2$.*

▶ **Proposition 22.** *Fix an integer $k \geq 4$. Consider some minimal systems α and β in the surface Σ_g for which $d(K_\alpha, K_\beta) = 0$. Then there is a curve s and some full minimal system β' (potentially $\beta' = \beta$) in Σ_g such that*

$$\min_{i=\alpha,\beta'}\{d(K_i,s)\} \geq 2,\ \max_{i=\alpha,\beta'}\{d(K_i,s)\} \geq k, d(K_\alpha,s)+d(K_{\beta'},s)-2 > n,$$
$$\text{and } \langle(\Sigma_g,\beta)\rangle_{\mathcal{C}}^{RT} = \langle(\Sigma_g,\beta')\rangle_{\mathcal{C}}^{RT}, \tag{2}$$

where $n = \min\{d(K_\alpha, K_{\beta'}), 1\}$. Moreover, if $m = \max\{\|I_T(\alpha)\|, \|I_T(\beta)\|\}$, then $I_T(s)$ and $I_T(\beta')$ will have complexity $Nkc_2^{O(k^{c_1})}$ and are computed in similar time for any choice of Vafa's constant $N \in \mathbb{Z}^+$, where $c_1 = \log 3$ and $c_2 = O(gm|T|k)$.

Proof. We use Proposition 20 to find two curves, s_α and s_β, such that $d(K_\alpha, s_\alpha)$ and $d(K_\beta, s_\beta)$ are larger than k. Note that if s_α is diskbusting in K_β, we are done, as we can let $s = s_\alpha$, $\beta' = \beta$, and

$$d(K_\alpha, s) \geq k,\ d(K_{\beta'}, s) = d(K_\beta, s_\alpha) \geq 2, \text{ and } d(K_\alpha, s) + d(K_{\beta'}, s) - 2 > 1.$$

Therefore, assume $d(K_\beta, s_\alpha) < 2$.

By applying Theorem 15, we compute an intersection word for $\widetilde{\beta} = \tau_{s_\beta}^{N(k+3)}(\beta)$. Note that $\|I_T(\widetilde{\beta})\| = O(Nk\|I_T(s_\beta)\|^3) = O(Nk(gm|T|)^{243k^{c_1}}(4k)^{15k^{c_1}})$ and, by Theorem 9, $d(K_\beta, K_{\beta'}) \geq k$. Moreover, because β' was constructed by applying a power of Dehn twists multiple of N to β, by Theorem 11, $\langle(\Sigma_g,\beta)\rangle_{\mathcal{C}}^{RT} = \langle(\Sigma_g,\beta')\rangle_{\mathcal{C}}^{RT}$.

Pick any component $\widetilde{b}$ of $\widetilde{\beta}$. If $\widetilde{b}$ is diskbusting for K_α, then $s = \widetilde{b}$ and $\beta' = \beta$ have the desired properties. In particular, notice that

$$d(K_\alpha, s) \geq 2,\ \ d(K_{\beta'}, s) \geq d(K_\beta, K_{\widetilde{\beta}}) \geq k, \text{ and } d(K_\alpha, s) + d(K_{\beta'}, s) - 2 > 1.$$

If, however, $\widetilde{b}$ is not diskbusting for K_α, it means that $d(K_\alpha, K_{\widetilde{\beta}}) \leq d(K_\alpha, \widetilde{b}) < 2$. Letting $s = s_\alpha$ and $\beta' = \widetilde{\beta}$ gives

$$d(K_\alpha, s) \geq k, \;\; d(K_{\beta'}, s) \geq 2, \text{ and } d(K_\alpha, s) + d(K_{\beta'}, s) - 2 \geq k \geq 3 > d(K_\alpha, K_{\beta'}),$$

where the second inequality comes from $d(K_\beta, s_\alpha) < 2$ and $d(K_\beta, K_{\widetilde{\beta}}) \geq k \geq 4$ applied to Lemma 21. ◀

We can, once again, combine the above result with Theorem 9 to prove our main reduction.

▶ **Theorem 1.** *Let $(\Sigma_g, \alpha, \beta)$ be a Heegaard diagram of a closed 3-manifold M of complexity m to a triangulation T of Σ_g. Choose a modular category $\mathcal{C}$ with Vafa's constant N and fix an integer $k \geq 4$. Then there is a set of three Heegaard diagrams, computed in time $O_k(\text{poly}(g, m, |T|, N))$ (of degree depending on k), representing manifolds with RT invariant over $\mathcal{C}$ equal to $\langle M \rangle_{\mathcal{C}}^{RT}$, one of them guaranteed to be hyperbolic and with no embedded incompressible orientable surface of genus at most $2k$.*

Proof. We start by using Lemma 18 to compute a stabilized splitting $(\Sigma_{g+1}, \alpha', \beta')$; note that $d(K_{\alpha'}, K_{\beta'}) = 0$. Construct two curves s_α and s_β as in the proof of Proposition 22. Let $\beta_1 = \beta'$ and $s_1 = s_\alpha$.

We proceed as in the proof of Proposition 22, computing an intersection word of a new minimal system $\widetilde{\beta} = \tau_{s_\beta}^{N(k+3)}(\beta')$ and defining $\beta_2 = \beta'$ and $\beta_3 = \widetilde{\beta}$, s_2 as any component of $\widetilde{\beta}$, and $s_3 = s_\alpha$. By Proposition 20, for at least one $1 \leq j \leq 3$

$$\min_{i=\alpha',\beta^j} \{d(K_i, s_j)\} \geq 2, \max_{i=\alpha',\beta^j} \{d(K_i, s_j)\} \geq k, d(K_{\alpha'}, s_j) + d(K_{\beta_j}, s_j) - 2 > n$$

where $n = \max\{d(K_{\alpha'}, K_{\beta_j}), 1\}$. Applying, once again, Theorems 15 and 9, we conclude that one of the splittings $(\Sigma_g, \alpha', \tau_{s_j}^{N(k+3)}(\beta_j))$ has a Hempel distance of, at least, k and an intersection word for $\tau_{s_j}^{N(k+3)}(\beta_j)$ can be found in time $O(Nk(\max\{\|I_T(\beta_j)\|, \|I_T(s_j)\|\})^3)$ $= O(N^4 k^4 (gm|T|)^{729k^{c_1}} (4k)^{45k^{c_1}})$. Theorem 7 finishes the proof. ◀

5 Computational reduction for quantum invariants

Theorem 1 gives a polynomial time algorithm to change a general closed 3-manifold into another manifold with very restricted topology without altering the RT invariant in the process. Therefore, the problems of either exactly computing or approximating the invariant of general 3-manifolds *reduce*, in a Cook-Turing sense, to the problems of exactly computing or approximating the invariant when the manifolds are assumed to have the properties of Theorem 1. Ultimately, this means that the *hardness of computation* is not altered in this restricted topology scenario.

We illustrate this reduction by showing that *value-distinguishing* approximations of the Reshetikhin–Turaev and the Turaev-Viro invariants are #P-hard, even for manifolds with the properties of Theorem 1, when we take $\mathcal{C}$ to be the category of representations of the quantum group $SO_r(3)$, for some prime $r \geq 5$ (in this case, one can use $N = 4r$ for Vafa's constant). We note that, by a value-distinguishing approximation, we mean the ability to determine whether the approximated quantity $c \in \mathbb{R}^+$ is $a > c$ or $b < c$ for any fixed $0 < a < b$ where we assume, as a premise, one of the two to hold. In particular, multiplicative approximations are value-distinguishing, although other less restrictive schemes also are [30].

5.1 Reshetikhin–Turaev invariant

For every Heegaard diagram $(\Sigma_g, \alpha, \beta)$, there is an orientation-preserving homeomorphism $\phi : \Sigma_g \to \Sigma_g$ such that $\beta = \phi(\alpha)$ [18, 56]. A lthough not unique, this map is well-defined on $\text{Mod}(\Sigma_g)$, so it can be described by a word on Lickorish generators. We do not differentiate between the notation of the homomorphism ϕ from its equivalence class in $\text{Mod}(\Sigma_g)$.

▶ **Theorem 23** ([3, 20]). *Consider the problem $\mathcal{P}$ of determining a value-distinguishing approximation of the $SO_r(3)$-RT invariant, $r \geq 5$ prime, of a manifold M, represented through a Heegaard splitting described by a word $\phi \in \text{Mod}(\Sigma_g)$ for some known $g \geq 2$. Then $\mathcal{P}$ is #P-hard in the sense of a Cook-Turing reduction.*

Before applying Theorem 1 to this result, we need to find an algorithm to convert the pair (Σ_g, ϕ) into a proper Heegaard diagram (Σ_g, β). This cannot be done through brute force computing $\beta = \phi(\alpha)$, since, as we saw in the proof of Proposition 20, it can lead to exponential bottlenecks. We fix the problem with the following lemma at the cost of potentially increasing the value of g. The proof of the next, as well as all other results of this section (except for Corollary 25), is found in the paper's full version.

▶ **Lemma 24.** *Consider a Heegaard splitting described by a word $\phi \in \text{Mod}(\Sigma_g)$ for some known $g \geq 2$. Then it is possible to compute, in time $O(\text{poly}(g, \phi))$, a Heegaard diagram $(\Sigma_{g'}, \beta)$ representing the same manifold, with β normal to a triangulation T of $\Sigma_{g'}$.*

▶ **Corollary 25.** *Fix a prime $r \geq 5$. Consider the problem $\mathcal{P}$ of, given a Heegaard diagram (Σ_g, β) of a closed 3-manifold M, returning a value-distinguishing approximation of its $SO_r(3)$-RT invariant if M has the properties of Theorem 1, otherwise remaining silent. Then $\mathcal{P}$ is #P-hard in the sense of a Cook-Turing reduction.*

Proof. Let $\mathcal{O}$ be an oracle machine that solves $\mathcal{P}$ and consider the problem $\mathcal{P}'$ of finding a value-distinguishing approximation of a general Heegaard splitting given as a pair (Σ_g, ϕ) for $\phi \in \text{Mod}(\Sigma_g)$. We will show that $\mathcal{O}$ solves P' with only a polynomial overhead. Because $\mathcal{O}$ solves $\mathcal{P}$ and, by Theorem 23, $\mathcal{P}'$ is #P-hard, then so is $\mathcal{P}$.

Let (Σ_g, ϕ) encode a Heegaard splitting of a manifold M, not necessarily with the properties of Theorem 1. Using Lemma 24, we transform (Σ_g, ϕ) into a Heegaard diagram $(\Sigma_{g'}, \beta)$ in polynomial time. Then apply the algorithm of Theorem 1 to this diagram, returning three new diagrams as output. We run the oracle $\mathcal{O}$ in parallel to these three diagrams, stopping the program whenever it halts for one of them. In the end, this gives, in polynomial time, a value-distinguishing approximation of $\langle M \rangle^{RT}_{SO_r(3)}$, concluding the proof. ◀

▶ Remark 26. We note that the sort of reduction assumed by the statement of Corollary 25 is related to what is often called in the complexity literature by a *semi-decision problem*, that is, an oracle that cannot return an incorrect answer, but may not halt for some inputs. Although weaker than the more common approach in which we assume that $\mathcal{P}$ *can* give incorrect approximation of the invariant if the input does not have the expected properties, this sort of oracle has also already been discussed for algorithm on 3-manifolds, e.g. see the Definition 1.3 in [35].

5.2 Turaev-Viro invariant

A compact 3-manifold can also be combinatorially described by a set of tetrahedra $\mathcal{T}$, together with rules on how to glue their triangular faces [10, 40]. This description is called a *triangulation of the 3-manifold*, but it should not be confused with triangulations of surfaces. Nonetheless, if M has a boundary, $\mathcal{T}$ naturally defines a (surface) triangulation for ∂M. For each $g \geq 1$, there exists a one-vertex triangulation of the handlebody of genus g [27].

The Turaev-Viro invariant (TV) is another quantum invariant for closed 3-manifolds. It is defined for spherical categories (which include modular categories) and is computed directly from a triangulation [7]. The Turaev-Walker theorem [53] states that, given a manifold M, $|\langle M\rangle_{\mathcal{C}}^{RT}|^2 = \langle M\rangle_{\mathcal{C}}^{TV}$, provided $\mathcal{C}$ is a modular category. We show that an approximation of the $SO_r(3)$-TV invariant can be used to compute an approximation of $SO_r(3)$-RT [3]. For such, we use the next theorem due to [21].

▶ **Theorem 27.** *Suppose β is a normal minimal system with respect to a one-vertex triangulation of the standard embedding of the genus g handlebody in $\mathbb{R}^3$, with $\|I_T(\beta)\| = m$. There is an algorithm to compute, in time* $\mathrm{poly}(m, g)$*, a triangulation of the 3-manifold of Heegaard diagram (Σ_g, β).*

▶ **Corollary 28.** *Fix a prime $r \geq 5$. Consider the problem $\mathcal{P}$ of, given a triangulation of a closed 3-manifold M, returning a value-distinguishing approximation of its $SO_r(3)$-TV invariant if M has the properties of Theorem 1, otherwise remaining silent. Then $\mathcal{P}$ is #P-hard in the sense of Cook-Turing reduction.*

References

1 Ian Agol, Joel Hass, and William Thurston. The computational complexity of knot genus and spanning area. *Transactions of the American Mathematical Society*, 358(9):3821–3850, 2006. arXiv:math/0205057.

2 D. Aharonov, V. Jones, and Z Landau. A polynomial quantum algorithm for approximating the Jones polynomial. *Algorithmica*, 55:395–421, 2009. doi:10.1007/s00453-008-9168-0.

3 Gorjan Alagic and Catharine Lo. Quantum invariants of 3-manifolds and NP vs #P. *Quantum Info. Comput.*, 17(1–2):125–146, February 2017. URL: https://arxiv.org/abs/1411.6049.

4 Itai Arad and Zeph Landau. Quantum computation and the evaluation of tensor networks. *SIAM Journal on Computing*, 39(7):3089–3121, 2010. doi:10.1137/080739379.

5 Rodrigo Arouca, Andrea Cappelli, and Hans Hansson. Quantum field theory anomalies in condensed matter physics. *SciPost Physics Lecture Notes*, September 2022. doi:10.21468/scipostphyslectnotes.62.

6 David Bachman, Ryan Derby-Talbot, and Eric Sedgwick. Computing Heegaard genus is NP-hard. *A Journey Through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 59–87, 2017. arXiv:1606.01553.

7 John Barrett and Bruce Westbury. Invariants of piecewise-linear 3-manifolds. *Transactions of the American Mathematical Society*, 348(10):3997–4022, 1996. URL: https://arxiv.org/abs/hep-th/9311155.

8 Mark C. Bell and Richard C. H. Webb. Polynomial-time algorithms for the curve graph. *arXiv: Geometric Topology*, 2016. URL: https://arxiv.org/abs/1609.09392.

9 Mark Christopher Bell. *Recognising mapping classes.* PhD thesis, University of Warwick, 2015. URL: https://wrap.warwick.ac.uk/id/eprint/77123/.

10 RH Bing. An alternative proof that 3-manifolds can be triangulated. *Annals of Mathematics*, 69(1):37–65, 1959. doi:10.2307/1970092.

11 Benjamin A. Burton. The HOMFLY-PT Polynomial is Fixed-Parameter Tractable. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2018.18.

12 Benjamin A. Burton. The Next 350 Million Knots. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2020.25.

13 Benjamin A. Burton, Clément Maria, and Jonathan Spreer. Algorithms and complexity for Turaev-Viro invariants. *Journal of Applied and Computational Topology*, 2(1-2):33–53, 2018. doi:10.1007/s41468-018-0016-2.

14 Qingtao Chen and Tian Yang. Volume conjectures for the Reshetikhin–Turaev and the Turaev–Viro invariants. *Quantum Topology*, 3:419–460, 2018. arXiv:1503.02547.

15 Colleen Delaney, Clément Maria, and Eric Samperton. An algorithm for Tambara-Yamagami quantum invariants of 3-manifolds, parameterized by the first Betti number, 2024. arXiv:2311.08514.

16 Pavel Etingof. On Vafa's theorem for tensor categories. *Mathematical Research Letters*, 9:651–657, 2002. arXiv:0207007.

17 Tatiana Evans. High distance Heegaard splittings of 3-manifolds. *Topology and its Applications*, 153(14):2631–2647, 2006. doi:10.1016/j.topol.2005.11.003.

18 Benson Farb and Dan Margalit. *A primer on mapping class groups*, volume 41. Princeton University Press, 2011.

19 Eduardo Fradkin. *Field theoretic aspects of condensed matter physics: An overview*, pages 27–131. Elsevier, 2024. doi:10.1016/b978-0-323-90800-9.00269-9.

20 M. Freedman, M. Larsen, and Z. Wang. The two-eigenvalue problem and density of Jones representation of braid groups. *Communications in Mathematical Physics*, 228:177–199, 2002. arXiv:0103200.

21 Alexander He, James Morgan, and Em K Thompson. An algorithm to construct one-vertex triangulations of Heegaard splittings. *arXiv preprint*, 2023. arXiv:2312.17556.

22 John Hempel. 3-manifolds as viewed from the curve complex. *Topology*, 40(3):631–657, 2001. arXiv:math/9712220.

23 Kristóf Huszár and Jonathan Spreer. 3-Manifold Triangulations with Small Treewidth. In Gill Barequet and Yusu Wang, editors, *35th International Symposium on Computational Geometry (SoCG 2019)*, volume 129 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2019.44.

24 Kristóf Huszár and Jonathan Spreer. On the Width of Complicated JSJ Decompositions. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry (SoCG 2023)*, volume 258 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2023.42.

25 Kristóf Huszár. On the pathwidth of hyperbolic 3-manifolds. *Computing in Geometry and Topology*, 1(1):1:1–1:19, February 2022. doi:10.57717/cgt.v1i1.4.

26 Ayako Ido, Yeonhee Jang, and Tsuyoshi Kobayashi. Heegaard splittings of distance exactly n. *Algebraic & geometric topology*, 14(3):1395–1411, 2014. arXiv:1210.7627.

27 William Jaco and J. Hyam Rubinstein. Layered-triangulations of 3-manifolds, 2006. arXiv:math/0603601.

28 Jesse Johnson. Non-uniqueness of high distance Heegaard splittings. *arXiv preprint*, 2013. arXiv:1308.4599.

29 A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003. doi:10.1016/S0003-4916(02)00018-0.

30 Greg Kuperberg. How hard is it to approximate the Jones Polynomial? *Theory Computing*, 11:183–219, 2009. arXiv:0908.0512.

31 Greg Kuperberg. Algorithmic homeomorphism of 3-manifolds as a corollary of geometrization. *Pacific Journal of Mathematics*, 301(1):189–241, 2019. arXiv:1508.06720.

32 William BR Lickorish. A finite set of generators for the homeotopy group of a 2-manifold. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 60, pages 769–778. Cambridge University Press, 1964. doi:10.1017/S030500410003824X.

33 Martin Lustig and Yoav Moriah. High distance Heegaard splittings via fat train tracks. *Topology and its Applications*, 156(6):1118–1129, 2009. arXiv:0706.0599.

34 J.A. Makowsky and J.P. Mariño. The parametrized complexity of knot polynomials. *Journal of Computer and System Sciences*, 67(4):742–756, 2003. Parameterized Computation and Complexity 2003. doi:10.1016/S0022-0000(03)00080-1.

35 Jason Fox Manning. Algorithmic detection and description of hyperbolic structures on closed 3–manifolds with solvable word problem. *Geometry & Topology*, 6(1):1–26, 2002. arXiv:math/0102154.

36 Clément Maria. Parameterized complexity of quantum knot invariants. In *Proceedings of the International Symposium on Computational Geometry, SoCG*, volume 189 of *LIPIcs*, pages 53:1–53:17, 2021. doi:10.4230/LIPIcs.SoCG.2021.53.

37 Clément Maria and Jessica S. Purcell. Treewidth, crushing and hyperbolic volume. *Algebraic & Geometric Topology*, 19:2625–2652, 2019. arXiv:1805.02357.

38 Clément Maria and Owen Rouillé. Computation of large asymptotics of 3-manifold quantum invariants. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX*, pages 193–206. SIAM, 2021. doi:10.1137/1.9781611976472.15.

39 Clément Maria and Jonathan Spreer. A polynomial-time algorithm to compute Turaev-Viro invariants tv(4,q) of 3-manifolds with bounded first Betti number. *Foundations of Computational Mathematics*, 20(5):1013–1034, 2020. arXiv:1607.02218.

40 Edwin E Moise. Affine structures in 3-manifolds: V. the triangulation theorem and hauptvermutung. *Annals of mathematics*, 56(1):96–114, 1952. doi:10.2307/1969769.

41 Lukas Müller and Lukas Woike. The Dehn twist action for quantum representations of mapping class groups. *arXiv preprint*, 2023. arXiv:2311.16020.

42 Ruifeng Qiu, Yanqing Zou, and Qilong Guo. The Heegaard distances cover all nonnegative integers. *Pacific Journal of Mathematics*, 275(1):231–255, 2015. arXiv:1302.5188.

43 Paolo Salvatore and Nathalie Wahl. Framed discs operads and the equivariant recognition principle, 2001. arXiv:math/0106242.

44 Eric Samperton. Topological quantum computation is hyperbolic. *Communications in Mathematical Physics*, 402:79–96, 2023. arXiv:2201.00857.

45 Nikolai Saveliev. *Lectures on the topology of 3-manifolds: an introduction to the Casson invariant*. Walter de Gruyter, 2011.

46 Marcus Schaefer, Eric Sedgwick, and Daniel Štefankovič. Algorithms for normal curves and surfaces. In *Computing and Combinatorics: 8th Annual International Conference, COCOON 2002 Singapore, August 15–17, 2002 Proceedings 8*, pages 370–380. Springer, 2002. doi:10.1007/3-540-45655-4_40.

47 Marcus Schaefer, Eric Sedgwick, and Daniel Stefankovic. Computing Dehn twists and geometric intersection numbers in polynomial time. In *CCCG*, volume 20, pages 111–114, 2008. URL: https://www.cs.rochester.edu/~stefanko/Publications/geometric.pdf.

48 Martin Scharlemann and Maggy Tomova. Alternate Heegaard genus bounds distance. *Geometry & Topology*, 10(1):593–617, 2006. arXiv:math/0501140.

49 Jennifer Schultens. *Introduction to 3-manifolds*, volume 151. American Mathematical Soc., 2014.

50 Joe Scull. The homeomorphism problem for hyperbolic manifolds I, 2021. arXiv:2108.00779.

51 Isadore Manuel Singer and John A Thorpe. *Lecture notes on elementary topology and geometry*. Springer, 2015.

52 Edith Nelson Starr. *Curves in handlebodies*. PhD thesis, University of California, Berkeley, 1992.

53 Vladimir G Turaev. *Quantum invariants of knots and 3-manifolds*. de Gruyter, 2010.

54 Cumrun Vafa. Toward classification of conformal theories. *Physics Letters B*, 206(3):421–426, 1988.

55 Éric Colin De Verdière and Francis Lazarus. Optimal pants decompositions and shortest homotopic cycles on an orientable surface. *Journal of the ACM (JACM)*, 54(4):18–es, 2007. doi:10.1007/978-3-540-24595-7_45.

56 Michael Yoshizawa. High distance Heegaard splittings via Dehn twists. *Algebraic & Geometric Topology*, 14(2):979–1004, 2014. arXiv:1212.1199.

Tight Guarantees for Cut-Relative Survivable Network Design via a Decomposition Technique

Nikhil Kumar
University of Waterloo, Canada

J. J. Nan
University of Waterloo, Canada

Chaitanya Swamy
University of Waterloo, Canada

Abstract

In the classical *survivable-network-design problem* (SNDP), we are given an undirected graph $G = (V, E)$, non-negative edge costs, and some k tuples (s_i, t_i, r_i), where $s_i, t_i \in V$ and $r_i \in \mathbb{Z}_+$. The objective is to find a minimum-cost subset $H \subseteq E$ such that each s_i-t_i pair remains connected even after the failure of any $r_i - 1$ edges. It is well-known that SNDP can be equivalently modeled using a weakly-supermodular *cut-requirement function* f, where the objective is to find the minimum-cost subset of edges that picks at least $f(S)$ edges across every cut $S \subseteq V$.

Recently, motivated by fault-tolerance in graph spanners, Dinitz, Koranteng, and Kortsartz proposed a variant of SNDP that enforces a *relative* level of fault tolerance with respect to G. Even if a feasible SNDP-solution may not exist due to G lacking the required fault-tolerance, the goal is to find a solution H that is at least as fault-tolerant as G itself. They formalize the latter condition in terms of paths and fault-sets, which gives rise to *path-relative* SNDP (which they call relative SNDP). Along these lines, we introduce a new model of relative network design, called *cut-relative* SNDP (CR-SNDP), where the goal is to select a minimum-cost subset of edges that satisfies the given (weakly-supermodular) cut-requirement function to the maximum extent possible, i.e., by picking $\min\{f(S), |\delta_G(S)|\}$ edges across every cut $S \subseteq V$.

Unlike SNDP, the cut-relative and path-relative versions of SNDP are not equivalent. The resulting cut-requirement function for CR-SNDP (as also path-relative SNDP) is not weakly supermodular, and extreme-point solutions to the natural LP-relaxation need not correspond to a laminar family of tight cut constraints. Consequently, standard techniques cannot be used directly to design approximation algorithms for this problem. We develop a *novel decomposition technique* to circumvent this difficulty and use it to give a *tight* 2*-approximation algorithm for* CR-SNDP. We also show some new hardness results for these relative-SNDP problems.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Mathematics of computing → Discrete optimization

Keywords and phrases Approximation algorithms, Network Design, Cut-requirement functions, Weak Supermodularity, Iterative rounding, LP rounding algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.38

Related Version *Full Version*: `https://arxiv.org/abs/2507.04473`

Funding Supported in part by NSERC grant 327620-09.

1 Introduction

In the classical *survivable-network-design problem* (SNDP), we are given an undirected graph $G = (V, E)$, non-negative edge costs $\{c_e\}_{e \in E}$, and some k source-sink pairs $s_i, t_i \in V$ with requirements $r_i \in \mathbb{Z}_+$, for $i = 1, \ldots, k$. The goal is to find a minimum-cost set $H \subseteq E$ of edges such that there are at least r_i edge-disjoint s_i-t_i paths in H, for all $i \in [k]$. We overload notation and use H to denote both the edge-set, and the corresponding subgraph of G.


© Nikhil Kumar, J. J. Nan, and Chaitanya Swamy;
licensed under Creative Commons License CC-BY 4.0



SNDP imposes an absolute level of fault-tolerance in a solution subgraph H, by ensuring that for any bounded-size *fault-set* $F \subseteq E$ of edges that may fail, the graph $H - F$ retains some connectivity properties. More precisely, due to Menger's theorem, or the max-flow min-cut theorem, the feasibility condition on H can be stated equivalently as follows. For every $i \in [k]$, and every fault-set $F \subseteq E$ with $|F| < r_i$:

(S1) (Path-version) there is an s_i-t_i path in $H - F$;

(S2) (Cut-version) for every s_i-t_i cut $S \subseteq V$ (i.e., $|S \cap \{s_i, t_i\}| = 1$), we have $\delta_{H-F}(S) \neq \emptyset$, where $\delta_{H-F}(S) := \delta(S) \cap (H - F)$.

Recently, motivated by work on fault-tolerance in graph spanners, Dinitz, Koranteng, and Kortsartz [8], proposed a variant of SNDP that aims to impose a *relative* level of fault-tolerance with respect to the graph G. The idea is that even if the SNDP instance in infeasible, because G itself does not possess the required level of fault-tolerance, one should not have to completely abandon the goal of fault-tolerance: it is still meaningful and reasonable to seek a solution that is "as fault-tolerant as G", and in this sense is fault-tolerant relative to G.

To formalize this, it is useful to consider the definition of SNDP in terms of fault-sets. Roughly speaking, we would like to capture that H is a feasible solution if

$$\text{for every fault-set } F \text{ (valid for SNDP)}, \quad H - F \text{ and } G - F \text{ have similar connectivity.} \quad (*)$$

To elaborate, in SNDP, if there is *even one* fault-set F' under which $G - F'$ fails to have the required connectivity, i.e., $|F'| < r_i$ and $F' \supseteq \delta_G(S)$ for some s_i-t_i cut S, then "all bets are off"; we declare that the instance in infeasible. But declaring infeasibility here feels unsatisfactory because we allow one specific "problematic" fault-set F' to block us from obtaining any kind of fault-tolerance. In contrast, (*) aims to provide a *per-fault-set guarantee*, by asking for a solution H that functions as well as G in terms of connectivity under the failure of any fault-set F (considered for SNDP). We can formalize "$H - F$ and $G - F$ have similar connectivity" in two ways, via paths or cuts, and this gives rise to the following two problem definitions.

▶ **Definition 1.1.** *Let* $\big(G = (V, E), \{c_e\}_{e \in E}, \{s_i, t_i, r_i\}_{i \in [k]}\big)$ *be an* SNDP *instance.*

(R1) Path-relative SNDP*: $H \subseteq E$ is a feasible solution if for every $i \in [k]$ and $F \subseteq E$ with $|F| < r_i$, $G - F$ has an s_i-t_i path $\implies$ $H - F$ has an s_i-t_i path.*

(R2) Cut-relative SNDP *(CR-SNDP): $H \subseteq E$ is a feasible solution if for every $i \in [k]$ and $F \subseteq E$ with $|F| < r_i$, for every s_i-t_i cut S, $\delta_{G-F}(S) \neq \emptyset \implies \delta_{H-F}(S) \neq \emptyset$.*

The goal in both problems is to find a minimum-cost feasible solution.

Perhaps surprisingly, path-relative SNDP and cut-relative SNDP are *not equivalent*, even when the SNDP instance involves a single s-t pair. It is not hard to show that if H is feasible for CR-SNDP, it is also feasible for path-relative SNDP[1] but the converse fails to hold; see Section 7 for an example.

Path-relative SNDP was defined by Dinitz et al. [8], who referred to this problem simply as relative SNDP. They considered (among other problems) the path-relative version of k-edge connected subgraph (k-ECSS), which is the special case of SNDP where we have an s_i-t_i pair with $r_i = k$ for every pair of nodes. (In this case, the path-relative and cut-relative versions do turn out to be equivalent [8].) They called the resulting path-relative problem, *k-edge*

[1] Fix any $i \in [k]$ and $F \subseteq E$ with $|F| < r_i$. If $G - F$ has an s_i-t_i path, then $\delta_{G-F}(S) \neq \emptyset$ for every s_i-t_i cut. So since H is feasible for cut-relative SNDP, we have $\delta_{H-F}(S) \neq \emptyset$ for every s_i-t_i cut, which implies that $H - F$ has an s_i-t_i path.

fault-tolerant subgraph (k-EFTS), and observed that the feasibility condition for k-EFTS can be equivalently stated as: H is feasible iff $|\delta_H(S)| \geq g(S) := \min\{k, |\delta_G(S)|\}$ for all $S \subseteq V$. The function $g : 2^V \mapsto \mathbb{Z}$ is called a *cut-requirement function*, as it stipulates the (minimum) number of edges across any cut in any feasible solution. Cut-requirement functions constitute a very versatile framework for specifying network-design problems, where given a cut-requirement function $f : 2^V \mapsto \mathbb{Z}$, the corresponding *$f$-network-design problem* (f-NDP), also called the *f-connectivity problem*, is to find a minimum-cost set $H \subseteq E$ such that $|\delta_H(S)| \geq f(S)$ for all $S \subseteq V$. For instance, it is easy to see that SNDP corresponds to f-NDP for the cut-requirement function f^{SNDP}, defined by $f^{\mathsf{SNDP}}(S) := \max\{r_i : |S \cap \{s_i, t_i\}| = 1\}$ (and k-ECSS is $f^{k\text{-ECSS}}$-NDP where $f^{k\text{-ECSS}}(S) := k$ for all $\emptyset \neq S \subsetneq V$); various other network-design problems, such as the T-join problem, point-to-point connection problem etc., can be modeled using suitable cut-requirement functions (see, e.g., [10, 11]).

Cut-relative SNDP is a problem that we introduce in this paper. Its formulation in terms of cuts is the problem one obtains when we replace $f^{k\text{-ECSS}}$ in the cut-based formulation of k-EFTS by the cut-requirement function f^{SNDP} for (general) SNDP: that is, (similar to k-EFTS), we can say that $H \subseteq E$ is feasible for CR-SNDP iff $|\delta_H(S)| \geq g^{\text{CR-SNDP}}(S) := \min\{f^{\mathsf{SNDP}}(S), |\delta_G(S)|\}$ for all $S \subseteq V$ (we show this easy equivalence in Section 2); in other words, CR-SNDP corresponds to $g^{\text{CR-SNDP}}$-NDP. To gain some intuition for cut-relative SNDP, and contrast it with path-relative SNDP, consider SNDP with a single s-t pair and requirement r, for simplicity. A feasible solution H to path-relative SNDP offers a per-fault-set guarantee as encapsulated by (R1). But, for a given fault-set F with $|F| < r$, we get a weak fault-tolerance guarantee in terms of s-t cuts: if $\delta_{G-F}(S) = \emptyset$ for *even one* s-t cut S then there is no requirement on H for this fault-set F. Cut-relative SNDP offers a per-fault-set *and per-cut guarantee*, since for (every F with $|F| < r$ and) every s-t cut S, we require that $\delta_{H-F}(S) \neq \emptyset$ if $\delta_{G-F}(S) \neq \emptyset$.

Along the lines of CR-SNDP, we can easily extend the cut-relative model to capture, more broadly, the cut-relative version of any network-design problem specified by a cut-requirement function: given a cut-requirement function f, in the corresponding *cut-relative f-network-design problem* (CR-f-NDP), we seek a min-cost $H \subseteq E$ such that $|\delta_H(S)| \geq g^{\text{CR-}f\text{-NDP}}(S) := \min\{f(S), |\delta_G(S)|\}$ for all $S \subseteq V$; that is, we seek to satisfy the cut-requirement function f to the maximum extent possible. We refer to f as the *base cut-requirement function*, to distinguish it from the cut-requirement function $g^{\text{CR-}f\text{-NDP}}$ that defines CR-f-NDP.

Modeling power. We arrived at CR-f-NDP, which can be seen as a fault-tolerant cut-covering problem, as a technically natural extension of k-EFTS. We show below that CR-f-NDP offers a surprising amount of modeling power, and, in particular, allows one to capture stronger forms of relative fault-tolerance compared to k-EFTS. In k-EFTS, there is a sharp relative-fault-tolerance threshold at k: if H is feasible, then $G - F$ and $H - F$ have the same components whenever $|F| < k$, but there are no guarantees for larger fault-sets. With CR-f-NDP, one can capture a weaker guarantee also for other fault-sets, which allows for a more *graceful degradation* of relative fault-tolerance as $|F|$ increases.

▶ **Example 1.2.** One way of modeling graceful relative-fault-tolerance degradation is as follows: given a *non-increasing function* $\tau : \mathbb{Z}_+ \mapsto \mathbb{Z}_+$, we seek a min-cost subgraph H such that for *every* fault-set F, $G - F$ and $H - F$ have exactly the same components with at most $\tau(|F|)$ nodes. If G models a communication network where connected nodes can communicate with each other, then this yields the desirable guarantee that, *post-faults*, if a node can communicate with more than $\tau(|F|)$ nodes in G then the same holds for H; if

this number is at most $\tau(|F|)$, then post-faults, it can communicate with the same nodes in G and H. Note that if $\tau(k-1) \geq n = |V|$, then H is feasible for k-EFTS; by adding more "levels" to τ, we can obtain fault-tolerance guarantees for larger fault-sets.

In Section 3, we show that this problem can be modeled as CR-f-NDP with the weakly-supermodular function $f = f^{\mathsf{grace}}$, where $f^{\mathsf{grace}}(S) := \min\{\ell : \tau(\ell) < |S|\}$, for $S \neq \emptyset$.

▶ **Example 1.3.** Generalizing Example 1.2 substantially, suppose along with τ, we have a *monotone* function $\pi : 2^V \mapsto \mathbb{R}_+$, i.e., $\pi(T) \leq \pi(S)$ if $T \subseteq S$. We now seek a subgraph H such that for every fault-set F and $S \subseteq V$ with $\pi(S) \leq \tau(|F|)$, we have that S is (the node-set of) a component of $G - F$ iff it is a component of $H - F$. Similar to Example 1.2, we can model this as CR-f-NDP by defining $f(S) := \min\{\ell : \tau(\ell) < \pi(S)\}$, for $S \neq \emptyset$ (Theorem 3.2).

This setup yields fault-tolerance degradation depending on monotone properties of components other than just their size, which creates a rich space of problems. For example, suppose $\pi(S) = \max_{u,v \in S} \min_{u\text{-}v \text{ paths } P \text{ in } G} |P|$ is the weak-diameter of S. Setting $\tau(\ell) = n$ if $\ell < k$, and some $t < n$ for larger ℓ, we seek a k-EFTS solution H such that for $F \subseteq E$ with $|F| \geq k$, $G - F$ and $H - F$ have the same components of weak-diameter at most t.

1.1 Our contributions

We introduce cut-relative network-design problems and develop strong approximation guarantees for these problems. *We obtain an approximation guarantee of* 2 *for cut-relative* SNDP, which, notably, matches the best-known approximation factor for SNDP. Our guarantee applies more generally to CR-f-NDP, whenever the base cut-requirement function f satisfies certain properties and the natural LP-relaxation of the problem ($\text{CRLP}_{f,G}$) can be solved efficiently. Our guarantee is relative to the optimal value of this LP, *and is tight in that it matches the integrality gap of this LP.*

We also show that even in the simplest SNDP setting with only one s-t pair (wherein SNDP is polytime solvable), cut-relative SNDP and path-relative SNDP capture k-ECSS as a special case, and are thus APX-hard (Section 6). Previously, even NP-hardness of path-relative SNDP in the s-t case (which is studied by [8, 9]) was not known.

Technical contributions and overview. Technically, our main contribution is to show that we can obtain such a strong guarantee *despite the fact that the cut-requirement function defining cut-relative network-design (i.e., $g^{\text{CR-}f\text{-NDP}}$) is not weakly supermodular*, which is the key property that drives the 2-approximation algorithm for SNDP. To elaborate, there is a celebrated 2-approximation algorithm for SNDP due to Jain [15] based on iterative rounding. This *crucially* exploits the fact that the underlying cut-requirement function f^{SNDP} is *weakly supermodular*: for any two node-sets $A, B \subseteq V$, we have:

$$f^{\mathsf{SNDP}}(A)+f^{\mathsf{SNDP}}(B) \leq \max\{f^{\mathsf{SNDP}}(A\cap B)+f^{\mathsf{SNDP}}(A\cup B),\ f^{\mathsf{SNDP}}(A-B)+f^{\mathsf{SNDP}}(B-A)\}.$$

This property allows one to argue that given an LP-solution x, any two *tight* sets that cross – i.e., sets A, B for which $x(S) = f^{\mathsf{SNDP}}(S)$ holds for $S \in \{A, B\}$ and $A \cap B, A - B, B - A$ are all non-empty – can be "uncrossed", and thereby an extreme-point solution $\widehat{x}$ to the LP can be defined via a *laminar family* of tight cut constraints. By a laminar family, we mean that any two sets A, B in the family satisfy $A \cap B = \emptyset$ or $A \subseteq B$, and uncrossing two crossing tight sets means that they can be replaced by an equivalent laminar family of tight sets. Jain's seminal contribution was to show that given such a laminar family defining an extreme point, there always exists an edge e for which $\widehat{x}_e \geq \frac{1}{2}$; picking such an edge and iterating then yields the 2-approximation. This technique of uncrossing tight cut-constraints to obtain a laminar

family of tight cut-constraints has proved to be quite versatile, becoming a staple technique that has been leveraged to obtain strong guarantees in various network-design settings such as, most notably, network-design problems with degree constraints [13, 17, 19, 5, 4, 16]. More generally, this technique of uncrossing to obtain a structured family of tight constraints has been utilized for various combinatorial-optimization problems; see, e.g., [18].

As mentioned earlier, the main complication with CR-f-NDP is that the underlying cut-requirement function $g^{\text{CR-}f\text{-NDP}}$ is usually *not* weakly-supermodular, even if the base cut-requirement function f is. In particular, the cut-requirement function $g^{\text{CR-SNDP}}$ defining cut-relative SNDP is not weakly-supermodular. This difficulty was also noted by Dinitz et al. [8] when they considered k-edge fault-tolerant subgraph (k-EFTS, or path- or cut-relative k-ECSS in our terminology). For this special case, they identify a notion called local weak-supermodularity that is satisfied by the cut-requirement function $\min\{k, |\delta_G(S)\}$ for k-EFTS, and observed that Jain's machinery can be used as is if this property holds; that is, one can uncross tight sets to obtain a laminar family, and hence obtain a 2-approximation for k-EFTS proceeding as in Jain's algorithm for SNDP.

However, for general CR-SNDP, such an approach breaks down, because, as we show in Section 2.1, one can construct quite simple CR-SNDP instances where an (optimal) extreme point *cannot be defined by any laminar family of tight cut constraints*. In particular, this implies that the cut-requirement function $g^{\text{CR-}f\text{-NDP}}$ is (in general) not locally weakly-supermodular (even when f is weakly-supermodular); moreover, it does not satisfy any property that allows for the uncrossing of tight sets. The upshot here is that the standard technique of uncrossing tight sets to obtain a laminar family does not work, and one needs new ideas to deal with cut-relative network design.

The chief technical novelty of our work is to show how to overcome the impediment posed by the lack of any such nice structural property for $g^{\text{CR-}f\text{-NDP}}$, via a novel *decomposition technique* that allows one to *reduce* cut-relative f-NDP with a weakly supermodular *base cut-requirement function* to (standard) f-NDP with (different) weakly supermodular cut-requirement functions (see Theorem 4.1).

We introduce a little notation to describe this. Let CR-(h, D)-NDP denote CR-f-NDP on graph D with base cut-requirement function h. For $S \subseteq V$, define its *deficiency* to be $f(S) - |\delta_G(S)|$; we say that S is a *small cut* if its deficiency is positive (i.e., $|\delta_G(S)| < f(S)$). We use $G[S]$ to denote the subgraph induced by S. Let $\overline{S}$ denote $V - S$.

Assume that the underlying base cut-requirement function f is weakly supermodular (as is the case with SNDP). Observe first that if there is no small cut then CR-f-NDP is the same as f-NDP. Our key technical insight is that, otherwise, we can utilize a novel splitting operation to simplify our problem. We show (see Theorem 4.5) that if we pick a small cut $A \subseteq V$ with *maximum deficiency*, then one can define suitable weakly-supermodular functions $f_A : 2^A \mapsto \mathbb{Z}$ and $f_{\overline{A}} : 2^{\overline{A}} \mapsto \mathbb{Z}$ such that *any solution to the original* CR-f-NDP*-instance yields solutions to* CR-$(f_A, G[A])$-NDP *and* CR-$(f_{\overline{A}}, G[\overline{A}])$-NDP*, and vice-versa.* The intuition here is as follows. Since A is a small cut, any feasible CR-f-NDP solution must include all edges in $\delta_G(A)$. The functions f_A and $f_{\overline{A}}$ are essentially the restrictions of f to A and $\overline{A}$ respectively, taking this into account. Consequently, moving to CR-$(f_A, G[A])$-NDP and CR-$(f_{\overline{A}}, G[\overline{A}])$-NDP does not really impact the constraints for cuts that do not cross A; but we do eliminate the constraints for crossing cuts. This is the simplification that we obtain via the splitting operation, and one needs to argue that this does not hurt feasibility; that is, the constraints for such cuts are still satisfied when we take $\delta_G(A)$ along with the CR-$(f_A, G[A])$-NDP and CR-$(f_{\overline{A}}, G[\overline{A}])$-NDP solutions. Here we exploit the weak-supermodularity of f, which implies that the deficiency

function $\mathrm{def}(S) := f(S) - |\delta_G(S)|$ is also weakly supermodular. One can use this, and the fact that A has maximum deficiency, to bound the deficiency of any crossing set T in terms of the deficiency of non-crossing sets, and hence show that the cut-constraint for T is satisfied.

By recursively applying the above splitting operation to A and $\overline{A}$, we end up with a partition $V_1, \ldots, V_k$ of V and weakly-supermodular functions $f_i : 2^{V_i} \mapsto \mathbb{Z}$ for all $i \in [k]$ such that, for each $i \in [k]$, the graph $G[V_i]$ contains no small cut with respect to f_i. It follows that solving f_i-NDP on the graph $G[V_i]$ for all $i \in [k]$ yields a solution to the original CR-f-NDP instance (Theorem 4.1).

While we can obtain this decomposition efficiently, assuming that one has suitable algorithmic primitives involving the base cut-requirement function f (shown in the full version), in fact we only need this decomposition in the *analysis*: in order to obtain the stated LP-relative 2-approximation algorithm, we only need to be able to find an extreme-point optimal solution to the natural LP-relaxation ($\mathrm{CRLP}_{f,G}$) for CR-f-NDP (also after fixing some variables to 1). This is because the translation between CR-f-NDP-solutions for G, and f_i-NDP-solutions on $G[V_i]$ for all $i \in [k]$, also applies to *fractional solutions*, and implies that any extreme-point solution to ($\mathrm{CRLP}_{f,G}$) yields extreme-point solutions to the LP-relaxations of f_i-NDP on $G[V_i]$ for all $i \in [k]$ (parts (b) and (c) of Theorem 4.1); hence, by Jain's result, we can always find some fractional edge of value at least $\frac{1}{2}$, round its value to 1, and iterate. For SNDP, we can find an extreme-point optimal solution efficiently because we argue that one can devise an efficient separation oracle for ($\mathrm{CRLP}_{f,G}$) (Lemma 2.3). This yields a 2-approximation algorithm for CR-SNDP, and more generally CR-f-NDP, whenever f is weakly supermodular and ($\mathrm{CRLP}_{f,G}$) can be solved efficiently (see Algorithm CRNDP-Alg in Section 5).

Other related work. Network design is a fairly broad research topic, with a vast amount of literature. We limit ourselves to the work that is most closely connected to our work.

As mentioned earlier, path-relative SNDP was introduced by Dinitz et al. [8]. In addition to k-EFTS, which is the path-relative version of k-ECSS, they study the path-relative versions of two other special cases of SNDP, for which they developed constant-factor approximation algorithms: (1) SNDP with $r_i \leq 2$ for all i; (2) one s_i-t_i pair with $r_i \leq 3$. Both results were subsequently extended by [9] who gave a 2-approximation for SNDP with $r_i \leq 3$ for all i, and a $2^{O(k^2)}$-approximation when there is one s_i-t_i pair with $r_i = k$. The only result known for general path-relative SNDP is due to [7], who (among other results) devise an $O(k^2 \log^2 n \log\log n)$-approximation algorithm with $n^{O(k)}$ running time. As noted earlier, on the hardness side, prior to our work, it was not known if the setting with one s_i-t_i pair is even *NP*-hard.

An influential paper by Goemans and Williamson [10] (see also the survey [11]), which built upon the work of [1] for the Steiner forest problem, developed the primal-dual method for network-design problems. Their work also popularized the use of cut-requirement functions to specify network-design problems, by showing how this framework can be used to capture a slew of problems, and how their primal-dual framework leads to a 2-approximation algorithm for $\{0,1\}$-cut-requirement functions satisfying some properties. This work was extended to handle general integer-valued cut-requirement functions, which captures SNDP, in [20, 12], which led to an $O(\log \max_i r_i)$-approximation for SNDP. The seminal work of Jain [15] later improved this to a 2-approximation, and in doing so, introduced the powerful technique of *iterative rounding*. Later work of [17, 19] on degree-bounded network design added another ingredient to iterative rounding, namely iteratively dropping some constraints. This paradigm of *iterative rounding and relaxation* has proved to be an extremely versatile and powerful tool in developing algorithms (and structural results) for network-design problems, and more generally in combinatorial optimization; see [18] for an extensive study.

Iterative rounding and relaxation derives its power from the fact that an extreme-point solution to an LP-relaxation of the problem can be defined using a structured family of tight constraints, such as, often a laminar family of cut constraints in the case of network-design problems. As noted earlier, this key property does not hold for cut-relative SNDP (and path-relative SNDP). Recently, various works have considered network-design problems that give rise to cut constraints that are less structured and not quite uncrossable; see [2, 3, 6] and the references therein. Our work can be seen as furthering this research direction.

2 Preliminaries and notation

Recall that we are given an undirected graph $G = (V, E)$ with nonnegative edge costs $\{c_e\}_{e \in E}$. For a subset $S \subseteq V$ and subset $Z \subseteq E$, which we will interchangeably view as the subgraph (V, Z) of G, we use $\delta_Z(S)$ to denote $\delta_G(S) \cap Z$, i.e., the edges of Z on the boundary of S. Recall that $\overline{S}$ denotes $V - S$, and $G[S]$ denotes the subgraph induced by S. A cut-requirement function on G is a function $f : 2^V \mapsto \mathbb{Z}$. (We allow $f(S)$ to be negative chiefly for notational simplicity; this does not affect anything, as we will only examine sets for which $f(S) \geq 0$.) Recall the following network-design problems.

- *f-network-design problem* (f-NDP): find a min-cost edge-set H such that $|\delta_H(S)| \geq f(S)$ for all $S \subseteq V$.
- *Cut-relative network-design problem* (CR-f-NDP): find a min-cost edge-set H such that $|\delta_H(S)| \geq g^{\text{CR-}f\text{-NDP}}(S) := \min\{f(S), |\delta_G(S)|\}$ for all $S \subseteq V$. This corresponds to $g^{\text{CR-}f\text{-NDP}}$-NDP. We call f the base cut-requirement function.
- *Survivable network design problem* (SNDP) is a special case of f-NDP. The input also specifies k tuples $\{(s_i, t_i, r_i)\}_{i \in [k]}$, where $s_i, t_i \in V$ and $r_i \in \mathbb{Z}_+$ for all $i \in [k]$. Defining $f^{\text{SNDP}}(S) := \max\{r_i : |S \cap \{s_i, t_i\}| = 1\}$ for all $S \subseteq V$, we obtain that SNDP is the f^{SNDP}-network-design problem.
- *Cut-relative* SNDP (CR-SNDP): The input here is the same as in SNDP. In Definition 1.1 (R2), CR-SNDP was defined as: find a min-cost $H \subseteq E$ such that for every $i \in [k]$, every $F \subseteq E$ with $|F| < r_i$, and every s_i-t_i cut S, we have $\delta_{H-F}(S) \neq \emptyset$ whenever $\delta_{G-F}(S) \neq \emptyset$. Later in Section 1, we stated that this can be equivalently formulated as CR-f-NDP with base cut-requirement function f^{SNDP}. We now justify this statement.
  Suppose that H is feasible under (R2) but $|\delta_H(S)| < \min\{f^{\text{SNDP}}(S), |\delta_G(S)|\}$ for some $S \subseteq V$. Suppose $f^{\text{SNDP}}(S) = r_i$. Then taking $F = \delta_H(S)$, we have $|F| < r_i$, $\delta_{G-F}(S) \neq \emptyset$, but $\delta_{H-F}(S) = \emptyset$, yielding a contradiction. So H is feasible for CR-f^{SNDP}-NDP. Conversely, suppose H is feasible for CR-f^{SNDP}-NDP. Consider any $i \in [k]$, any $F \subseteq E$ with $|F| < r_i$, and any s_i-t_i cut S with $\delta_{G-F}(S) \neq \emptyset$. Then, $|\delta_F(S)| < \min\{r_i, |\delta_G(S)|\} \leq \min\{f(S), |\delta_G(S)|\} \leq |\delta_H(S)|$; so $\delta_{H-F}(S) \neq \emptyset$, showing that H satisfies (R2).

We will consider network-design problems defined on various graphs and with various cut-requirement functions. Given a graph $D = (V_D, E_D)$ and cut-requirement function $h : 2^{V_D} \mapsto \mathbb{Z}$, we use (h, D)-NDP to denote f-NDP on the graph D with cut-requirement function h, and CR-(h, D)-NDP to denote CR-f-NDP on the graph D with base cut-requirement function h.

Properties of cut-requirement functions. Let $f : 2^V \mapsto \mathbb{R}$. We say that f is *weakly supermodular* if for any two node-sets $A, B \subseteq V$, we have $f(A) + f(B) \leq \max\{f(A \cap B) + f(A \cup B), f(A - B) + f(B - A)\}$. We say that f is *symmetric* if $f(S) = f(V - S)$ for all $S \subseteq V$, and is *normalized* if $f(\emptyset) = f(V) = 0$.

Since we are working on undirected graphs, we can always replace any cut-requirement function f by its symmetric, normalized version f^{sym}, where $f^{\mathsf{sym}}(S) := \max\{f(S), f(V-S)\}$ for all $\emptyset \neq S \subsetneq V$ and $f^{\mathsf{sym}}(\emptyset) = f^{\mathsf{sym}}(V) := 0$, without changing the resulting network-design problem(s), i.e., we have f-NDP $\equiv f^{\mathsf{sym}}$-NDP and CR-f-NDP $\equiv$ CR-f^{sym}-NDP.

Lemma 2.1 shows that weak-supermodularity is preserved under various operations. This will be quite useful in the analysis of our decomposition technique. Part a of the lemma shows that weak supermodularity is preserved under symmetrization and normalization, so for any f-NDP or CR-f-NDP instance involving a weakly-supermodular base cut-requirement function f, we may assume that f is symmetric and normalized.

All omitted proofs and details can be found in the full version of the paper.

▶ **Lemma 2.1.** *Let $f : 2^V \mapsto \mathbb{R}$ be weakly supermodular. Then*

(a) *f^{sym} is weakly supermodular.*

(b) *Let $w \in \mathbb{R}^E_+$. Define $g_1(S) := f(S) - w\big(\delta_G(S)\big)$ for all $S \subseteq V$. Then g_1 is weakly supermodular, and g_1 is symmetric if f is symmetric. In particular, for any $Z \subseteq E$, the function $f(S) - |\delta_Z(S)|$ is weakly supermodular, and is symmetric if f is symmetric.*

(c) *Let $X \subseteq V$, and $\mathcal{C}$ be a collection of subsets of $V - X$ closed under taking set intersections, unions, differences, and complements with respect to $V - X$, i.e., if $S \in \mathcal{C}$ then $V - X - S \in \mathcal{C}$, and $S, T \in \mathcal{C}$ implies that $\{S \cap T, S \cup T, S - T, T - S\} \subseteq \mathcal{C}$.*

Define the projection of f onto X with respect to $\mathcal{C}$ *as the function $g_2 : 2^X \mapsto \mathbb{R}$ given by $g_2(\emptyset) = g_2(X) := 0$, and $g_2(S) := \max_{T \in \mathcal{C}} f(S \cup T)$ for all $\emptyset \neq S \subsetneq X$. If f is symmetric, then g_2 is weakly supermodular and symmetric.*

Proof. Part (a) follows from a simple case analysis; part (b) follows because $w\big(\delta_G(S)\big)$ is symmetric, submodular. We focus on proving part (c). Essentially, the closure properties of $\mathcal{C}$ enable us to transfer the weak-supermodularity of f to the function g_2.

We first argue that g_2 is symmetric. If $A \in \{\emptyset, X\}$, we have $g_2(A) = g_2(X - A)$ by definition. Otherwise, if $g_2(A) = f(A \cup T_A)$, we have $f(A \cup T_A) = f\big(V - (A \cup T_A)\big) = f\big((X - A) \cup (V - X - T_A)\big) \leq g_2(X - A)$ where the last inequality follows since $V - X - T_A \in \mathcal{C}$. We also have $g_2(X - A) \leq g_2(A)$ by the same type of reasoning, and so $g_2(A) = g_2(X - A)$ for all $A \subseteq X$.

Now we show that g_2 is weakly supermodular. Consider $A, B \subseteq X$. If any of the sets $A - B, B - A, A \cap B, X - (A \cup B)$ are empty, then $g_2(A) + g_2(B) = g_2(A \cap B) + g_2(A \cup B)$ or $g_2(A) + g_2(B) = g_2(A - B) + g_2(B - A)$. The former holds when $A - B$ or $B - A$ is empty; the latter holds when $A \cap B$ or $X - (A \cup B)$ is empty, where we also utilize the fact that g_2 is symmetric. So assume otherwise. Then, for every $S \in \{A, B, A \cap B, A \cup B, A - B, B - A\}$, we have $g_2(S) = \max_{T \in \mathcal{C}} f(S \cup T)$.

Let $g_2(A) = f(A \cup T_A)$ and $g_2(B) = f(B \cup T_B)$, where $T_A, T_B \in \mathcal{C}$. Let $A' = A \cup T_A$ and $B' = B \cup T_B$. For any $S \in \{A' \cap B', A' \cup B', A' - B', B' - A'\}$, we can write $S = (S \cap X) \cup T$, where $T \in \mathcal{C}$ due to the closure properties of $\mathcal{C}$. It follows that $g_2(S \cap X) \geq f(S)$ for all $S \in \{A' \cap B', A' \cup B', A' - B', B' - A'\}$. So if $f(A') + f(B') \leq f(A' \cap B') + f(A' \cup B')$, we obtain that $g_2(A) + g_2(B) \leq g_2(A \cap B) + g_2(A \cup B)$, and if $f(A') + f(B') \leq f(A' - B') + f(B' - A')$, we obtain that $g_2(A) + g_{@}(B) \leq g_2(A - B) + g_2(B - A)$. ◀

The intuition behind the projection operation is as follows. Suppose f is a cut-requirement function and we know that no edges are picked from $E_G(V - X) \cup \delta_G(X)$. Then, roughly speaking, $g_2(S)$ captures the constraints that arise on $S \subseteq X$ due to f and $\mathcal{C}$.

2.1 LP-relaxation for CR-f-NDP

We consider the following natural LP-relaxation for CR-f-NDP. From now on, we assume that the cut-requirement function $f : 2^V \mapsto \mathbb{Z}$ is weakly supermodular, symmetric, and normalized. Given $\alpha \in \mathbb{R}^E$ and $F \subseteq E$, let $\alpha(F)$ denote $\sum_{e \in F} \alpha_e$.

$$\min \quad \sum_{e \in E} c_e x_e \tag{CRLP$_{f,G}$}$$

$$\text{s.t.} \quad x\big(\delta_G(S)\big) \geq \min\big\{f(S), |\delta_G(S)|\big\} \qquad \forall S \subseteq V \tag{1}$$

$$0 \leq x_e \leq 1 \qquad \forall e \in E. \tag{2}$$

The LP-relaxation for f-NDP is similar, with the cut-constraints taking the simpler form $x\big(\delta_G(S)\big) \geq f(S)$ for all $S \subseteq V$. We denote this LP by (NDP$_{f,G}$). Our algorithm is based on iterative rounding, where we iteratively fix some x_e variables to 1. Suppose that we have fixed $x_e = 1$ for all $e \in Z_1$. Then, in that iteration, we are considering the residual problem on the subgraph $G' = (V, E' = E - Z_1)$, which leads to the following variant of (CRLP$_{f,G}$):

$$\min \quad \sum_{e \in E'} c_e x_e \tag{P'}$$

$$\text{s.t.} \quad x\big(\delta_{G'}(S)\big) \geq \min\big\{f(S), |\delta_G(S)|\big\} - |\delta_{Z_1}(S)| \qquad \forall S \subseteq V \tag{3}$$

$$0 \leq x_e \leq 1 \qquad \forall e \in E'.$$

Defining $f'(S) = f(S) - |\delta_{Z_1}(S)|$ for all $S \subseteq V$, we can rephrase constraints (3) as $x\big(\delta_{G'}(S)\big) \geq \min\big\{f'(S), |\delta_{G'}(S)|\big\}$ for all $S \subseteq V$. Since f' is weakly supermodular, symmetric (Lemma 2.1 (b)), and normalized, (P') is of the same form as (CRLP$_{f,G}$): we have (P') $\equiv$ (CRLP$_{f',G'}$). This will be convenient for iterative rounding, as the residual problem is of the same form as the original problem. (CRLP$_{f,G}$) and (P') involve an exponential number of constraints, but using the ellipsoid method, we can find an extreme-point optimal solution provided we have a separation oracle for them. Recall that a separation oracle is a procedure that given a candidate point determines if it is feasible, or returns a violated inequality. Observe that a separation oracle for (CRLP$_{f,G}$) also yields a separation oracle for (P'), since (P') is simply (CRLP$_{f,G}$) after fixing some variables to 1.

▶ **Fact 2.2** (Ellipsoid method [14]). *Given a polytime separation oracle for* (CRLP$_{f,G}$)*, one can find an extreme-point optimal solution to LPs* (CRLP$_{f,G}$) *and* (P') *in polytime.*

For CR-SNDP, i.e., when $f = f^{\mathsf{SNDP}}$, a polytime separation oracle for (CRLP$_{f,G}$) can be obtained using min-cut computations. This was shown by [8] for path-relative k-ECSS, which is equivalent to cut-relative k-ECSS; refining their ideas yields a separation oracle for (CRLP$_{f,G}$) for general SNDP.

▶ **Lemma 2.3.** *When $f = f^{\mathsf{SNDP}}$, there is a polytime separation oracle for* (CRLP$_{f,G}$)*. Hence, one can obtain extreme-point optimal solutions to* (CRLP$_{f,G}$) *and* (P') *in polytime.*

Proof. The second statement follows from Fact 2.2. Let $(s_i, t_i, r_i)_{i \in [k]}$ be the k tuples defining the SNDP instance, which recall leads to the cut-requirement function $f^{\mathsf{SNDP}}(S) := \max\{r_i : |S \cap \{s_i, t_i\}| = 1\}$ for all $S \subseteq V$. Say that $S \subseteq V$ is a small cut if $f^{\mathsf{SNDP}}(S) > |\delta_G(S)|$, and it is a small cut for i if $|S \cap \{s_i, t_i\}| = 1$ and $r_i > |\delta_G(S)|$. Note that if $e \in \delta_G(S)$ for any small cut S, then we must have $x_e = 1$ in any feasible solution.

Now we can equivalently phrase the constraints of (CRLP$_{f,G}$) as follows: for any edge $a = uv \in E$, we must have $x_a = 1$ or, for every $i \in [k]$, the minimum s_i-t_i cut also separating u and v under the $\{x_e\}_{e \in E}$-capacities should have capacity at least r_i. So to detect if x is

a feasible solution, we simply need to consider every edge $a = uv$ with $x_a < 1$ and every $i \in [k]$, and find the minimum s_i-t_i cut also separating u and v under the $\{x_e\}_{e \in E}$ capacities. If an s_i-t_i cut separates u, v, then u lies on the s_i-side and v lies on the t_i-side, or the other way around. So to find the minimum s_i-t_i cut also separating u and v, we find the minimum $\{s_i, u\}$-$\{t_i, v\}$ cut – by which we mean the minimum s_i-t_i cut where s_i, u are on the same side, and t_i, v are on the other side–and the minimum $\{s_i, v\}$-$\{t_i, u\}$-cut and take the one with smaller capacity. ◀

Extreme-point solution to $(\mathrm{CRLP}_{f,G})$ not defined by a laminar family

We show that for the CR-SNDP instance shown in Fig. 1, the extreme-point solution $\widehat{x}$ shown in the figure, cannot be expressed as the solution to *any laminar family of tight cut constraints* (1) (along with tight edge constraints (2)). The SNDP instance is s-t 2-edge connectivity: that is, the base cut-requirement function is given by $f(S) = 2$ if S is an s-t cut, and $f(S) = 0$ otherwise.

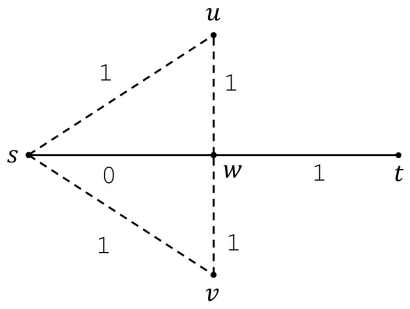


Figure 1 CR-SNDP instance, and an extreme-point solution $\widehat{x}$ to $(\mathrm{CRLP}_{f,G})$. The solid edges have $\widehat{x}_e = 1$ and the dashed edges have $\widehat{x}_e = 0.5$, so $\widehat{x}_e = 0.5$ for $e \in \{su, uw, sv, vw\}$ and $\widehat{x}_e = 1$ for $e \in \{sw, wt\}$.

Note that the only small cut above containing s is $S = \{s, u, v, w\}$. One can verify by inspection that $\widehat{x}$ is feasible. It is not hard to see that $\widehat{x}$ is the unique solution to the following set of equations, which come from constraints that are tight at $\widehat{x}$.

$$x\big(\delta(S)\big) = 2 \qquad \forall S \in \big\{\{s\}, \{s, u, v\}, \{s, u, w\}, \{s, v, w\}\big\}, \tag{4}$$

$$x_e = 1 \qquad \forall e \in \{sw, wt\}. \tag{5}$$

This shows that $\widehat{x}$ is an extreme-point solution to $(\mathrm{CRLP}_{f,G})$. With edge costs $c_{sw} = 0$ and $c_e = 1$ for all other edges, $\widehat{x}$ is also an optimal solution. The constraints $x\big(\delta(S)\big) \geq 2$ for all $S \in \big\{\{s\}, \{s, u, v\}, \{s, u, w\}, \{s, v, w\}\big\}$ imply that (since $x_e \leq 1$ for all e)

$$x_{su} + x_{sv} \geq 1, \quad x\big(\delta(u)\big) \geq 1, \quad x\big(\delta(v)\big) \geq 1, \quad x_{uw} + x_{vw} \geq 1.$$

Adding these constraints, we obtain that $x_{su} + x_{sv} + x_{uv} + x_{vw} \geq 2$. Therefore, the cost of any feasible solution is at least 3, and we have $c^T \widehat{x} = 3$.

$\widehat{x}$ cannot be defined via a laminar family. We now argue that $\widehat{x}$ cannot be defined by any laminar family of tight cut constraints and tight edge constraints. Note that any tight cut constraint corresponds to an s-t cut. So a laminar family $\mathcal{L}$ of tight cut constraints consists of a union of two chains $\mathcal{L}_1 \cup \mathcal{L}_2$, where a chain is a nested collection of sets: $\mathcal{L}_1$

comprising sets containing s but not t, and $\mathcal{L}_2$ comprising sets containing t but not s. Then $\mathcal{L}' = \mathcal{L}_1 \cup \{V - S : S \in \mathcal{L}_2\}$ is also a laminar family, yielding the same cut-constraints as $\mathcal{L}$, and and all sets in $\mathcal{L}'$contain s but not t. So we only need to consider laminar families of tight cut constraints corresponding to a chain of sets containing s and not t.

A simple argument now rules out the existence of any such chain that can uniquely define $\widehat{x}$ along with the tight edge constraints (5). Any such chain of sets $\mathcal{C}$ is obtained by ordering the elements u, v, w, and taking sets of the form $\{s\} \cup T$, where T is some prefix (including the empty prefix) of this ordered sequence; so $\mathcal{C}$ contains at most 4 sets. Since $\widehat{x}$ has 4 fractional edges, $E' = \{su, sv, uw, vw\}$, it must be that $\{\chi^{\delta_{E'}(S)}\}_{S \in \mathcal{C}}$ contains 4 linearly independent vectors. So $\mathcal{C}$ contains exactly 4 sets, and in particular the set $\{s, u, v, w\}$, and all vectors in $\{\chi^{\delta_{E'}(S)}\}_{S \in \mathcal{C}}$ are linearly independent. This yields a contradiction, since $\chi^{\delta_{E'}(\{s,u,v,w\})} = \vec{0}$.

3 Modeling power of CR-f-NDP

We prove that the CR-f-NDP Examples 1.2 and 1.3 listed under "Modeling power" in Section 1 model the stated problems. Recall that in both examples, we have a non-increasing function $\tau : \mathbb{Z}_+ \mapsto \mathbb{Z}_+$.

▶ **Theorem 3.1.** *Consider the base cut-requirement function f^{grace} in Example 1.2, where $f^{\text{grace}}(S) := \min\{\ell : \tau(\ell) < |S|\}$, for $S \neq \emptyset$.*

(a) *The function f^{grace} is weakly-supermodular.*

(b) *H is feasible for CR-f^{grace}-NDP iff for every fault-set $F \subseteq E$, and every $S \subseteq V$ with $|S| \leq \tau(|F|)$, S is (the node-set of) a component of $G - F$ iff it is a component of $H - F$.*

Proof. The function f^{grace} is *downwards-monotone*: if $\emptyset \neq T \subseteq S$, then $f^{\text{grace}}(T) \geq f^{\text{grace}}(S)$. This follows from the definition, since τ is a non-increasing function. Weak-supermodularity of f^{grace} is now immediate since we have $f^{\text{grace}}(A) + f^{\text{grace}}(B) \leq f^{\text{grace}}(A - B) + f^{\text{grace}}(B - A)$, if $A - B, B - A \neq \emptyset$, and otherwise $f^{\text{grace}}(A) + f^{\text{grace}}(B) = f^{\text{grace}}(A \cap B) + f^{\text{grace}}(A \cup B)$.

Note that f^{grace} is not symmetric or normalized, but as indicated by Lemma 2.1, we can consider CR-f-NDP with the symmetric, normalized version of f^{grace} as the base cut-requirement function, without changing the problem.

To prove the feasibility characterization in part (b), we note that the function f^{grace} is constructed so that for any $S \subseteq V$ and $F \subseteq E$, we have $|S| \leq \tau(|F|)$ iff $f^{\text{grace}}(S) > |F|$, so that $|\delta_H(S)| \geq f^{\text{grace}}(S)$ implies that S is not a component of $H - F$. Suppose H is such that $|\delta_H(S)| \geq \min\{f^{\text{grace}}(S), |\delta_G(S)|\}$ for all $S \subseteq V$. Consider any fault-set $F \subseteq E$ and any $S \subseteq V$ such that $|S| \leq \tau(|F|)$. By the above, if S is a component of $H - F$, then $|\delta_H(S)| < f^{\text{grace}}(S)$, and so S must also be a component of $G - F$. But this also implies that if S is a component of $G - F$, then it is a component of $H - F$; otherwise, there is some $A \subsetneq S$ that is a component of $H - F$ with $|A| < |S| \leq \tau(|F|)$, and so A must be a component of $G - F$, which is not the case.

Conversely, suppose that for every $F \subseteq E$, and $S \subseteq V$ with $|S| \leq \tau(|F|)$, we have that S is a component of $G - F$ iff it is a component of $H - F$. If $|\delta_H(S)| < \min\{f^{\text{grace}}(S), |\delta_G(S)|\}$ for some $S \subseteq V$, then take $F = \delta_H(S)$. Since $f^{\text{grace}}(S) > |F|$, we have $|S| \leq \tau(|F|)$. But S is a component of $H - F$ and not $G - F$, which yields a contradiction. ◀

▶ **Theorem 3.2.** *Let $\pi : 2^V \mapsto \mathbb{R}_+$ be a monotone function, i.e., $\pi(T) \leq \pi(S)$ if $T \subseteq S$, as in Example 1.3. Consider CR-f-NDP, where the base cut-requirement function f is given by $f(S) := \min\{\ell : \tau(\ell) < \pi(S)\}$, for all $S \neq \emptyset$. Then*

(a) *f is weakly-supermodular;*

(b) *H is feasible for* CR-f-NDP *iff for every fault-set $F \subseteq E$, and every $S \subseteq V$ with $\pi(S) \leq \tau(|F|)$, S is (the node-set of) a component of $G - F$ iff it is a component of $H - F$.*

Proof. We mimic the proof of Theorem 3.1. Since π is monotone, we obtain that f is downwards-monotone, and hence weakly-supermodular.

For part (b), similar to before, we have that $\pi(S) \leq \tau(|F|)$ iff $f(S) > |F|$. Suppose H is feasible for CR-f-NDP. Consider any fault-set $F \subseteq E$ and any $S \subseteq V$ be such that $\pi(S) \leq \tau(|F|)$. By the above, $f(S) > |F|$, so if S is a component of $H - F$, then it must also be a component of $G - F$. Again, this also implies that if S is a component of $G - F$, then it is a component of $H - F$: otherwise, there is some $A \subsetneq S$ that is a component of $H - F$, and $\pi(A) \leq \pi(S) \leq \tau(|F|)$, which means that A is a component of $G - F$.

Conversely, suppose that for every $F \subseteq E$, and $S \subseteq V$ with $\pi(S) \leq \tau(|F|)$, we have that S is a component of $G - F$ iff it is a component of $H - F$. If $|\delta_H(S)| < \min\{f(S), |\delta_G(S)|\}$ for some $S \subseteq V$, then take $F = \delta_H(S)$. Since $f(S) > |F|$, we have $\pi(S) \leq \tau(|F|)$. But S is a component of $H - F$ and not $G - F$, which yields a contradiction. ◀

Our intent with Examples 1.2 and 1.3 is to illustrate that CR-f-NDP can be used to model some interesting problems. The choice of the τ and/or π functions can of course impact our ability to solve the LP-relaxation (CRLP$_{f,G}$) efficiently.

4 Structure of feasible CR-f-NDP solutions: a decomposition result

We now prove our decomposition result, showing that CR-f-NDP can be reduced to f-NDP with suitably-defined weakly-supermodular cut-requirement functions. Recall that the base cut-requirement function f is weakly supermodular (and symmetric, normalized), but the cut-requirement function $g^{\text{CR-}f\text{-NDP}}$ underlying CR-f-NDP need not be. Given $x \in \mathbb{R}^E$ and $S \subseteq V$, we use x^S to denote the restriction of x to edges in S, i.e., the vector $(x_e)_{e \in E(S)} \in \mathbb{R}^{E(S)}$.

▶ **Theorem 4.1 (Decomposition result).** *There is a partition $V_1, V_2, \ldots, V_k$ of V and weakly-supermodular, symmetric, and normalized functions $f_i : 2^{V_i} \mapsto \mathbb{Z}$ such that the following hold.*

(a) *$F \subseteq E$ is a feasible solution to* CR-(f, G)-NDP *iff $F \supseteq \delta_G(V_i)$ and $F(V_i)$ is a feasible solution to* $(f_i, G[V_i])$-NDP *for all $i \in [k]$.*

(b) *$x \in \mathbb{R}^E$ is feasible to* (CRLP$_{f,G}$) *iff for all $i \in [k]$, we have $x_e = 1$ for all $e \in \delta_G(V_i)$, and x^{V_i} is feasible to* (NDP$_{f_i, G[V_i]}$).

(c) *x is an extreme-point solution to* (CRLP$_{f,G}$) *iff for all $i \in [k]$, we have $x_e = 1$ for all $e \in \delta_G(V_i)$, and x^{V_i} is an extreme-point solution to* (NDP$_{f_i, G[V_i]}$).

Before delving into the proof of Theorem 4.1, we state the following immediate corollary of Theorem 4.1 (c), which directly leads to a 2-approximation algorithm for CR-f-NDP whenever (CRLP$_{f,G}$) can be solved efficiently.

▶ **Corollary 4.2.** *Let $\widehat{x}$ be an extreme-point solution to* (CRLP$_{f,G}$). *There exists some $e \in E$ for which $\widehat{x}_e \geq 1/2$.*

Proof. By Theorem 4.1 (c), there is a partition of V and f-NDP-instances defined on each part such that $\widehat{x}$ induces an extreme-point solution in each f-NDP-instance. By Jain's result, this implies that in each of these instances, there is some edge e with $\widehat{x}_e \geq 1/2$. ◀

The rest of this section is devoted to the proof of Theorem 4.1. For a set $S \subseteq V$, define its *deficiency* $\mathrm{def}(S) := f(S) - |\delta_G(S)|$. We use $\mathrm{def}_{f,G}(S)$ (or $\mathrm{def}_{f,E}(S)$) when we want to make the cut-requirement function and graph (or edge-set) explicit. By Lemma 2.1 (b), $\mathrm{def}(S)$ is weakly supermodular; also, it is symmetric (and normalized). We say that S is a *small cut* (or more explicitly a *small-(f,G)-cut*) if $\mathrm{def}(S) > 0$. Clearly, if there is no small cut, then CR-f-NDP becomes equivalent to f-NDP. Our chief insight and key structural result is that if there is a small cut, then we can simplify the CR-f-NDP instance and move to smaller CR-f-NDP instances, by "splitting" the instance along a suitable small cut (Theorem 4.5). By repeating this operation, we eventually obtain an instance with no small cuts, which leads to Theorem 4.1. The splitting operation relies on the following modification of f.

▶ **Definition 4.3.** *Let $\emptyset \neq S \subsetneq V$. Recall that $\overline{S} := V - S$. Let $Z = \delta_G(S)$. Define the function $f_S : 2^S \mapsto \mathbb{Z}$, which we call the* restriction of f to S, *as follows.*

$$f_S(T) = \begin{cases} 0; & \text{if } T = \emptyset \text{ or } T = S \\ \max\{f(T) - |\delta_Z(T)|, f(T \cup \overline{S}) - |\delta_Z(T \cup \overline{S})|\} & \text{otherwise} \end{cases} \qquad \forall T \subseteq S.$$

The intuition here is that for $T \subseteq S$, $f_S(T)$ is a lower bound on the residual requirement that needs to be met from $\delta_{G[S]}(T)$ given that we pick all edges of Z. Note that since f and $|\delta_Z(.)|$ are symmetric, $f(T \cup \overline{S}) - |\delta_Z(T \cup \overline{S})| = f(S - T) - |\delta_Z(S - T)|$, and so f_S is the symmetric, normalized version of of the residual requirement function $f - |\delta_Z(\cdot)|$.

In the splitting operation, we pick a small cut A of *maximum* deficiency, and move to CR-f-NDP instances on the smaller graphs $G[A]$, and $G[\overline{A}]$ obtained by restricting f to A, and restricting f to $\overline{A}$ respectively (Theorem 4.5). Lemma 4.4 states some basic properties of the restriction of f. Part (b) will be quite useful in proving that the splitting operation maintains feasibility, and part (a) allows us to repeat the splitting operation on the smaller instances. For a set $S \subseteq V$, and set $F \subseteq E$ of edges, we use $F(S)$ to denote the edges of F with both ends in S.

▶ **Lemma 4.4.** *Let $\emptyset \neq S \subsetneq V$. Let f_S be the restriction of f to S. Let $\delta_G(S) \subseteq F \subseteq E$.*

(a) *f_S is weakly supermodular, symmetric, and normalized.*

(b) *Let $\emptyset \neq T \subsetneq S$. Then, $\mathrm{def}_{f_S,F(S)}(T) = \max\{\mathrm{def}_{f,F}(T), \mathrm{def}_{f,F}(T \cup \overline{S})\}$.*

Proof. Let $Z = \delta_G(S)$. Define $f' : 2^V \mapsto \mathbb{R}$ by $f'(T) := f(T) - |\delta_Z(T)|$. Then, f' is weakly supermodular and symmetric (Lemma 2.1 (b)).

For part (a), we can proceed in two ways. As noted earlier, f_S is the symmetric, normalized version f', so part (a) follows from Lemma 2.1 (a). Alternatively, observe that f_S is the projection of f' onto S with respect to $\mathcal{C} := \{\emptyset, \overline{S}\}$, which is closed under set intersections, unions, differences, and complements with respect to $V - S$. So by Lemma 2.1 (c), we obtain that f_S is weakly supermodular and symmetric. Also, f_S is normalized by definition.

For part (b), the stated equality follows by simply plugging in f_S, and noting that for $Y \in \{T, T \cup \overline{S}\}$, we have $|\delta_F(Y)| = |\delta_{F(S)}(T)| + |\delta_Z(Y)|$. ◀

▶ **Theorem 4.5** (Splitting along a small cut)**.** *Suppose that there is some small cut. Let $A \subseteq V$ be a maximum-deficiency cut. Note that $\mathrm{def}(A) > 0$ and $\emptyset \neq A \subsetneq V$.*

(a) *$F \subseteq E$ is a feasible solution to* CR-(f,G)-NDP *iff $F \supseteq \delta_G(A)$, $F(A)$ is a feasible solution to* CR-$(f_A, G[A])$-NDP*, and $F(\overline{A})$ is a feasible solution to* CR-$(f_{\overline{A}}, G[\overline{A}])$-NDP*.*

(b) *$x \in \mathbb{R}^E_+$ is feasible to $(\mathrm{CRLP}_{f,G})$ iff $x_e = 1$ for all $e \in \delta_G(A)$, x^A is feasible to $(\mathrm{CRLP}_{f_A,G[A]})$, and $x^{\overline{A}}$ is feasible to $(\mathrm{CRLP}_{f_{\overline{A}},G[\overline{A}]})$.*

Proof. Part (a) can be seen as the special case of part (b), where x is integral. We prove part (a) here, as the proof is somewhat (notational) simpler. For notational simplicity, let $(f_1, G_1, F_1) = \big(f_A, G[A], F(A)\big)$ and $(f_2, G_2, F_2) = \big(f_{\overline{A}}, G[\overline{A}], F(\overline{A})\big)$. Let $Z = \delta_G(A)$.

The "only if" direction follows easily from the definition of the restriction of f. Let F be feasible to CR-(f, G)-NDP. Since A is a small cut, we must have $F \supseteq Z$. Consider any $\emptyset \neq T \subsetneq A$. Since F is feasible, we have $|\delta_F(T)| \geq \min\big\{f(T), |\delta_G(T)|\big\}$, which implies that $|\delta_{F_1}(T)| \geq \min\big\{f(T) - |\delta_Z(T)|, |\delta_{G_1}(T)|\big\}$. Similarly, we have $|\delta_F(T \cup \overline{A})| \geq \min\big\{f(T \cup \overline{A}), |\delta_G(T \cup \overline{A})|\big\}$, so $|\delta_{F_1}(T)| \geq \min\big\{f(T \cup \overline{A}) - |\delta_Z(T \cup \overline{A})|, |\delta_{G_1}(T)|\big\}$. Combining the inequalities, and using the definition of restriction, we obtain that $|\delta_{F_1}(T)| \geq \min\big\{f_1(T), |\delta_{G_1}(T)|\big\}$. Since T was arbitrary, this shows that F_1 is feasible to CR-(f_1, G_1)-NDP.

A symmetric argument shows that F_2 is a feasible solution to CR-(f_2, G_2)-NDP.

Conversely, suppose that $F \supseteq \delta_G(A)$, $F_1 = F(A)$ is a feasible solution to CR-(f_1, G_1)-NDP, and $F_2 = F(\overline{A})$ is a feasible solution to CR-(f_2, G_2)-NDP. Recall that $Z = \delta_G(A)$. Consider any $\emptyset \neq T \subsetneq V$. Lemma 4.4 (b) can be used to readily argue that if $T \subseteq A$ or $T \subseteq \overline{A}$, then the constraint for set T in CR-(f, G)-NDP is satisfied. We show this for $T \subseteq A$; a symmetric argument applies for $T \subseteq \overline{A}$. If T is a small-(f_1, G_1)-cut, then $F_1 \supseteq \delta_{G_1}(T)$. Since $F \supseteq Z$, this implies that $F \supseteq \delta_G(T)$. Otherwise, we have $\mathrm{def}_{f_1,F_1}(T) \leq 0$, so by Lemma 4.4 (b), we have that $\mathrm{def}_{f,F}(T) \leq \mathrm{def}_{f_1,F_1}(T) \leq 0$. Thus, we always have $|\delta_F(T)| \geq \min\big\{f(T), |\delta_G(T)|\big\}$.

Now consider T such that $T \cap A, T \cap \overline{A} \neq \emptyset$. We exploit the weak supermodularity of $\mathrm{def}_{f,F}$, and that A has maximum deficiency, to bound $\mathrm{def}_{f,F}(T)$ in terms of the deficiency of sets that do not cross A, and thus show that the constraint for T is satisfied. We have

$$\mathrm{def}_{f,F}(T) + \mathrm{def}_{f,F}(A) \leq \mathrm{def}_{f,F}(T \cap A) + \mathrm{def}_{f,F}(T \cup A) = \mathrm{def}_{f,F}(T \cap A) + \mathrm{def}_{f,F}(\overline{A} - T) \quad \text{or}$$
$$\mathrm{def}_{f,F}(T) + \mathrm{def}_{f,F}(A) \leq \mathrm{def}_{f,F}(A - T) + \mathrm{def}_{f,F}(T - A)$$

where in the first inequality we also use the fact that $\mathrm{def}_{f,F}$ is symmetric. In both cases, we have $\mathrm{def}_{f,F}(T) + \mathrm{def}_{f,F}(A) \leq \mathrm{def}_{f,F}(X) + \mathrm{def}_{f,F}(Y)$ where $X \subseteq A$, $Y \subseteq \overline{A}$ and $\delta_G(T) \subseteq \delta_G(X) \cup \delta_G(Y) \cup Z$. As shown previously, for $S \in \{X, Y\}$, we have $|\delta_F(S)| \geq \min\big\{f(S), |\delta_G(S)|\big\}$, or equivalently, $\mathrm{def}_{f,F}(S) \leq \max\big\{0, \mathrm{def}_{f,G}(S)\big\}$. Note that $\mathrm{def}_{f,F}(A) = \mathrm{def}_{f,G}(A) > 0$ since $F \supseteq \delta_G(A)$. Since A is a maximum-deficiency small cut, this also implies that $\mathrm{def}_{f,F}(A) \geq \mathrm{def}_{f,F}(S)$ for $S \in \{X, Y\}$: we have $\mathrm{def}_{f,F}(S) \leq \max\big\{0, \mathrm{def}_{f,G}(S)\big\} \leq \max\big\{0, \mathrm{def}_{f,G}(A)\big\} = \mathrm{def}_{f,F}(A)$.

If both X and Y are small-(f, G)-cuts, then we are done since then we have that $F \supseteq \delta_G(X) \cup \delta_G(Y) \cup Z \supseteq \delta_G(T)$. Otherwise, since $\mathrm{def}_{f,F}(A) \geq \max\big\{\mathrm{def}_{f,F}(X), \mathrm{def}_{f,F}(Y)\big\}$, we have $\mathrm{def}_{f,F}(T) \leq \min\big\{\mathrm{def}_{f,F}(X), \mathrm{def}_{f,F}(Y)\big\} \leq 0$, since at least one of X, Y is not a small-(f, G)-cut. ◀

Proof of Theorem 4.1. Parts (a) and (b) follow easily by induction on the number of nodes, using parts (a) and (b) of Theorem 4.5. Consider part (a). The base case is when there is no small-(f, G)-cut, in which case, we have $V_1 = V$, and $f_1 = f$. Otherwise, let $A = \mathrm{argmax}_{S \subseteq V} \mathrm{def}_{f,G}(S)$. Since f_A and $f_{\overline{A}}$ are weakly-supermodular, we can recurse and induct on CR-$(f_A, G[A])$-NDP and CR-$(f_{\overline{A}}, G[\overline{A}])$-NDP instances, both of which have fewer nodes, to obtain partitions of A and $\overline{A}$; we combine these to obtain $V_1, \ldots, V_k$. Theorem 4.5 (a) and the induction hypothesis then readily yield part (a) here. The only thing to note is that since the induction hypothesis yields $F \supseteq \delta_{G[A]}(V_i)$ for a part $V_i \subseteq A$, and $F \supseteq \delta_G(A)$ by Theorem 4.5 (a), we have $F \supseteq \delta_G(V_i)$. Similarly, $F \supseteq \delta_G(V_i)$ for any part $V_i \subseteq \overline{A}$.

Part (b) follows similarly from Theorem 4.5 (b). Part (c) follows from part (b), because it is not hard to see, using part (b), that any convex combination of feasible solutions to (CRLP$_{f,G}$) yields a convex combination of feasible (NDP$_{f_i, G[V_i]}$)-solutions, and vice versa. ◀

One may wonder if the $(f_i, G[V_i])$-NDP-instances in the decomposition of Theorem 4.1 can be specified more directly or succinctly. In the full version, we prove some properties of the decomposition given by its proof, which as a *consequence*, enable one to do so. But we need our splitting approach to argue why this succinct description fulfills the properties in Theorem 4.1. The properties we establish also imply that the decomposition in Theorem 4.1 can be computed efficiently given suitable algorithmic access to the base cut-requirement function f.

5 Algorithmic implication: LP-relative 2-approximation algorithm

We now exploit the decomposition result stated in Theorem 4.1, and Corollary 4.2, to obtain a 2-approximation algorithm. We assume that we have a separation oracle for $(\text{CRLP}_{f,G})$, which implies that we can efficiently find an extreme-point optimal solution to $(\text{CRLP}_{f,G})$ and any LP that arises in a subsequent iteration after fixing some x_e variables to 1 (Fact 2.2).

Algorithm 1 CRNDP-Alg.

The input is $\big(G = (V, E), \{c_e \geq 0\}, f : 2^V \mapsto \mathbb{Z}\big)$, where f is weakly-supermodular (and symmetric, normalized). We assume there is a separation oracle for $(\text{CRLP}_{f,G})$.

A1. Initialize $F \leftarrow \emptyset$, $G' = (V, E') \leftarrow G$, and $f' \leftarrow f$.

A2. While F is not a feasible solution (which can be detected using the separation oracle), repeat:

A2.1 Find an extreme-point optimal solution $\widehat{x}$ to $(\text{CRLP}_{f',G'})$.

A2.2 Let $Z_1 = \{e \in E' : \widehat{x}_e \geq 1/2\}$. Update $F \leftarrow F \cup Z_1$, $E' \leftarrow E' - Z_1$, and $f'(S) = f'(S) - |\delta_{Z_1}(S)|$ for all $S \subseteq V$.

A3. Return F.

▶ Remark 5.1. The collection of small cuts *does not change across iterations*; that is, S is a small-(f, G)-cut iff it is a small-(f', G')-cut in some other iteration. This is because, for any $S \subseteq V$, we change $|\delta_{G'}(S)|$ and $f'(S)$ by the same amount in every iteration. In the first iteration, we have $\widehat{x}_e = 1$ for every $e \in \delta_G(S)$ and every small cut S, and so $\delta_{G'}(S) = \emptyset$ in any subsequent iteration. So we do not have any constraints for small cuts in subsequent iterations, and this causes problems with uncrossing tight constraints. Nevertheless, we are able to argue, due to our decomposition result, that there is always some edge with $\widehat{x}_e \geq 1/2$.

▶ **Theorem 5.2.** *Let OPT denote the optimal value of $(\text{CRLP}_{f,G})$. Algorithm* CRNDP-Alg *returns a feasible* CR-f-NDP*-solution $F \subseteq E$ such that $c(F) \leq 2 \cdot OPT$.*

Proof. By Corollary 4.2, at least one edge gets added to F in every iteration, so the algorithm terminates in at most $|E|$ iterations. The cost bound follows by induction on the number of iterations to termination. The base case is trivial. Suppose $F = Z_1 \cup F'$, where F' is the solution found in subsequent iterations for the instance CR-(f', G')-NDP. Let OPT' be the optimal solution to $(\text{CRLP}_{G',f'})$. Then, by induction, we have $c(F_1) \leq 2 \cdot OPT'$ and $OPT' \leq \sum_{e \in E'} c_e \widehat{x}_e$, since $(\widehat{x}_e)_{e \in E'}$ is a feasible solution to $(\text{CRLP}_{f',G'})$. We also have $c(Z_1) \leq 2 \cdot \sum_{e \in Z_1} c_e \widehat{x}_e$. Therefore, $c(F) \leq 2c^T \widehat{x} = 2 \cdot OPT$. ◀

Since $(\text{CRLP}_{f^{\text{SNDP}},G})$ admits a polytime separation oracle (Lemma 2.3), we obtain the following result as a corollary.

▶ **Theorem 5.3.** *There is a 2-approximation algorithm for* CR-SNDP.

6 Hardness result

We now show that cut-relative SNDP and path-relative SNDP are APX-hard, even when the base SNDP instance involves only one s-t pair, by showing that the k-edge connected subgraph (k-ECSS) problem can be cast as a special case of these problems. (This hints at the surprising amount of modeling power of {cut, path}- relative SNDP.) Previously, even NP-hardness of path-relative SNDP in the s-t case was not known.

Let $\big(G = (V, E), \{c_e\}_{e \in E}\big)$ be an instance of k-ECSS. We may assume that G is at least k-edge connected, i.e., $|\delta_G(S)| \geq k$ for all $\emptyset \neq S \subsetneq V$, as otherwise there is no feasible solution (and this can be efficiently detected).

Consider the following path-relative SNDP instance. We add a source s and sink t, and edges sv, vt of cost 0, for all $v \in V$. Let $G' = (V', E')$ denote this graph. The base-SNDP instance asks for $(k+n)$-edge connectivity between s and t, where $n = |V|$. That is, the base cut-requirement function f is given by $f(S') = k + n$ if $S' \subseteq V'$ is an s-t cut, and $f(S') = 0$ otherwise. We show that path-relative SNDP and cut-relative SNDP are actually equivalent in this case, and that they encode k-ECSS on the graph G.

▶ **Theorem 6.1.** *$H \subseteq E'$ is a feasible solution to the above {cut, path}-relative* SNDP *instance $\iff$ $H \supseteq \delta_{G'}(s) \cup \delta_{G'}(t)$ and $H(V)$ is a feasible k-*ECSS*-solution for G. Hence, this {cut,path}-relative* SNDP *instance is the same as k-*ECSS *on the graph G.*

Proof. Recall that we are assuming that G is (at least) k-edge connected. The following observation will be handy. The only small cuts in G' are $\{s\}$ and $\{t\}$ (and their complements). For any other s-t cut $S' \subseteq V'$, taking $S = S' \cap V$, we have that $\emptyset \neq S \subsetneq V$. So we have $|\delta_{G'}(S')| = |\delta_G(S)| + |S| + |V - S| \geq k + n$.

$\Leftarrow$ direction. We argue that H is feasible for cut-relative SNDP, which also implies that it is feasible for path-relative SNDP. Clearly, H covers both small cuts. For any other s-t cut $S' \subseteq V'$, taking $S = S' \cap V$, we have $|\delta_H(S')| \geq |\delta_{H(V)}(S)| + |S| + |V - S| \geq k + n$, since $H(V)$ is a feasible k-ECSS-solution for G.

$\Rightarrow$ direction. Suppose that H is feasible for path-relative SNDP. If there is some edge $sv \notin H$, then consider the fault-set $F = \delta_{G'}(s) - \{sv\}$. Then $H - F$ clearly has no s-t path, since $\delta_{H-F}(s) = \emptyset$. But $G' - F$ has an s-t path sv, vt. So we must have $H \supseteq \delta_{G'}(s)$. A similar argument shows that $H \supseteq \delta_{G'}(t)$.

We next argue that $|\delta_H(S')| \geq n + k$ for every s-t cut S' for which $\emptyset \neq S := S' \cap V \subsetneq V$. This is equivalent to showing that $|\delta_{H(V)}(S)| \geq k$, since $|\delta_H(S') \cap \delta_{G'}(s)| + |\delta_H(S') \cap \delta_{G'}(t)| = n$. Assume that $s \in S'$ without loss of generality. Suppose $|\delta_{H(V)}(S)| < k$. Since $|\delta_G(S)| \geq k$, there is some edge $uv \in \delta_G(S) - \delta_{H(V)}(S)$. Suppose $u \in S$. Consider the fault-set $F = \delta_{H(V)}(S) \cup \{sw : w \in V - S\} \cup \{wt : w \in S\}$. Then $|F| < k + n$ and $H - F$ has no s-t path, since we have constructed F to ensure that $\delta_{H-F}(S') = \emptyset$. However, $G - F$ has an s-t path: su, uv, vt. This contradicts that H is feasible for path-relative SNDP.

So we have shown both that $H(V)$ is feasible k-ECSS solution for G, and that H is feasible for cut-relative SNDP.

Since the cost of the edges in $E' - E$ is 0, the above {cut, path}-relative SNDP instance is exactly the same as k-ECSS on the graph G. ◀

7 Non-equivalence of path-relative SNDP and cut-relative SNDP

We consider the same graph and base SNDP instance as in Fig. 1. The graph is reproduced below for convenience, and recall that the base cut-requirement function models s-t 2-edge connectivity: so $f(S) = 2$ if S is an s-t cut, and $f(S) = 0$ otherwise.

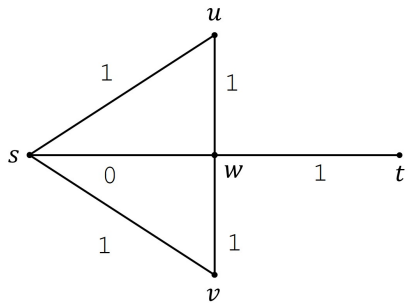


Figure 2 An instance where path-relative SNDP and cut-relative SNDP are not equivalent. The number labeling an edge gives the cost of the edge.

A feasible solution to path-relative SNDP is given by the edge-set $H_1 = \{su, uw, sw, wt\}$, which has cost 3. (This is in fact an optimal solution to path-relative SNDP.)

However, H_1 *is not feasible for cut-relative* SNDP. This is because, we require $\delta_H(\{s, u, w\}) \geq \min\{2, 3\} = 2$ in any feasible CR-SNDP solution H. Moreover, one can infer that any feasible CR-SNDP-solution must include all but one of the edges from $\{su, uw, sw, sv, vw\}$. This is because for any $F \subseteq \{su, uw, sw, sv, vw\}$ with $|F| = 2$, one can verify that there is an s-t cut S such that $|\delta_G(S)| = 3$ and $F \subseteq \delta_G(S)$; so $|\delta_{G-F}(S)| \leq 1$, which implies that there is no feasible solution that excludes F. A feasible CR-SNDP-solution must also include the edge wt, so the optimal value for cut-relative SNDP is 4.

While cut-relative SNDP and path-relative SNDP are not in general equivalent, they are closely related. Dinitz et al. [9] showed (see Lemma A.1 in [9]) that one can give a cut-based formulation for path-relative SNDP, where H is feasible iff $|\delta_H(S)| \geq \min\{f(S), |\delta_G(S)|\}$ for every $S \subseteq V$ *such that* $G[S]$ *and* $G[V' - S]$ *are connected graphs*, where $V' \subseteq V$ is the vertex-set of the component containing S. That is, we require the cut-constraints of CR-SNDP to hold for a suitable collection of node-sets (as opposed to all node-sets in CR-SNDP). This immediately implies that CR-SNDP and path-relative SNDP are equivalent when the underlying graph is a (capacitated) complete graph. Also, as mentioned earlier, path-relative k-ECSS and cut-relative k-ECSS are equivalent [8].

References

1 A. Agrawal, Phil Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24:440–456, 1995. doi:10.1137/S0097539792236237.

2 Ishan Bansal. A global analysis of the primal-dual method for edge-augmentation problems. In *Proceedings of the 26th IPCO*, pages 58–71, 2025. doi:10.1007/978-3-031-93112-3_5.

3 Ishan Bansal, Joseph Cheriyan, Logan Grout, and Sharat Ibrahimpur. Improved approximation algorithms by generalizing the primal-dual method beyond uncrossable functions. *Algorithmica*, 86(8):2575–2604, 2024. doi:10.1007/S00453-024-01235-2.

4 Nikhil Bansal, Rohit Khandekar, Jochen Könemann, Viswanath Nagarajan, and Britta Peis. On generalizations of network design problems with degree bounds. *Mathematical Programming*, 141(1-2):479–506, 2013. doi:10.1007/S10107-012-0537-8.

5 Nikhil Bansal, Rohit Khandekar, and Viswanath Nagarajan. Additive guarantees for degree-bounded directed network design. *SIAM Journal on Computing*, 39(4):1413–1431, 2009. doi:10.1137/080734340.

6 Sylvia C. Boyd, Joseph Cheriyan, Arash Haddadan, and Sharat Ibrahimpur. Approximation algorithms for flexible graph connectivity. *Mathematical Programming*, 204:493–516, 2024. doi:10.1007/S10107-023-01961-5.

7 Chandra Chekuri and Rhea Jain. Approximation algorithms for network design in non-uniform fault models. In *Proceedings of the 50th ICALP*, pages 36:1–36:20, 2023. doi:10.4230/LIPICS.ICALP.2023.36.

8 Michael Dinitz, Ama Koranteng, and Guy Kortsarz. Relative survivable network design. In *Proceedings of the 25th APPROX*, pages 41:1–41:19, 2022. doi:10.4230/LIPICS.APPROX/RANDOM.2022.41.

9 Michael Dinitz, Ama Koranteng, Guy Kortsarz, and Zeev Nutov. Improved approximations for relative survivable network design. In *Proceedings of the 21st WAOA*, pages 190–204, 2023. doi:10.1007/978-3-031-49815-2_14.

10 M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM J. on Computing*, 24:296–317, 1995. doi:10.1137/S0097539793242618.

11 M. Goemans and D. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In Dorit Hochbaum, editor, *Approximation algorithms for NP-hard problems*, chapter 4. PWS Publishing Company, 1996.

12 Michel X. Goemans, Andrew V. Goldberg, Serge A. Plotkin, David B. Shmoys, Éva Tardos, and David P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the 5th SODA*, pages 223–232, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314497.

13 M.X. Goemans. Minimum bounded-degree spanning trees. In *Proceedings of the 47th FOCS*, pages 273–282, 2006.

14 Martin Grötschel, László Lovász, and Lex Schrijver. *Geometric algorithms and combinatorial optimization*. Algorithms and Combinatorics 2. Springer, 1993.

15 Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001. doi:10.1007/S004930170004.

16 N. Klein and N. Olver. Thin trees for laminar families. In *Proceedings of the 64th FOCS*, 2023.

17 Lap Chi Lau, Seffi Naur, Mohammad Salavatipour, and Mohit Singh. Survivable network design with degree or order constraints. *SIAM Journal on Computing*, 39(3):1062–1087, 2009. doi:10.1137/070700620.

18 Lap Chi Lau, R. Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*. Cambridge University Press, 2011.

19 Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In *Proceedings of the 39th STOC*, pages 661–670. ACM, 2007. doi:10.1145/1250790.1250887.

20 David P. Williamson, Michel X. Goemans, Milena Mihail, and Vijay V. Vazirani. A primal-dual approximation algorithm for generalized steiner network problems. *Combinatorica*, 15(3):435–454, 1995. doi:10.1007/BF01299747.

Going Beyond Surfaces in Diameter Approximation

Michał Włodarczyk
University of Warsaw, Poland

Abstract

Calculating the diameter of an undirected graph requires quadratic running time under the Strong Exponential Time Hypothesis and this barrier works even against any approximation better than 3/2. For planar graphs with positive edge weights, there are known $(1+\varepsilon)$-approximation algorithms with running time $\mathsf{poly}(1/\varepsilon, \log n) \cdot n$. However, these algorithms rely on shortest path separators and this technique falls short to yield efficient algorithms beyond graphs of bounded genus.

In this work we depart from embedding-based arguments and obtain diameter approximations relying on VC set systems and the local treewidth property. We present two orthogonal extensions of the planar case by giving $(1+\varepsilon)$-approximation algorithms with the following running times:

- $\mathcal{O}_h((1/\varepsilon)^{\mathcal{O}(h)} \cdot n \log^2 n)$-time algorithm for graphs excluding an apex graph of size h as a minor,
- $\mathcal{O}_d((1/\varepsilon)^{\mathcal{O}(d)} \cdot n \log^2 n)$-time algorithm for the class of d-apex graphs.

As a stepping stone, we obtain efficient $(1+\varepsilon)$-approximate distance oracles for graphs excluding an apex graph of size h as a minor. Our oracle has preprocessing time $\mathcal{O}_h((1/\varepsilon)^8 \cdot n \log n \log W)$ and query time $\mathcal{O}_h((1/\varepsilon)^2 \cdot \log n \log W)$, where W is the metric stretch. Such oracles have been so far only known for bounded genus graphs. All our algorithms are deterministic.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases diameter, approximation, distance oracles, graph minors, treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.39

Related Version *Full Version*: https://arxiv.org/abs/2507.03447

Funding Supported by Polish National Science Centre SONATA-19 grant 2023/51/D/ST6/00155.

1 Introduction

The diameter of a graph G, denoted $\mathsf{diam}(G)$, is the largest distance between a pair of its vertices in the shortest path metric. We assume that each edge is assigned some real positive weight, and the length of a path is the sum of its weights. The obvious way to calculate $\mathsf{diam}(G)$ is to check the distances for all vertex pairs, but this inevitably requires quadratic time. This trivial approach is essentially optimal because already unweighted graphs of diameter 2 or 3 cannot be distinguished in time $\mathcal{O}(m^{1.99})$ under the Strong Exponential Time Hypothesis [49] (see [2] for hardness in sparse graphs). On the other hand, truly subquadratic algorithms to compute diameter are available in planar graphs [10, 34] and graph classes excluding a minor [26, 42, 46]. Still, the best algorithm for planar graphs runs in time[1] $\tilde{\mathcal{O}}(n^{5/3})$ [34], which is far from linear.

The absence of fast algorithms calculating the diameter exactly led the researchers to seek fast approximation algorithms. Weimann and Yuster [55] gave the first almost-linear $(1+\varepsilon)$-approximation for planar diameter, running in time $\tilde{\mathcal{O}}(2^{\mathcal{O}(1/\varepsilon)} \cdot n)$. This was improved to randomized time $\tilde{\mathcal{O}}((1/\varepsilon)^5 \cdot n)$ by Chan and Skrepetos [12] (see also [18]). Li and Parter [47] gave a deterministic algorithm of this type, as well as a distributed algorithm with a small number of rounds. Observe that each of these can be used to recognize unweighted planar

[1] The variable n refers to the number of vertices in a graph but all the considered graph classes are sparse so the number of edges is $\mathcal{O}(n)$. The $\tilde{\mathcal{O}}$-notation hides factors polylogarithmic in n. We refer to the dependence $\tilde{\mathcal{O}}(n)$ as *almost-linear*.


© Michał Włodarczyk;
licensed under Creative Commons License CC-BY 4.0

graphs of diameter D, simply by setting $\varepsilon = 1/(2D)$. There are also specialized algorithms for this task by Eppstein [28], with running time $2^{\mathcal{O}(D \log D)} \cdot n$, and by Abboud, Mozes, and Weimann [1], with running time $\tilde{\mathcal{O}}(D^2 \cdot n)$.

Shortest path separators. An area that is technique-wise related is the design of approximate distance oracles. After a preprocessing step, we would like to quickly estimate any (u, v)-distance within factor $(1 + \varepsilon)$. Thorup [53] exploited the fact that planar graphs admit balanced separators consisting of two shortest paths to give an oracle with preprocessing time $\mathcal{O}((1/\varepsilon)^2 \cdot n \log^3 n)$, space $\mathcal{O}((1/\varepsilon) \cdot n \log n)$, and query time $\mathcal{O}(1/\varepsilon)$. The shortest path separators constitute a crucial ingredient of all the three aforementioned algorithms for estimating planar diameter as well.

Abraham and Gavoille [3] presented a generalization of shortest path separators to graph classes excluding a minor and extended Thorup's oracle to such classes. However, their separators no longer enjoy such a nice structure. The main caveat is that already in a single balanced separator one might need to use paths P_1, P_2 such that P_2 is a shortest path only in $G - P_1$. Hence P_2 may be much longer than $\mathsf{diam}(G)$, which is easily seen to be unavoidable already in the class of apex graphs[2], and it is unclear how to exploit such a separator for diameter computation. The second issue is that finding the separator requires highly superlinear time, which burdens the preprocessing time of the oracle. Almost-linear preprocessing time is currently available only for graphs of bounded genus [40] and for unit disk graphs [13]. The lack of well-structured balanced separators beyond surface-embeddable graphs seems to have limited the development of efficient metric-related algorithms so far.

Beyond surface embedding. There are two directions of generalizing the topological graph classes. First, one can pick $\mathcal{O}(1)$ faces in the surface embedding of a graph and replace them with certain graphs of pathwidth $\mathcal{O}(1)$, which are called the *vortices*. Second, one can insert a set of $\mathcal{O}(1)$ vertices, called the *apices*, and connect them possibly to anyone. The structural theorem of Robertson and Seymour [48] states that for any graph H, all the graphs excluding H as a minor can be constructed from graphs embeddable in some fixed surface using these two operations and by taking a clique-sum of them.

When the excluded minor H is an apex graph, this description can be slightly restricted [25]. What is important to us, apex-minor-free graphs enjoy the *local treewidth property*: their treewidth is bounded in terms of the (unweighted) diameter and this relation is known to be linear [24]. Note that the diameter of an edge-weighted graph cannot be related to treewidth by any means because the diameter can be reduced by scaling the weights, while the topological structure of the graph remains the same. Nevertheless, we will see that the local treewidth property is still useful in the considered setting.

The main message of this work is that the local treewidth property can be used instead of shortest path separators for the sake of approximating diameter and radius, as well as for efficient construction of distance oracles. We first employ it to tackle apex-minor-free graphs but we also show that its usefulness is not limited to such graph classes.

Additive core-sets. Suppose we have a list of vertices $S = (s_0, s_1, \ldots, s_{k-1})$ in G and we want to (approximately) learn all the distance patterns, between a vertex v and all of S. Chan and Skrepetos [12] employed Cabellos's technique of abstract Voronoi diagrams [10] to

[2] G is called an *apex graph* if G can be made planar by removal of a single vertex.

build a data structure to browse such distance patterns in the case when G is planar and S lie on a single face. However, this technique relies heavily on the embedding of the graph.

A more versatile argument was given by Li and Parter [47] in order to approximate planar diameter in a distributed regime. Following their convention, a subset $U' \subseteq U$ is called a *δ-additive core-set* of U if for every $u \in U$ there exists $u' \in U'$ such that for each $i \in [0, k)$ we have $|d(u, s_i) - d(u', s_i)| \leq \delta$. When $\delta = \varepsilon \cdot \mathsf{diam}(G)$, a trivial rounding argument yields a δ-additive core-set of size $(1/\varepsilon)^k$. Remarkably, the dependence on k can be in fact removed from the exponent [47]. This theorem relies on a bound on the VC dimension of a certain set system and this argument carries over to any minor-free graph class [46]. A direct consequence of these theorems reads as follows. (Here K_h stands for the h-vertex clique.)

▶ **Lemma 1** (Corollary of [46, Theorem 1.3]). *Let G be an edge-weighted K_h-minor-free graph G, $S = (s_0, \ldots, s_{k-1}) \in V(G)^k$, $U \subseteq V(G)$, and $\varepsilon > 0$. Then U admits an $(\varepsilon \cdot \mathsf{diam}(G))$-additive core-set (with respect to S) of size $\mathcal{O}(k^{h-1} \cdot (1/\varepsilon)^h)$.*

For the sake of completeness, we provide a proof in Section 2. The local treewidth property will allow us to apply this lemma for $k = \mathsf{poly}(1/\varepsilon)$ to efficiently compress information about distances in large subgraphs.

1.1 Our contribution

A crucial property of a separator S consisting of $\mathcal{O}(1)$ shortest paths is that S can be covered by $\mathcal{O}(1/\varepsilon)$ subgraphs of strong[3] diameter $\varepsilon \cdot \mathsf{diam}(G)$. Then the interaction between the two sides of S can be channeled through $\mathcal{O}(1/\varepsilon)$ *portals* on S when small distance distortion is allowed. We show that balanced separators of this kind exist and can be efficiently found in any class excluding an apex minor. Specifically, we construct a clustering of an apex-minor-free graph into pieces of small strong diameter such that the graph obtained by contracting each cluster has bounded treewidth. We use this construction to estimate the eccentricity $\mathsf{ecc}_G(v)$ of every vertex v, that is, $\max_{u \in V(G)} d_G(v, u)$.

▶ **Theorem 2.** *Let H be an apex graph of size h. There is an algorithm that, given an edge-weighted H-minor-free graph G and $\varepsilon > 0$, runs in time $\mathcal{O}_h((1/\varepsilon)^{3h+2} \cdot n \log^2 n)$ and estimates eccentricity of each vertex vertex $v \in V(G)$ within additive error $\varepsilon \cdot \mathsf{diam}(G)$. Moreover, for each $v \in V(G)$ the algorithm outputs vertex v' for which $d_G(v, v') \geq \mathsf{ecc}_G(v) - \varepsilon \cdot \mathsf{diam}(G)$.*

As a corollary, we get $(1 + \varepsilon)$-approximation for both diameter and radius, which are respectively the largest and the smallest eccentricity.

To employ the additive core-sets in the proof of Theorem 2, we first need to precompute approximate distances for $\mathcal{O}((1/\varepsilon)^2 \cdot n \log n)$ pairs of vertices. To this end, we design a distance oracle with preprocessing and query times tailored to our needs. Similar oracles are available for minor-free graphs [3, 16, 40] but their preprocessing times are far from linear. An oracle with additive error $\varepsilon \cdot \mathsf{diam}(G)$ suffices for the proof of Theorem 2 but we can improve it to multiplicative error by taking into account the metric stretch, i.e., the ratio between the largest and smallest positive distances.

▶ **Theorem 3.** *Let H be an apex graph. Given an edge-weighted H-minor-free graph G of metric stretch W and $\varepsilon > 0$, one can build an $(1 + \varepsilon)$-approximate distance oracle with preprocessing time $\mathcal{O}_H((1/\varepsilon)^8 \cdot n \log n \log W)$ and query time $\mathcal{O}_H((1/\varepsilon)^2 \cdot \log n \log W)$.*

[3] A subgraph H of G has *strong diameter* D if every pair of vertices $u, v \in V(H)$ can be connected by a path in H of length at most D. For comparison, *weak diameter* refers to distances measured in G.

The simplest case not covered by Theorem 2 is of course given by the class of apex graphs. This class no longer enjoys the local treewidth property. Moreover, some problems which are easy on planar graphs become much harder after insertion of a single apex [5, 11]. Note that removing just a single vertex can increase the diameter significantly and so our clustering technique cannot be directly applied to decompose such a graph.

We give a stronger theorem that in particular covers the apex graphs. As a consequence of Robertson and Seymour's structural theorem, for every graph K there exists an apex graph H and a constant c, such that every K-minor-free graph can be constructed via a clique-sum from graphs $G_1, \ldots, G_\ell$ with the following property: each G_i becomes H-minor-free after deletion of c vertices [23, Theorem VI.1]. We extend Theorem 2 to such graphs, assuming that the deletion set is known.

▶ **Theorem 4.** *Let $h, c \in \mathbb{N}$ and $\mathcal{H}$ be a graph class excluding an apex graph of size h. There is an algorithm that, given an edge-weighted graph G, $\varepsilon > 0$, and a set $A \subseteq V(G)$ of size c, for which $G - A \in \mathcal{H}$, runs in time $\mathcal{O}_{h+c}((1/\varepsilon)^\alpha \cdot n \log^2 n)$, where $\alpha = \mathcal{O}((h+1)(c+1))$, and estimates $\mathsf{diam}(G)$ within factor $(1+\varepsilon)$.*

Unlike the apex-minor-free case, this time we do not estimate eccentricities because Theorem 4 involves a certain "irrelevant vertex rule" that only preserves the diameter (approximately). The assumption that the set A is given in the input can be dropped for the class $\mathcal{H}$ of planar graphs, i.e., when the input is a d-apex graph. In this case, one can find the set A in time $2^{\mathcal{O}(d \log d)} \cdot n$ [37], which leads to the following unconditional statement.

▶ **Corollary 5.** *There is an $\mathcal{O}_d((1/\varepsilon)^{\mathcal{O}(d)} \cdot n \log^2 n)$-time algorithm that estimates the diameter of an edge-weighted d-apex graph within factor $(1+\varepsilon)$.*

This result can be regarded as a generalization of the planar case in a direction that is orthogonal to Theorem 2. Unfortunately, when the class $\mathcal{H}$ excludes an arbitrary apex graph H, we do not know whether one can drop the assumption about the set A being provided. Indeed, the fastest parameterized algorithm to find set A for which $G - A$ is H-minor-free (where H is some apex graph) requires quadratic time [50]. We discuss the perspective of handling more general graph classes in Section 6.

1.2 Related Work

Apart from the ones used in this work, there are more VC set systems considered in the context of minor-free graph metrics [7, 8, 21, 22, 26, 39]. Some other approaches to such metrics are based on a decomposition into low-diameter subgraphs, including the classic KPR theorem [41]. Our low-diameter clustering scheme bears resemblance to the *shortcut partitions* [14, 15, 16] which provide even stronger guarantees but it is not known how to construct them in almost-linear time. The related *buffered cop decompositions* [16] yield cluster graphs of bounded weak coloring numbers [6]. These in turn are connected to *sparse covers* [4, 9, 29], however there the low-diameter subgraphs need not to be disjoint.

An alternative way to represent a metric by a bounded treewidth graph is a *metric embedding*. For example, an edge-weighted planar graph G can be embedded into a metric given by a graph of treewidth $f(\varepsilon)$ so that the additive distortion is $\varepsilon \cdot \mathsf{diam}(G)$ [32]. The function f can be estimated as $f(\varepsilon) = \mathcal{O}(\varepsilon^{-4})$ [14]. For apex-minor-free graphs, the current best treewidth bound is $f(\varepsilon) = 2^{\mathcal{O}(1/\varepsilon)}$ [16]. In the latter case, we again do not know if such an embedding can be computed in almost-linear time. See also [23, 30] for metric embeddings of minor-free graphs and a variant with multiplicative distortion.

For more advances in diameter computation, see [13, 17, 27], and for recent developments in planar distance oracles, see [18, 19, 33, 35, 44].

1.3 Organization of the Paper

We begin with Section 2 including preliminaries on VC set systems and additive core-sets. The following three sections contain the main technical lemmas. Sections 3 and 4 culminate with the proof of Theorem 2 while Section 5 contains the crucial lemma for Theorem 4. The formal proofs of Theorems 3 and 4 can be found in the full version of the article. The full version also contains the proofs of the statements marked with (★).

2 Preliminaries

All our statements concern undirected graphs with real positive edge weights. In an unweighted graph, each weight is set to one.

▶ **Definition 6** (Treewidth). *A* tree decomposition *of a graph G is a pair (T, χ) where T is a tree, and $\chi\colon V(T) \to 2^{V(G)}$ is a function, such that:*
1. *for each $v \in V(G)$ the nodes $\{t \mid v \in \chi(t)\}$ form a non-empty connected subtree of T,*
2. *for each edge $uv \in E(G)$ there is a node $t \in V(T)$ with $\{u, v\} \subseteq \chi(t)$.*

The width *of (T, χ) is defined as $\max_{t \in V(T)} |\chi(t)| - 1$. The* treewidth *of a graph G (denoted $\mathsf{tw}(G)$) is the minimal width a tree decomposition of G.*

VC dimension. A set system is a collection $\mathcal{F}$ of subsets of some set U. We say that a subset $Y \subseteq A$ is *shattered* by $\mathcal{F}$ if $\{Y \cap S\colon S \in \mathcal{F}\} = 2^Y$, that is, every subset of Y is an intersection of Y with some $S \in \mathcal{F}$. The VC dimension of a set system is the size of the largest shattered subset. The notion of VC dimension was introduced by Vapnik and Chervonenkis [54]. What is important to us, a set system of bounded VC dimension cannot be too large.

▶ **Lemma 7** (Sauer–Shelah Lemma [51]). *Let $\mathcal{F}$ be a set system of VC dimension d over a set of size m. Then $|\mathcal{F}| = \mathcal{O}(m^d)$.*

2.1 Additive core-sets

For a vector $\mathbf{x} \in \mathbb{R}^{\mathbf{k}}$ its ℓ_∞-norm is defined as $\ell_\infty(\mathbf{x}) = \max_{i=1}^{k} |x_i|$.

▶ **Definition 8** ([47]). *For a sequence of vertices $S = (s_0, \ldots, s_{k-1})$ from G we define the* distance tuple *of v with respect to S as $\Phi_S(v) = (d_G(v, s_0), \ldots, d_G(v, s_{k-1}))$.*

For a vertex set V, a subset $V' \subseteq V$ is called a δ-additive core-set *with respect to S if for every $v \in V$ there exists $v' \in V'$ such that $\ell_\infty(\Phi_S(v) - \Phi_S(v')) \leq \delta$.*

We will take advantage of the bound on VC dimension of certain set systems related to a graph metric. Le and Wulff-Nilsen [46] considered a slight modification of the set systems introduced by Li and Parter [47], and so they denoted them by $\widehat{\mathcal{LP}}$.

▶ **Definition 9** ([46]). *Let G be an edge-weighted graph, S be a k-tuple of vertices from G, and $M \subseteq \mathbb{R}$. For $v \in V(G)$ we define:*

$$\hat{X}_v = \{(i, \Delta)\colon i \in [1, k-1],\ \Delta \in M,\ d_G(v, s_i) - d_G(v, s_0) \leq \Delta\}.$$

Let $\widehat{\mathcal{LP}}_{G,M}(S) = \{\hat{X}_v\colon v \in V(G)\}$ be a collection of subsets of the ground set $[k-1] \times M$.

▶ **Theorem 10** ([46, Theorem 1.3]). *Let G be an edge-weighted K_h-minor free graph and S, M be as in Definition 9. Then $\widehat{\mathcal{LP}}_{G,M}(S)$ has VC dimension at most $h - 1$.*

We adapt the construction of additive core-sets from [47] to rely on the set system $\widehat{\mathcal{LP}}$.

▶ **Lemma 1** (Corollary of [46, Theorem 1.3]). *Let G be an edge-weighted K_h-minor-free graph G, $S = (s_0, \ldots, s_{k-1}) \in V(G)^k$, $U \subseteq V(G)$, and $\varepsilon > 0$. Then U admits an $(\varepsilon \cdot \mathsf{diam}(G))$-additive core-set (with respect to S) of size $\mathcal{O}(k^{h-1} \cdot (1/\varepsilon)^h)$.*

Proof. Let $D = \mathsf{diam}(G)$, $\delta = \varepsilon \cdot D/2$, and $z = \lceil 2/\varepsilon \rceil$. Next, we choose $M = \{j \cdot \delta \colon j \in [-z, z]\}$. We will extend Definition 9 slightly. For vertices u, v we define $d'(u, v)$ as the largest integer j for which $j \cdot \delta \leq d_G(u, v)$. Note that $d'(u, v)$ is always bounded by z. For a vertex v let $\hat{X}_v^\circ$ be the pair $(d'(v, s_0), \hat{X}_v)$. By Theorem 10 and Lemma 7, the family $\{\hat{X}_v \colon v \in V(G)\}$ has size $\mathcal{O}\left(((k-1) \cdot |M|)^{h-1}\right) = \mathcal{O}\left((k/\varepsilon)^{h-1}\right)$. It follows that the family $\{\hat{X}_v^\circ \colon v \in V(G)\}$ has $\mathcal{O}\left(k^{h-1}\varepsilon^{-h}\right)$ elements.

We construct U' by picking for each $X \in \{\hat{X}_v^\circ \colon v \in V(G)\}$ an arbitrary vertex $v \in U$ with $\hat{X}_v^\circ = X$. Then $|U'| = \mathcal{O}\left(k^{h-1}\varepsilon^{-h}\right)$. To show that U' is an (εD)-additive core-set, pick $u \in U$. We need to prove that there exists $u' \in U'$ such that $\ell_\infty(\Phi_S(u) - \Phi_S(u')) \leq \varepsilon D$. By construction, there is $u' \in U'$ for which $\hat{X}_u^\circ = \hat{X}_{u'}^\circ$. Now we need to argue that for each $i \in [0, k-1]$ it holds that $|d_G(u, s_i) - d_G(u', s_i)| \leq \varepsilon D$.

The simplest case is for $i = 0$. As $d'(u, s_0) = d'(u', s_0)$, there is an integer j such that both $d_G(u, s_0)$, $d_G(u', s_0)$ belong to the interval $[j \cdot \delta, (j+1) \cdot \delta)$. Hence $|d_G(u, s_0) - d_G(u', s_0)| \leq \delta = \varepsilon D/2$.

Now consider $i \geq 1$. Let j be the largest integer (possibly negative) for which $d_G(u, s_i) - d_G(u, s_0) \leq j \cdot \delta$. We define j' analogously with respect to u'. Note that $j, j' \in [-z, z]$. Because $\hat{X}_u = \hat{X}_{u'}$ it must be $j = j'$. By the same argument as before, the numbers $x = d_G(u, s_i) - d_G(u, s_0)$, $y = d_G(u', s_i) - d_G(u', s_0)$ differ by at most δ. Finally, since $|d_G(u, s_0) - d_G(u', s_0)| \leq \delta$ we obtain that $|d_G(u, s_i) - d_G(u', s_i)| \leq 2\delta = \varepsilon D$. ◀

In our setting we will only know approximations of the distances from S. We show that this suffices to construct an additive core-set.

▶ **Lemma 11.** *Let G be an edge-weighted K_h-minor-free graph G, $S = (s_0, \ldots, s_{k-1}) \in V(G)^k$, and $U \subseteq V(G)$. Let δ be given and denote $\varepsilon = \delta/\mathsf{diam}(G)$. Next, suppose that for each $v \in U$ we are given $\mathbf{y}_v \in \mathbb{R}^k$ that satisfies $\ell_\infty(\mathbf{y}_v - \Phi_S(v)) \leq \delta$. Then in time $\mathcal{O}((k/\varepsilon)^h \cdot |U|)$ we can compute a (5δ)-additive core-set of U of size $\mathcal{O}\left(k^{h-1}\varepsilon^{-h}\right)$.*

Proof. We first describe the algorithm, which is a simple greedy construction of a 3δ-cover in a metric space. Initialize $\hat{U} = \emptyset$ and iterate over $v \in U$ in any fixed order. When processing v, check if $\hat{U}$ contains u for which $\ell_\infty(\mathbf{y}_v - \mathbf{y}_u) \leq 3\delta$. If there is no such element, add v to $\hat{U}$.

First we show that every vector in $\{\mathbf{y}_v \colon v \in U\}$ is within distance 3δ in the ℓ_∞ metric from the set $\{\mathbf{y}_v \colon v \in \hat{U}\}$. Suppose otherwise and let $v \in U$ be a vertex whose vector $\mathbf{y}_v$ is further than 3δ from this set. In particular, this means that $\ell_\infty(\mathbf{y}_v - \mathbf{y}_u) > 3\delta$ for each u considered before v. So v must have been added to $\hat{U}$ at this point; a contradiction.

Now we argue that $\hat{U}$ is a (5δ)-additive core-set. For each $u \in U$ we know that there exists $v \in \hat{U}$ such that $\ell_\infty(\mathbf{y}_v - \mathbf{y}_u) \leq 3\delta$. Next, by assumption $\ell_\infty(\mathbf{y}_u - \Phi_S(u)) \leq \delta$ and likewise for v. Then by the triangle inequality we get $\ell_\infty(\Phi_S(v) - \Phi_S(u)) \leq 5\delta$, as intended.

Next, we bound the size $\hat{U}$. By construction, for each $u, v \in \hat{U}$ we have $\ell_\infty(\mathbf{y}_v - \mathbf{y}_u) > 3\delta$. We apply the triangle inequality again to get $\ell_\infty(\Phi_S(v) - \Phi_S(u)) > \delta$. By applying Lemma 1 for $\varepsilon' = \varepsilon/2$ we obtain $U' \subseteq U$ so that the $(\delta/2)$-radius balls (in the ℓ_∞ metric) centered at $\{\Phi_S(v) \colon v \in U'\}$ cover the set $\{\Phi_S(v) \colon v \in U\}$. Each such ball has diameter δ so it can contain at most one element from the set $\{\Phi_S(v) \colon v \in \hat{U}\}$. This implies $|\hat{U}| \leq |U'|$ and so the size bound follows from Lemma 1.

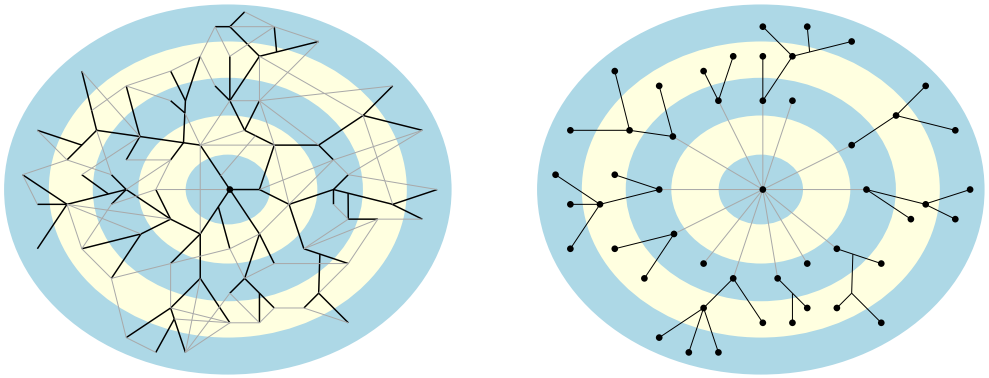


Figure 1 Construction of the clustering in Lemma 14. Left: The black edges form a shortest-path tree T rooted at r. The alternating background colors represented the layers. Each connected subgraph of T contained within a single layer forms a cluster. Right: An argument that $G_{\mathcal{C}}^{I}$ has bounded treewidth for $I = \{2, 3, 4\}$. For simplicity, the edges not originating from T are omitted. After contracting the layers 0 and 1 into a single vertex, we obtain a graph of (unweighted) diameter at most 6. Since this graph is apex-minor-free, we can use the local treewidth property.

Finally, we analyze the running time. When inspecting a vertex $u \in U$ we need to compute distances in $\mathbb{R}^k$ between $\mathbf{x}_u$ and the vectors of all the vertices already placed in the core-set. This requires at most $k \cdot |\hat{U}|$ operations and so the running time bound follows from the bound on $|\hat{U}|$. ◀

3 Low Radius Clustering

We begin with summoning the local treewidth property of apex-minor-free graphs.

▶ **Lemma 12** ([24]). *For every apex graph H, there exists a constant c such that for every unweighted H-minor-free graph G it holds that $\mathsf{tw}(G) \leq c \cdot \mathsf{diam}(G)$.*

Our first tool is a clustering of a graph into induced subgraphs of small strong diameter.

▶ **Definition 13.** *For an edge-weighted graph G, we define a* δ-radius clustering *of G as a collection of pairs $(C_i, c_i)_{i=1}^{\ell}$ with the following properties.*

1. *$(C_i)_{i=1}^{\ell}$ forms a partition of $V(G)$.*
2. *For each $i \in [\ell]$ it holds that $G[C_i]$ is connected, $c_i \in C_i$, and each $v \in C_i$ is within distance δ from c_i in $G[C_i]$.*

For $\mathcal{C} = (C_i, c_i)_{i=1}^{\ell}$ we denote by $G_{\mathcal{C}}$ the cluster graph *obtained from G by contracting each C_i into a single vertex. The vertex c_i is called the* center *of the i-th cluster.*

Note that each cluster in a δ-radius clustering has strong diameter at most 2δ. As our first goal, we aim to construct a δ-radius clustering, for $\delta = \varepsilon \cdot \mathsf{diam}(G)$, whose cluster graph has treewidth $\mathcal{O}(1/\varepsilon)$. Secondly, even when δ is chosen to be much smaller $\mathsf{diam}(G)$ we would like to put vertices u, v, satisfying $d_G(u, v) = \mathcal{O}(\delta)$, into "nearby" clusters.

For a graph G, a *layering* is simply a function $\mathsf{layer} \colon V(G) \to \mathbb{N}$. When such a function is clear from the context, for $I \subseteq \mathbb{N}$ we denote by G^I the subgraph of G induced by the vertices v with $\mathsf{layer}(v) \in I$. Given a clustering $\mathcal{C}$ of G and a layering of $G_{\mathcal{C}}$, we naturally extend it to a layering of G.

▶ **Lemma 14 (★).** *Let H be an apex graph. Given a connected edge-weighted n-vertex H-minor-free graph G and $\delta > 0$, we can in time $\mathcal{O}(n \log n)$ find a δ-radius clustering $\mathcal{C}$ of G and a layering function* $\mathsf{layer}\colon V(G_{\mathcal{C}}) \to \mathbb{N}$ *such that the following hold.*

1. *For an interval $I \subseteq \mathbb{N}$ of size p, the graph $G_{\mathcal{C}}^I$ has treewidth $\mathcal{O}_H(p)$.*
2. *The treewidth of $G_{\mathcal{C}}$ is bounded by $\mathcal{O}_H(\mathsf{diam}(G)/\delta)$.*
3. *For each $u, v \in V(G)$ and $p \in \mathbb{N}$ satisfying $d_G(u,v) \leq \delta \cdot p$, it holds that $|\mathsf{layer}(u) - \mathsf{layer}(v)| \leq p$. In this case, there is an interval $I \subseteq \mathbb{N}$ depending only on $(\mathsf{layer}(u), \mathsf{layer}(v), p)$, such that $|I| \leq 2p+1$ and the (u,v)-distance is the same in G and G^I.*

The proof constructs the layering by analyzing a shortest-path tree T rooted at an arbitrary vertex r. Then the j-th layer corresponds to vertices whose distance from r belongs to $[j \cdot \delta,\ j \cdot \delta + \delta)$. The clustering $\mathcal{C}$ is obtained by contracting all the edges in T connecting vertices sharing the same layer.

Clearly, each root-to-leaf path in T traverses at most $\mathsf{diam}(G)/\delta$ layers, which bounds the (unweighted) diameter of $G_{\mathcal{C}}$. Since $G_{\mathcal{C}}$ is a contraction of G, it is H-minor-free as well and we can take advantage of the local treewidth property. A similar argument yields Item 1 while Item 3 follows from the triangle inequality.

Lemma 12 only guarantees that $\mathsf{tw}(G_{\mathcal{C}})$ can be bounded in terms of ε, without providing an efficient way to construct a tree decomposition. Moreover, the known algorithms for computing optimal (or $\mathcal{O}(1)$-approximate) tree decomposition require time exponential in treewidth, whereas we aim at polynomial dependence on ε. Luckily, there is one algorithm available that suits our needs, with a bit worse width guarantee.

While several algorithms rely on dynamic programming over a tree decomposition to compute the diameter exactly [2, 28, 36], we cannot afford such an strategy due to prohibitive error accumulation. Instead, we will turn the decomposition into a balanced one, again by increasing its width slightly, and use it to build our data structures.

▶ **Lemma 15.** *There is an algorithm that, given a graph G of treewidth at most k, runs in time $\mathcal{O}(k^7 \cdot n \log n)$ and outputs a binary rooted tree decomposition of G of width $\mathcal{O}(k^2)$, depth $\mathcal{O}(\log n)$, and size $\mathcal{O}(n)$.*

Proof. First, in time $\mathcal{O}(k^7 \cdot n \log n)$ one can compute a tree decomposition (T, χ) of G of width $k' = \mathcal{O}(k^2)$ [31]. By a standard argument, we can modify (T, χ) to have size $\mathcal{O}(n)$. Next, we turn (T, χ) into a binary rooted tree decomposition of depth $\mathcal{O}(\log n)$ and width at most $4k' + 3$ in time $\mathcal{O}(n)$ [20]. This also preserves the bound $|V(T)| = \mathcal{O}(n)$. ◀

4 Processing a Tree Decomposition

We will need an argument to reroute paths to go through a center c_i of some cluster C_i.

▶ **Lemma 16.** *Let G be an edge-weighted graph, $u, v, w \in V(G)$, and let C be a vertex set contained in the δ-radius ball around w. Suppose there is a shortest (u,v)-path P in G that intersects C. Then $d_G(u,w) + d_G(w,v) \leq d_G(u,v) + 2\delta$.*

Proof. Orient the path P from u to v and define u', v' as the first and the last vertices from C appearing on P. Since u', v' lie on a shortest (u,v)-path, we have $d_G(u,u') + d_G(v',v) \leq d_G(u,v)$. Next, it holds that each of $d_G(w,u')$, $d_G(w,v')$ is at most δ. Hence $d_G(u,w) \leq d_G(u,u') + \delta$ and $d_G(w,v) \leq d_G(v',v) + \delta$, which yields the desired inequality. ◀

We will now show how, using a tree decomposition of the cluster graph of width $\mathcal{O}((1/\varepsilon)^2)$, to construct a distance oracle and estimate eccentricities. We give a unified proof for both statements because the constructed oracle is directly applied to obtain the second result. The presented algorithm is arguably simpler than the recursive schemes based on shortest path separators since in the analysis we only need to manage $\mathcal{O}(1)$ rounding errors, rather than $\mathcal{O}(\log n)$. We no longer need to assume that the excluded minor is an apex graph and this will allow us later to apply Theorem 17 in a more general setting.

▶ **Theorem 17.** *Suppose we are given an edge-weighted n-vertex K_h-minor-free graph G, $\delta > 0$, a δ-radius clustering $\mathcal{C}$ of G, and a binary rooted tree decomposition (T, χ) of $G_{\mathcal{C}}$ of width w, depth $\mathcal{O}(\log n)$, and size $\mathcal{O}(n)$.*

1. *In time $\mathcal{O}_h(w \cdot n \log n)$ we can build a data structure that, when queried for $u, v \in V(G)$, outputs $x \in [d_G(u,v),\, d_G(u,v) + 2 \cdot \delta]$ in time $\mathcal{O}(w \cdot \log n)$.*
2. *Assume additionally that G is connected and let $\tau = \mathsf{diam}(G)/\delta$. In time $\mathcal{O}_h(w^{h+1} \cdot \tau^h \cdot n \log^2 n)$ we can compute eccentricity of each vertex $v \in V(G)$ within additive error $16 \cdot \delta$. Moreover, within the same time we can also output a vertex $u \in V(G)$ for which $d_G(v,u) \geq \mathsf{ecc}_G(v) - 16 \cdot \delta$.*

Let us clarify that the parameter τ does not have to be given and it only appears in the analysis. However, we can always assume that the diameter is known within factor 2 after computing any eccentricity.

Proof. We begin with some notation that will be used throughout the proof. We will use variables u, v to denote vertices of G and variables i, j to index the set of clusters. For a cluster $i \in \mathcal{C}$, let $\mathsf{loc}(i)$ denote the node $t \in V(T)$ that is closest to the root among those satisfying $i \in \chi(t)$. We naturally extend this notation to vertices of G: all vertices $v \in C_i$ receive $\mathsf{loc}(v) = \mathsf{loc}(i)$. This mapping can be computed for all clusters and vertices in linear time by scanning T top-down.

For $t \in V(T)$ let $U_t \subseteq \mathcal{C}$ denote the set of clusters $i \in \mathcal{C}$ for which $\mathsf{loc}(i)$ lies in the subtree of T rooted at t, inclusively. Next, let $V_t = \bigcup_{i \in U_t} C_i$.

▷ Claim 18. It holds that $\sum_{t \in V(T)} |V_t| = \mathcal{O}(n \log n)$, and all the sets V_t can be computed in total time $\mathcal{O}(n \log n)$.

Proof. For each $i \in \mathcal{C}$ the nodes t satisfying $i \in U_t$ must be lie on the $\mathsf{loc}(i)$-to-root path in T and so there are $\mathcal{O}(\log n)$ of them. Hence each $i \in \mathcal{C}$ contributes $\mathcal{O}(|C_i| \cdot \log n)$ to the sum $\sum_{t \in V(T)} |V_t|$, which yields the upper bound as $\mathcal{C}$ is a partition of $|V(G)|$. This argument leads directly to an algorithm computing the sets V_t. ◁

Distance oracle. Consider $t \in V(T)$, $i \in \mathcal{C}$, and $v \in V(G)$. We call (t, i, v) a *valid triple* when $t = \mathsf{loc}(i)$ and $v \in V_t$. For fixed $t \in V(T)$, the number of valid triples in which t appears is at most $(w+1) \cdot |V_t|$, and so by Claim 18 the number of valid triples is $\mathcal{O}(w \cdot n \log n)$.

Recall that for $i \in \mathcal{C}$ the vertex $c_i \in C_i$ is the center of cluster i. We will populate table $\mathsf{inner\text{-}dist}$ indexed by valid triples, so that $\mathsf{inner\text{-}dist}[t, i, v]$ will store the exact (v, c_i)-distance within the graph $G[V_t]$. If there is no such path, we set $\mathsf{inner\text{-}dist}[t, i, v] = \infty$.

▷ Claim 19. We can compute $\mathsf{inner\text{-}dist}[t, i, v]$ for all valid triples in total time $\mathcal{O}_h(w \cdot n \log n)$.

Proof. Consider $t \in V(T)$. For each $i \in \mathcal{C}$ with $\log(i) = t$, we compute all distances from c_i in $G[V_t]$ using the linear single-source shortest path algorithm for K_h-minor-free graphs [52]. The total running time, over all $t \in V(T)$, is proportional to the number of valid triples. ◁

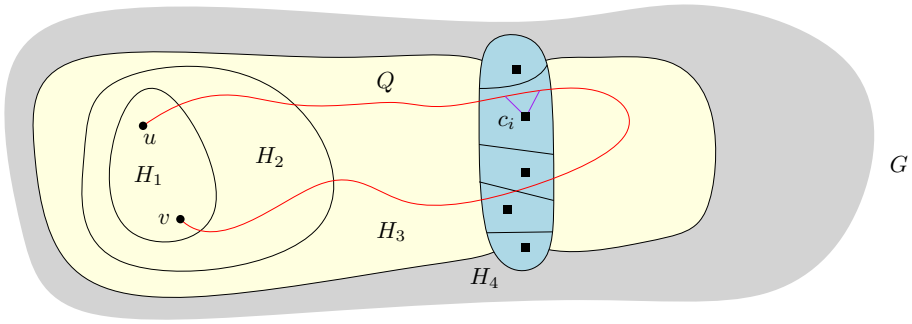


Figure 2 Illustration to Claim 20. By inspecting the nodes $t \in V(T)$ that are ancestral to both $\mathsf{loc}(u), \mathsf{loc}(v)$, we obtain a family of subgraphs $H_1 \subset H_2 \subset H_3 \subset H_4 \subset H_5 = G$, induced by the sets V_t. Here, H_4 is the smallest among them that accommodates the shortest (u, v)-path Q. This path must intersect some cluster C_i for which $\mathsf{loc}(i) = t$, where $G[V_t] = H_4$, as otherwise it would be contained in H_3. Hence the (u, v)-distance in H_4 is the same as in G and we can stretch Q within H_4 to go through c_i using Lemma 16.

We now argue that information gathered in table inner-dist is sufficient to retrieve all distances in G, up to small additive error. This is the only place where we rely on the fact that each cluster has small *strong* diameter.

▷ Claim 20. For each $u, v \in V(G)$ there exist $t \in V(T)$ and $i \in \mathcal{C}$ such that:
1. (t, i, u) and (t, i, v) are valid triples,
2. $\mathsf{inner\text{-}dist}[t, i, u] + \mathsf{inner\text{-}dist}[t, i, v] \leq d_G(u, v) + 2\delta$.

Proof. Let Q be a shortest (u, v)-path in G. Next, let $\mathcal{C}_Q \subseteq \mathcal{C}$ denote the set of clusters intersected by Q and let $T_Q \subseteq V(T)$ be the set of nodes in which $\mathcal{C}_Q$ appear, i.e., $T_Q = \chi^{-1}(\mathcal{C}_Q)$. Since $G_{\mathcal{C}}$ is a contraction of G, the subgraph $G_{\mathcal{C}}[\mathcal{C}_Q]$ is connected. Consequently, T_Q induces a connected subtree of T. Let $t \in T_Q$ be the node that is closest to the root among T_Q. Note that $t = \mathsf{loc}(i)$ for some $i \in \mathcal{C}_Q$.

We claim that the pair (t, i) satisfies the requested conditions. Note that for each $j \in \mathcal{C}_Q$ the node $\mathsf{loc}(j)$ lies in the subtree of T rooted at t. Hence $\mathcal{C}_Q \subseteq U_t$ and so $V(Q) \subseteq V_t$. In particular, we obtain that $u, v \in V_t$ so (t, i, u), (t, i, v) are valid triples (Item 1).

Since Q is a shortest (u, v)-path and $V(Q) \subseteq V_t$, we infer that the (u, v)-distance in $G[V_t]$ equals their distance in G. Now we apply Lemma 16 to the graph $H = G[V_t]$ and $C = C_i$, $w = c_i$. Because $i \in \mathcal{C}_Q$, we know that Q intersects C_i. It is crucial that all C_i-vertices are within distance δ from c_i when considering distances within $G[C_i]$ (Definition 13(2)), not just in G. As $C_i \subseteq V_t$, this property carries over to the graph H. Lemma 16 implies that $d_H(u, c_i) + d_H(c_i, v) \leq d_H(u, v) + 2\delta$. We have $\mathsf{inner\text{-}dist}(t, i, u) = d_H(u, c_i)$, $\mathsf{inner\text{-}dist}(t, i, v) = d_H(c_i, v)$, and $d_H(u, v) = d_G(u, v)$, which yields Item 2. ◁

▷ Claim 21 (Oracle). Let $u, v \in V(G)$ be given. Using the table inner-dist we can output $x \in [d_G(u, v), d_G(u, v) + 2\delta]$ in time $\mathcal{O}(w \cdot \log n)$.

Proof. We iterate over all pairs $t \in V(T)$, $i \in \mathcal{C}$ for which (t, i, u) and (t, i, v) are valid triples. There are only $\mathcal{O}(\log n)$ candidates for t just because (t, i, u) is a valid triple and so t must lie on the $\mathsf{loc}(u)$-to-root path. For each such t there are clearly at most $w + 1$ candidates for i.

We return x equal to the minimum value of $\mathsf{inner\text{-}dist}[t, i, u] + \mathsf{inner\text{-}dist}[t, i, v]$ taken over all such pairs (t, i). For each (t, i) the considered sum is either ∞ or it equals the length of some (u, v)-walk in $G[V_t]$. This implies that $x \geq d_G(u, v)$. The inequality $x \leq d_G(u, v) + 2\delta$ follows from Claim 20. ◁

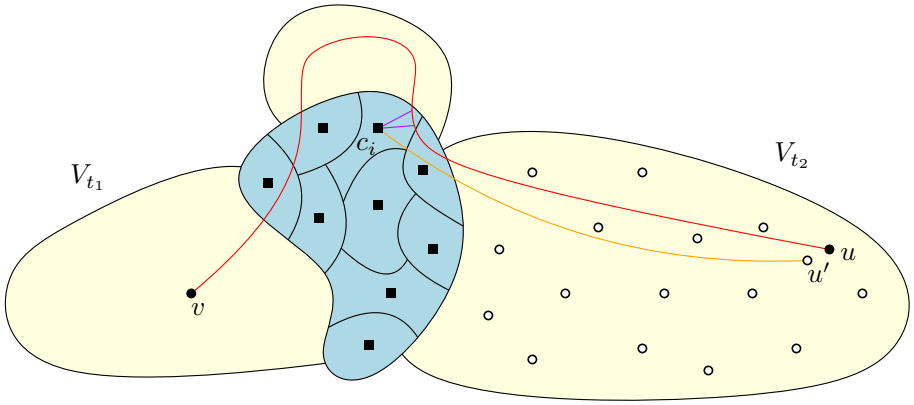


Figure 3 Illustration to Claim 23. The blue area corresponds to the union of clusters C_i over $i \in \chi(t)$. For any $v \in V_{t_1}$, $u \in V_{t_2}$ the shortest (v,u)-path (red) must traverse some C_i so by Lemma 16 there is a slightly longer path that goes through c_i (purple). The hollow discs represent the additive core-set of V_{t_2} with respect to the centers from $\chi(t)$. This core-set contains a vertex u' which is comparable to u with respect to distances from $\{c_i : i \in \chi(t)\}$ (e.g., the orange path). Hence u' can simulate u for the task of estimating the eccentricity of v.

Claims 19 and 21 yield the additive distance oracle (Item 1). Now we move on to a proof sketch of Item 2, whose complete proof can be found in the full version of the paper.

Eccentricities. For a non-root $t \in V(T)$ we refer to its parent in T as $\mathsf{parent}(t)$. For $u, v \in V(G)$ let $\mathsf{apx\text{-}dist}(u,v)$ denote the value returned by the oracle from Claim 21 when queried for the pair u, v. This is a well-defined function because the oracle is deterministic. Recall the concept of an additive core-set from Lemma 1 and that $\tau = \mathsf{diam}(G)/\delta$.

▷ Claim 22 (★). In total time $\mathcal{O}\left((w \cdot \tau)^h \cdot n + w^2 \cdot n \log^2 n\right)$ we can compute for each non-root $t \in V(T)$ a subset $V'_t \subseteq V_t$ of size $\mathcal{O}\left(w^{h-1}\tau^h\right)$ that is a $(10 \cdot \delta)$-additive core-set of V_t with respect to $\chi(\mathsf{parent}(t))$.

The total number of distances we care about is $(w+1) \cdot \sum_{t \in V(T)} |V_t| = \mathcal{O}(w \cdot n \log n)$. Each of them can be estimated using the oracle in time $\mathcal{O}(w \log n)$. We would like to use those numbers to compute the additive core-sets, however a priori the bound from Lemma 1 might not hold for approximate distances. Nevertheless, we prove that a greedy procedure still constructs a small core-set, even after perturbation of distances. A similar issue is handled in [47] via randomization but we can afford a simpler argument because our strategy allows us to first query the oracle for all the distances between V_t and $\chi(\mathsf{parent}(t))$.

In the next claim, we argue that the core-sets suffice to estimate the maximal distance between v and u, where v is fixed and u is quantified over all vertices from some subtree. There are three sources of accuracy loss: relying on the additive core-set, on the approximate distance oracle, and stretching the paths along a center from $\chi(t)$ (Lemma 16).

▷ Claim 23 (★). Consider $t \in V(T)$ with children t_1, t_2. Let $v \in V_{t_1}$ and $V' \subseteq V_{t_2}$ be a ρ-additive core-set with respect to $\chi(t)$. Let $y = \max d_G(v,u)$ over all $u \in V_{t_2}$, and x be defined as follows:

$$x = \max_{u \in V'} \min_{i \in \chi(t)} \left(\mathsf{apx\text{-}dist}(v, c_i) + \mathsf{apx\text{-}dist}(c_i, u)\right).$$

Then $|x - y| \leq \rho + 6 \cdot \delta$. Furthermore, the vertex $u^* \in V'$ realizing x satisfies $d_G(v, u^*) \geq y - \rho - 6 \cdot \delta$.

We now use the additive core-sets to estimate the eccentricity of a vertex. Essentially, we apply Claim 23 for each choice of the lowest common ancestor of v, u; hence $\mathcal{O}(\log n)$ times.

▷ Claim 24 (★). For a given vertex $v \in V(G)$ we can compute its eccentricity within additive error $16 \cdot \delta$ in time $\mathcal{O}\left(w^{h+1} \cdot \tau^h \cdot \log^2 n\right)$. Within the same time, we can output a vertex u for which $d_G(v,u) \geq \mathsf{ecc}_G(v) - 16 \cdot \delta$.

Theorem 17(2) follows by applying Claim 24 to each vertex. ◀

▶ **Theorem 2.** *Let H be an apex graph of size h. There is an algorithm that, given an edge-weighted H-minor-free graph G and $\varepsilon > 0$, runs in time $\mathcal{O}_h((1/\varepsilon)^{3h+2} \cdot n \log^2 n)$ and estimates eccentricity of each vertex vertex $v \in V(G)$ within additive error $\varepsilon \cdot \mathsf{diam}(G)$. Moreover, for each $v \in V(G)$ the algorithm outputs vertex v' for which $d_G(v,v') \geq \mathsf{ecc}_G(v) - \varepsilon \cdot \mathsf{diam}(G)$.*

Proof. One can assume that G is connected as otherwise the diameter is ∞. We compute the eccentricity D of an arbitrary vertex v: this can be done in linear time [52]. The diameter of G must be contained in $[D, 2D]$. We apply Lemma 14 for $\delta = \varepsilon D/16$ in time $\mathcal{O}(n \log n)$; let $\mathcal{C}$ denote the obtained δ-radius clustering. Next, let $\tau = \mathsf{diam}(G)/\delta$; note that $\tau = \mathcal{O}(1/\varepsilon)$. By Lemma 14(2) we know that $\mathsf{tw}(G_{\mathcal{C}}) = \mathcal{O}_h(1/\varepsilon)$. We use Lemma 15 to build a binary tree decomposition of $\mathsf{tw}(G_{\mathcal{C}})$ of width $w = \mathcal{O}_h(\varepsilon^{-2})$, depth $\mathcal{O}(\log n)$ and size $\mathcal{O}(n)$; this takes time $\mathcal{O}_h(\varepsilon^{-7} \cdot n \log n)$. Now we can take advantage of Theorem 17(2) to compute the eccentricities of all vertices within additive error $16 \cdot \delta \leq \varepsilon \cdot \mathsf{diam}(G)$, together with vertices that realize these distance approximately. This takes time $\mathcal{O}_h(w^{h+1} \cdot \tau^h \cdot n \log^2 n) = \mathcal{O}_h((1/\varepsilon)^{3h+2} \cdot n \log^2 n)$. Finally, observe that h is greater than 1 in any non-trivial instance so $7 < 3h + 2$ and so the time needed to compute the tree decomposition is negligible. ◀

In the full version of the article we show how to replace additive error in the distance oracle with multiplicative one, which yields Theorem 3.

5 Handling Apices

We move on to the proof of Theorem 4, extending the diameter approximation to graphs G with an apex set A, for which $G - A$ is apex-minor-free. We do not formulate an analogous theorem generalizing the distance oracle because it is trivial. To see this, first precompute all the distances from each $w \in A$ and then, given u, v, return the minimum of their distance in $G - A$ and $\min_{w \in A} d(u,w) + d(w,v)$. The first scenario can be readily handled with the same data structure as in Theorem 3.

The reason why we cannot apply the same reduction for diameter computation is that $\mathsf{diam}(G - A)$ might be huge compared to $\mathsf{diam}(G)$. Hence we lose the relation between the diameter and radii of clusters (governing the additive error), crucial to bound the treewidth of the cluster graph. As a remedy, we will first preprocess the graph $G - A$ to chop it into connected components of bounded diameter.

▶ **Lemma 25.** *Let H be a fixed graph and $c \in \mathbb{N}$. Let G be a connected edge-weighted n-vertex connected graph, $A \subseteq V(G)$ be such that $G - A$ is H-minor-free and $|A| = c$, and $D \in \mathbb{R}$ satisfy $2D \geq \mathsf{diam}(G)$. There is an algorithm that, given the above and $\varepsilon > 0$, runs in time $\mathcal{O}_{H,c}((1/\varepsilon)^c \cdot n)$ and either correctly reports that $\mathsf{diam}(G) > D$ or outputs a connected edge-weighted n-vertex graph G' and $A' \subseteq V(G')$ of size c, with the following guarantees.*

1. *$G' - A'$ is H-minor-free.*
2. *$\mathsf{diam}(G') \geq \mathsf{diam}(G)$.*
3. *If $\mathsf{diam}(G) \leq D$ then $\mathsf{diam}(G') \leq (1+\varepsilon) \cdot D$.*
4. *Each connected component of $G' - A'$ has strong diameter at most $8 \cdot (2/\varepsilon)^c \cdot D$.*

Proof. First of all, we remove all edges with weight larger then $2D$ as they cannot contribute to any shortest path. Observe that G is K_h-minor-free for $h = |V(H)| + c$. We compute shortest paths from each vertex $w \in A$ using the linear SSSP algorithm for minor-free graphs [52]. If any computed distance is greater than D, we can terminate the algorithm. Otherwise for each $w \in A$, $v \in V(G)$, we insert the wv-edge with weight $d_G(w, v)$ or update the weight if the edge is already present. This preserves all the pairwise distances in G and does not affect the graph $G - A$. We will keep the variable G to refer to the modified graph. The performed modification ensures the following property.

▷ Claim 26. Let $u, v \in V(G)$ and suppose that there is a shortest (u, v)-path in G that intersects A. Then there is a shortest (u, v)-path with one internal vertex that belongs to A.

For $u, v \in V(G)$ let $d'(u, v)$ denote the largest integer j for which $j \cdot \varepsilon D/2 \leq d_G(u, v)$. For $v \in V(G) \setminus A$ we define its *signature* as $S_v = (d'(v, a_1), \ldots, d'(v, a_c))$ where $(a_1, \ldots, a_c)$ is some fixed ordering of A. Let $\mathcal{S} = \{S_v \colon v \in V(G) \setminus A\}$. Since we have not terminated the algorithm in the first step, each $d'(v, a_i)$ is most $\lfloor 2/\varepsilon \rfloor$ and so $|\mathcal{S}| \leq (2/\varepsilon)^c$.

We say that a pair $S_1, S_2 \in \mathcal{S}$ is *relevant* if for each $i \in [c]$ we have $S_1[i] + S_2[i] > 2/\varepsilon$. We also call $S \in \mathcal{S}$ relevant if it appears in at least one relevant pair.

▷ Claim 27. Let $u, v \in V(G) \setminus A$ be such that (S_u, S_v) is a relevant pair. Then there is no (u, v)-path of length $\leq D$ that goes through A.

Proof. If there was such a path then $d_G(u, w) + d_G(w, v) \leq D$ for some $w \in A$. But for each $w \in A$ we have $d_G(u, w) + d_G(w, v) \geq (\varepsilon D/2) \cdot (d'(u, w) + d'(w, v)) > D$. ◁

▷ Claim 28. Let $u, v \in V(G) \setminus A$ be such that $S_u = S_v$ and S_u is relevant. If $d_{G-A}(u, v) > 2D$ then $\mathsf{diam}(G) > D$.

Proof. Suppose that $\mathsf{diam}(G) \leq D$. Since S_u is relevant, there exists $z \in V(G) \setminus A$ such that (S_u, S_z) is a relevant pair. Due to Claim 27, the shortest (u, z)-path avoids A so $d_{G-A}(u, z) \leq D$. By the same argument $d_{G-A}(v, z) \leq D$ and the claim follows from the triangle inequality. ◁

For each relevant $S \in \mathcal{S}$ let us choose an arbitrary v with $S_v = S$; we denote the set of chosen vertices as B. Clearly $|B| \leq |\mathcal{S}| \leq (2/\varepsilon)^c$. For each $b \in B$ we compute all the distances from b in $G - A$ in linear time and check whether there is any $v \in V(G) \setminus A$ with $S_v = S_b$ and $d_{G-A}(b, v) > 2D$. If so, then by Claim 28 we can again terminate the algorithm.

▷ Claim 29. Suppose that the algorithm has not been terminated and let $Z \subseteq V(G) \setminus A$ be the set of vertices z satisfying $d_{G-A}(z, b) > 3D$ for all $b \in B$. Next, let $F \subseteq E(G - A)$ be the set of edges from $G - A$ with at least one endpoint in Z. Suppose that $\mathsf{diam}(G) \leq D$. Then $\mathsf{diam}(G \setminus F) \leq (1 + \varepsilon) \cdot D$.

Proof. Consider $u, v \in V(G)$ and let P be a shortest (u, v)-path in G. By assumption, P is not longer than D. We need to show that there is a (u, v)-path of length $\leq (1 + \varepsilon) \cdot D$ that avoids F. If P intersects A then by Claim 26 we can assume that P consists only of edges incident to A (at most two). Then clearly P avoids F.

Suppose now that $V(P) \cap A = \emptyset$ and that P uses some edge from F. Then there is $z \in V(P) \cap Z$ whose distance in $G - A$ to each of u, v is at most D. We claim that S_u, S_v are irrelevant. Assume w.l.o.g. that S_u is relevant and let $b \in B$ be the chosen vertex with $S_u = S_b$. We have $d_{G-A}(u, b) \leq 2D$ as otherwise we would have terminated the algorithm due to Claim 28. But $d_{G-A}(u, z) \leq D$ so $d_{G-A}(z, b) \leq 3D$. This contradicts the property used to define Z.

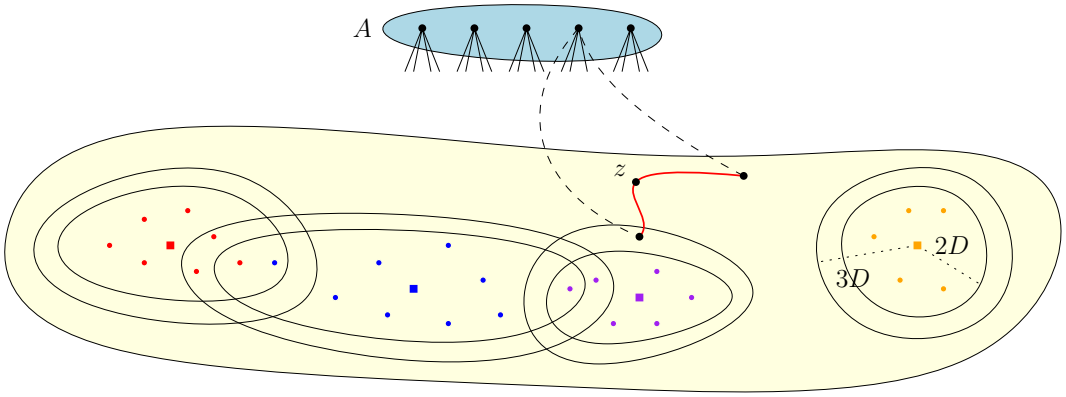


Figure 4 Illustration to Claim 29. The set B of representatives for relevant signatures is given by the squares. By Claim 28, every vertex sharing the same signature as $b \in B$ (represented by the same color) must lie in the $2D$-radius ball around b in $G - A$. If z is outside all $3D$-radius balls, then every path in $G - A$ of length $\leq D$ traversing z must connect vertices u, v for which (S_u, S_v) is an irrelevant pair. For such pairs, there must be an almost shortest path traversing A.

We established that S_u, S_v are irrelevant, so the pair (S_u, S_v) is also irrelevant. By definition, this means that there is $w \in A$ such that $d'(u, w) + d'(w, v) \leq 2/\varepsilon$. We have $d_G(u, w) \leq (\varepsilon D/2) \cdot (d'(u, w) + 1)$ and likewise for the pair (v, w). We estimate

$$d_G(u, w) + d_G(w, v) \leq (\varepsilon D/2) \cdot (d'(u, w) + d'(w, v) + 2) \leq D + (\varepsilon D/2) \cdot 2 = (1 + \varepsilon) \cdot D.$$

Hence the path (u, w, v) is sufficiently short and it avoids F. The claim follows. ◁

We showed that removing all edges from F is safe for approximating the diameter of G. It remains to prove that this operation splits $G - A$ into pieces of bounded strong diameter.

▷ Claim 30. Let $C \subseteq V(G) \setminus A$ be a set of vertices inducing a connected component of $(G - A) \setminus F$. Then $\mathsf{diam}(G[C]) \leq 8 \cdot (2/\varepsilon)^c \cdot D$.

Proof. For $b \in B$ let C_b be the set of vertices within distance $3D$ from b in $G - A$. Observe that C is either a single vertex from Z or a union of C_b for $b \in B'$ for some $B' \subseteq B$. In the first case the diameter is 0 so let us focus on the second case.

Let $u, v \in C$ and P be a shortest (u, v)-path in $G[C]$. Then $u \in C_{b_1}$ for some $b_1 \in B'$. Let v_1 be the farthest vertex on P (counting from u) that belongs to C_{b_1}. The (u, v_1)-distance in $G[C]$ is at most their distance in $G[C_{b_1}]$ so $d_P(u, v_1) \leq 6D$ as P is a shortest path. Next, take u_2 to be the next vertex on P after v_1 and pick $b_2 \in B$ satisfying $u_2 \in C_{b_2}$. Analogously as before, we find the vertex $v_2 \in C_{b_2}$ that is farthest on P and continue this process. By construction, every vertex $b \in B'$ appears at most once as b_i so this splits P into at most $|B'|$ subpaths of length $\leq 6D$. Also for each i the edge between v_i and u_{i+1} has weight at most $2D$ because we have removed all the heavier edges at the very beginning. The claim follows from $|B'| \leq |B| \leq (2/\varepsilon)^c$. ◁

We check that $G' = G \setminus F$ and $A' = A$ satisfy all the points of the lemma. First, $G' - A$ is a subgraph of $G - A$ so it remains H-minor-free. Next, removing edges can only increase the diameter so $\mathsf{diam}(G') \geq \mathsf{diam}(G)$ while the second inequality follows from Claim 29. The diameter bound for components of $G' - A'$ has been given in Claim 30. Finally, the sets Z and F can be identified in time $\mathcal{O}_H(|B| \cdot n) = \mathcal{O}_{H,c}((1/\varepsilon)^c \cdot n)$. ◀

We briefly explain how Lemma 25 leads to Theorem 4. Given $D \in [\mathsf{diam}(G), 2 \cdot \mathsf{diam}(G)]$ we want to learn that $\mathsf{diam}(G) > D$ or that $\mathsf{diam}(G) \leq (1+\varepsilon) \cdot D$. Let (G', A') be the output given by Lemma 25 and $\delta = \Theta(\varepsilon D)$. Each connected component G^i of $G' - A'$ satisfies $\mathsf{diam}(G^i) = \mathcal{O}((1/\varepsilon)^c \cdot D)$. We apply Lemma 14 to get a δ-radius clustering $\mathcal{C}^i$ of G^i so that the treewidth of the cluster graph $G^i_{\mathcal{C}^i}$ is $\mathcal{O}(\mathsf{diam}(G^i)/\delta) = \mathcal{O}((1/\varepsilon)^{c+1})$. We combine those to get a δ-radius clustering $\mathcal{C}'$ of G' by assigning each $w \in A'$ a singleton cluster.

Using Lemma 15 we can compute a balanced tree decomposition of $G'_{\mathcal{C}'}$ of width $w = \mathcal{O}((1/\varepsilon)^{2(c+1)})$. It can be now processed with Theorem 17(2): in time $\mathcal{O}(w^{h+1}(1/\varepsilon)^h \cdot n \log^2 n)$ we can estimate $\mathsf{diam}(G')$ within additive error $16 \cdot \delta = \Theta(\varepsilon D)$. If this value is too large, we learn that $\mathsf{diam}(G') > (1+\varepsilon) \cdot D$, implying $\mathsf{diam}(G) > D$. Otherwise we infer that $\mathsf{diam}(G) \leq \mathsf{diam}(G') \leq (1+\varepsilon) \cdot D$. Repeating this procedure for different D allows us to estimate $\mathsf{diam}(G)$ within multiplicative error $(1+\varepsilon)$. The details can be found in the full version of the article.

6 Conclusion and Open Problems

We have presented almost-linear algorithms approximating diameter that bypass topological arguments and work way beyond surface-embeddable graphs. A natural next step would be to generalize these results further, to all graph classes excluding a minor. The reason why we could not extend Corollary 5 to all cases covered in Theorem 4 is that we need to know the deletion set to the apex-minor-free class. The best known algorithm finding such a set requires quadratic time [50]. To apply the same strategy for the general case of H-minor-free graphs, one would need to compute some form of the Robertson and Seymour's decomposition, which seems an even harder task. However, recent advances in minor testing [43] suggest that this may be possible in time $\mathcal{O}_H(n^{1+o(1)})$ (see the section "Outlook" there). We have not attempted to generalize Theorem 4 to work on such decompositions because (1) this would require a tedious analysis of a balanced tree decomposition of the clique-sum decomposition, and (2) this strategy currently seems hopeless to achieve running time $\mathcal{O}_H(\mathsf{poly}(1/\varepsilon, \log n) \cdot n)$. Is there an alternative approach that would work without the decomposition at hand?

Another question is whether one can improve the dependence on n to $\mathcal{O}(n \log n)$ or even $\mathcal{O}(n)$? The current running time bound follows from the fact that we need to precompute $\Omega(n \log n)$ distances and each oracle call requires time $\Omega(\log n)$. Can this be improved? For example, for planar metrics there exist distance oracles with constant query time (for constant ε) albeit with preprocessing time $\Omega(n \log^3 n)$ [12, 45].

When it comes to the dependence on ε, our running time is burdened by the fact that for k-treewidth graphs we compute a tree decomposition of width $\mathcal{O}(k^2)$. Unfortunately, all the known almost-linear $\mathcal{O}(1)$-approximation algorithms for treewidth require time exponential in k. Since we rely on a construction from [31] that works for general graphs, it is tempting to improve the width guarantee therein when the input graph is H-minor-free. Again, this is doable in planar graphs [38].

References

1 Amir Abboud, Shay Mozes, and Oren Weimann. What Else Can Voronoi Diagrams Do for Diameter in Planar Graphs? In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.4.

2 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.CH28.

3 Ittai Abraham and Cyril Gavoille. Object location using path separators. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 188–197, 2006. doi:10.1145/1146381.1146411.

4 Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, and Udi Wieder. Strong-diameter decompositions of minor free graphs. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 16–24, 2007. doi:10.1145/1248377.1248381.

5 Francisco Barahona. The max-cut problem on graphs not contractible to K_5. *Operations Research Letters*, 2(3):107–111, 1983. doi:10.1016/0167-6377(83)90016-0.

6 Romain Bourneuf and Marcin Pilipczuk. Bounding ϵ-scatter dimension via metric sparsity. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 3155–3171. SIAM, 2025. doi:10.1137/1.9781611978322.101.

7 Nicolas Bousquet and Stéphan Thomassé. VC-dimension and Erdős–Pósa property. *Discrete Mathematics*, 338(12):2302–2317, 2015. doi:10.1016/j.disc.2015.05.026.

8 Vladimir Braverman, Shaofeng H-C Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in excluded-minor graphs and beyond. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2679–2696. SIAM, 2021. doi:10.1137/1.9781611976465.159.

9 Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Sparse covers for planar graphs and graphs that exclude a fixed minor. *Algorithmica*, 69:658–684, 2014. doi:10.1007/s00453-013-9757-4.

10 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *ACM Trans. Algorithms*, 15(2):21:1–21:38, 2019. doi:10.1145/3218821.

11 Dibyayan Chakraborty and Kshitij Gajjar. Finding geometric representations of apex graphs is NP-hard. *Theor. Comput. Sci.*, 971:114064, 2023. doi:10.1016/J.TCS.2023.114064.

12 Timothy M. Chan and Dimitrios Skrepetos. Faster Approximate Diameter and Distance Oracles in Planar Graphs. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2017.25.

13 Timothy M. Chan and Dimitrios Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. *Journal of Computational Geometry*, 10(2):3–20, February 2019. doi:10.20382/jocg.v10i2a2.

14 Hsien-Chih Chang, Jonathan Conroy, Hung Le, Lazar Milenkovic, Shay Solomon, and Cuong Than. Covering planar metrics (and beyond): $O(1)$ trees suffice. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2231–2261. IEEE, 2023. doi:10.1109/FOCS57990.2023.00139.

15 Hsien-Chih Chang, Jonathan Conroy, Hung Le, Lazar Milenkovic, Shay Solomon, and Cuong Than. Resolving the steiner point removal problem in planar graphs via shortcut partitions. *arXiv preprint arXiv:2306.06235*, 2023. doi:10.48550/arXiv.2306.06235.

16 Hsien-Chih Chang, Jonathan Conroy, Hung Le, Lazar Milenković, Shay Solomon, and Cuong Than. Shortcut partitions in minor-free graphs: Steiner point removal, distance oracles, tree covers, and more. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5300–5331. SIAM, 2024. doi:10.1137/1.9781611977912.191.

17 Hsien-Chih Chang, Jie Gao, and Hung Le. Computing Diameter+2 in Truly-Subquadratic Time for Unit-Disk Graphs. In Wolfgang Mulzer and Jeff M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:14, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2024.38.

18 Hsien-Chih Chang, Robert Krauthgamer, and Zihan Tan. Almost-linear ϵ-emulators for planar graphs. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1311–1324. ACM, 2022. doi:10.1145/3519935.3519998.

19 Panagiotis Charalampopoulos, Paweł Gawrychowski, Yaowei Long, Shay Mozes, Seth Pettie, Oren Weimann, and Christian Wulff-Nilsen. Almost optimal exact distance oracles for planar graphs. *Journal of the ACM*, 70(2):1–50, 2023. doi:10.1145/3580474.

20 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Optimal tree-decomposition balancing and reachability on low treewidth graphs. *IST Austria*, 2014. doi:10.15479/AT:IST-2014-314-v1-1.

21 Victor Chepoi, Bertrand Estellon, and Yann Vaxès. Covering planar graphs with a fixed number of balls. *Discret. Comput. Geom.*, 37(2):237–244, 2007. doi:10.1007/S00454-006-1260-0.

22 Vincent Cohen-Addad, Andrew Draganov, Matteo Russo, David Saulpic, and Chris Schwiegelshohn. A tight VC-dimension analysis of clustering coresets with applications. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 4783–4808. SIAM, 2025. doi:10.1137/1.9781611978322.162.

23 Vincent Cohen-Addad, Hung Le, Marcin Pilipczuk, and Michal Pilipczuk. Planar and minor-free metrics embed into metrics of polylogarithmic treewidth with expected multiplicative distortion arbitrarily close to 1. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2262–2277. IEEE, 2023. doi:10.1109/FOCS57990.2023.00140.

24 Erik D. Demaine and Mohammad Taghi Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 840–849. SIAM, 2004. URL: https://dl.acm.org/doi/10.5555/982792.982919.

25 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Approximation algorithms via structural results for apex-minor-free graphs. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 316–327. Springer, 2009. doi:10.1007/978-3-642-02927-1_27.

26 Guillaume Ducoffe, Michel Habib, and Laurent Viennot. Diameter, eccentricities and distance oracle computations on H-minor free graphs and graphs of bounded (distance) Vapnik–Chervonenkis dimension. *SIAM Journal on Computing*, 51(5):1506–1534, 2022. doi:10.1137/20M136551X.

27 Lech Duraj, Filip Konieczny, and Krzysztof Potepa. Better diameter algorithms for bounded vc-dimension graphs and geometric intersection graphs. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 51:1–51:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.51.

28 David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000. doi:10.1007/S004530010020.

29 Arnold Filtser. On sparse covers of minor free graphs, low dimensional metric embeddings, and other applications. *CoRR*, abs/2401.14060, 2024. doi:10.48550/arXiv.2401.14060.

30 Arnold Filtser and Hung Le. Low treewidth embeddings of planar and minor-free metrics. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1081–1092, 2022. doi:10.1109/FOCS54457.2022.00105.

31 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.

32 Eli Fox-Epstein, Philip N. Klein, and Aaron Schild. Embedding planar graphs into low-treewidth graphs with applications to efficient approximation schemes for metric problems. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1069–1088. SIAM, 2019. doi:10.1137/1.9781611975482.66.

33 Viktor Fredslund-Hansen, Shay Mozes, and Christian Wulff-Nilsen. Truly Subquadratic Exact Distance Oracles with Constant Query Time for Planar Graphs. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, volume 212 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:12, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2021.25.

34 Pawel Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{o}(n^{5/3})$ time. *SIAM J. Comput.*, 50(2):509–554, 2021. doi:10.1137/18M1193402.

35 Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 515–529. SIAM, 2018. doi:10.1137/1.9781611975031.34.

36 Thore Husfeldt. Computing Graph Distances Parameterized by Treewidth and Diameter. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:11, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2016.16.

37 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1802–1811. SIAM, 2014. doi:10.1137/1.9781611973402.130.

38 Frank Kammer and Torsten Tholey. Approximate tree decompositions of planar graphs in linear time. *Theor. Comput. Sci.*, 645:60–90, 2016. doi:10.1016/J.TCS.2016.06.040.

39 Adam Karczmarz and Da Wei Zheng. Subquadratic algorithms in minor-free digraphs: (weighted) distance oracles, decrementai reachability, and more. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 4338–4351. SIAM, 2025. doi:10.1137/1.9781611978322.147.

40 Ken-ichi Kawarabayashi, Philip N Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 135–146. Springer, 2011. doi:10.1007/978-3-642-22006-7_12.

41 Philip Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 682–690, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/167088.167261.

42 Kacper Kluk, Marcin Pilipczuk, Michal Pilipczuk, and Giannos Stamoulis. Faster diameter computation in graphs of bounded Euler genus. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, July 8-11, 2025, Aarhus, Denmark*, volume 334 of *LIPIcs*, pages 109:1–109:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ICALP.2025.109.

43 Tuukka Korhonen, Michal Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 53–61. IEEE, 2024. doi:10.1109/FOCS61266.2024.00014.

44 Hung Le. Approximate distance oracles for planar graphs with subpolynomial error dependency. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1877–1904. SIAM, 2023. doi:10.1137/1.9781611977554.CH72.

45 Hung Le and Christian Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 363–374. IEEE, 2021. doi:10.1109/FOCS52979.2021.00044.

46 Hung Le and Christian Wulff-Nilsen. VC set systems in minor-free (di)graphs and applications. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 5332–5360. SIAM, 2024. doi:10.1137/1.9781611977912.192.

47 Jason Li and Merav Parter. Planar diameter via metric compression. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 152–163, 2019. doi:10.1145/3313276.3316358.

48 Neil Robertson and Paul D Seymour. Graph minors. XVI. Excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43–76, 2003. doi:10.1016/S0095-8956(03)00042-X.

49 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 515–524, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2488608.2488673.

50 Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. k-apices of Minor-closed Graph Classes. II. Parameterized Algorithms. *ACM Trans. Algorithms*, 18(3):21:1–21:30, 2022. doi:10.1145/3519028.

51 Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972. doi:10.1016/0097-3165(72)90019-2.

52 Siamak Tazari and Matthias Müller-Hannemann. Shortest paths in linear time on minor-closed graph classes, with an application to steiner tree approximation. *Discret. Appl. Math.*, 157(4):673–684, 2009. doi:10.1016/J.DAM.2008.08.002.

53 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, November 2004. doi:10.1145/1039488.1039493.

54 Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity: festschrift for alexey chervonenkis*, pages 11–30. Springer, 2015. doi:10.1007/978-3-319-21852-6_3.

55 Oren Weimann and Raphael Yuster. Approximating the diameter of planar graphs in near linear time. *ACM Transactions on Algorithms (TALG)*, 12(1):1–13, 2015. doi:10.1145/2764910.

Maximum List r-Colorable Induced Subgraphs in kP_3-Free Graphs

Esther Galby
Chalmers University of Technology, Gothenburg, Sweden
University of Gothenburg, Sweden

Paloma T. Lima
IT University of Copenhagen, Denmark

Andrea Munaro
Department of Mathematical, Physical and Computer Sciences, University of Parma, Italy

Amir Nikabadi
IT University of Copenhagen, Denmark

Abstract

We show that, for every fixed positive integers r and k, Max-Weight List r-Colorable Induced Subgraph admits a polynomial-time algorithm on kP_3-free graphs. This problem is a common generalization of Max-Weight Independent Set, Odd Cycle Transversal and List r-Coloring, among others. Our result has several consequences.

First, it implies that, for every fixed $r \geq 5$, assuming $\mathsf{P} \neq \mathsf{NP}$, Max-Weight List r-Colorable Induced Subgraph is polynomial-time solvable on H-free graphs if and only if H is an induced subgraph of either kP_3 or $P_5 + kP_1$, for some $k \geq 1$. Second, it makes considerable progress toward a complexity dichotomy for Odd Cycle Transversal on H-free graphs, allowing to answer a question of Agrawal, Lima, Lokshtanov, Rzążewski, Saurabh, and Sharma [ACM Trans. Algorithms 2025]. Third, it gives a short and self-contained proof of the known result of Chudnovsky, Hajebi, and Spirkl [Combinatorica 2024] that List r-Coloring on kP_3-free graphs is polynomial-time solvable for every fixed r and k.

We also consider two natural distance-d generalizations of Max-Weight Independent Set and List r-Coloring and provide polynomial-time algorithms on kP_3-free graphs for every fixed integers r, k, and $d \geq 6$.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Graph algorithms analysis

Keywords and phrases Hereditary classes, list coloring, odd cycle transversal, independent set

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.40

Related Version *Full Version*: https://arxiv.org/abs/2505.00412

Funding *Paloma T. Lima*: Supported by Independent Research Fund Denmark grant agreement number 2098-00012B.
Amir Nikabadi: Supported by Independent Research Fund Denmark grant agreement number 2098-00012B.

1 Introduction

A fundamental class of graph optimization problems consists in finding a maximum-weight induced subgraph satisfying a certain fixed property Π. Lewis and Yannakakis [12] showed that, whenever this fixed property Π is nontrivial and hereditary, the corresponding problem is NP-hard. In this paper, we investigate the case where Π is the property of being list r-colorable, leading to a meta-problem called Max-Weight List r-Colorable Induced Subgraph. In order to properly define this problem, we first require some definitions.


© Esther Galby, Paloma T. Lima, Andrea Munaro, and Amir Nikabadi;
licensed under Creative Commons License CC-BY 4.0



Let $G = (V, E)$ be a finite simple graph. A *coloring* of G is a mapping $\phi\colon V \to \{1, 2, \ldots\}$ that gives each vertex $u \in V$ a *color* $\phi(u)$ in such a way that, for every two adjacent vertices u and v in G, we have that $\phi(u) \neq \phi(v)$. For $r \geq 1$, a coloring ϕ of G is an *r-coloring* if $\phi(u) \in \{1, \ldots, r\}$ for every $u \in V$, and a graph is *r-colorable* if it admits an r-coloring. For $r \geq 1$, an *r-list assignment* of G is a function $L\colon V \to 2^{\{1,\ldots,r\}}$ that assigns each vertex $u \in V$ a *list* $L(u) \subseteq \{1, \ldots, r\}$ of admissible colors for u. A coloring ϕ of G *respects* L if $\phi(u) \in L(u)$ for every $u \in V$. We are finally ready to define Max-Weight List r-Colorable Induced Subgraph, where r is a fixed positive integer.

Max-Weight List r-Colorable Induced Subgraph

Input: A graph G equipped with a weight function $w\colon V(G) \to \mathbb{Q}_+$, and an r-list assignment L of G.
Task: Find a subset $F \subseteq V(G)$ such that:
1. The induced subgraph $G[F]$ admits a coloring that respects L, and
2. The weight $w(F) = \sum_{v \in F} w(v)$ is maximum subject to the condition above.

Max-Weight List r-Colorable Induced Subgraph is a common generalization of several well-known and deeply investigated NP-hard problems, as we explain next. Max-Weight List r-Colorable Induced Subgraph generalizes List r-Coloring and hence r-Coloring as well, which are known to be NP-hard for all $r > 2$ [11]. Recall that, for a fixed $r \geq 1$, List r-Coloring is the problem to decide whether a given graph G with an r-list assignment L admits a coloring that respects L. By setting $L(u) = \{1, \ldots, r\}$ for every $u \in V(G)$, we obtain r-Coloring. Note also that, for $r_1 \leq r_2$, List r_1-Coloring is a special case of List r_2-Coloring.

Several other NP-hard problems are special cases of Max-Weight List r-Colorable Induced Subgraph for specific values of r. For example, for $r = 1$, Max-Weight List r-Colorable Induced Subgraph is equivalent to Max-Weight Independent Set, which is the problem of finding a maximum-weight subset of pairwise non-adjacent vertices of an input graph G. For $r = 2$, Max-Weight List r-Colorable Induced Subgraph generalizes the problem of finding a maximum-weight induced bipartite subgraph of an input graph G which, by complementation, is equivalent to finding a minimum-weight subset of vertices intersecting all odd cycles in G. The latter is the well-known Odd Cycle Transversal.

Given the hardness of Max-Weight List r-Colorable Induced Subgraph, it is natural to investigate whether the problem becomes tractable for restricted classes of inputs. The framework of hereditary graph classes (i.e., graph classes closed under vertex deletion) is particularly well suited for this type of research, where the ultimate goal is to obtain complexity dichotomies telling us for which hereditary graph classes the problem at hand can or cannot be solved efficiently (under the standard complexity assumption that P $\neq$ NP).

We recall some relevant definitions. A graph G is *H-free*, for some graph H, if it contains no induced subgraph isomorphic to H. For a set of graphs $\{H_1, \ldots, H_p\}$, a graph is *$(H_1, \ldots, H_p)$-free* if it is H_i-free for every $i \in \{1, \ldots, p\}$. The *disjoint union* $G + H$ of graphs G and H is the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$. We denote the disjoint union of k copies of G by kG and let P_s denote the chordless path on s vertices.

It is known that, for $r = 2$, Max-Weight List r-Colorable Induced Subgraph admits no polynomial-time algorithm on H-free graphs unless H is a *linear forest* (i.e., a disjoint union of paths). Indeed, Chiarelli et al. [3] showed that its special case Odd Cycle Transversal is NP-hard on H-free graphs if H contains a cycle or a claw (the claw is the 4-vertex star). In recent years, considerable work has been done toward classifying the

complexity of Odd Cycle Transversal (and its generalizations) on graphs forbidding an induced linear forest. In Theorem 1, we collect known results for Odd Cycle Transversal and its two generalizations Max-Weight r-Colorable Induced Subgraph and Max-Weight List r-Colorable Induced Subgraph on H-free graphs, where H is a linear forest. The problems are listed in increasing order of generality. In particular, an NP-hardness result for a certain problem implies NP-hardness for a more general problem. Note also that the hardness results hold even in the unweighted setting.

▶ **Theorem 1.** *The following hold:*

(i) *Odd Cycle Transversal on H-free graphs can be solved in polynomial time if*
- $H = P_5$ (Agrawal et al. [1]), *or*
- $H = kP_2$ *for all* $k \in \mathbb{N}$ (Chiarelli et al. [3]), *or*
- $H = P_3 + kP_1$ *for all* $k \in \mathbb{N}$ (Dabrowski et al. [7]),

and remains NP-hard if
- $H = (P_6, P_5 + P_2)$ (Dabrowski et al. [7]).

(ii) *Max-Weight r-Colorable Induced Subgraph on H-free graphs can be solved in polynomial time if*
- $H = P_5 + kP_1$ *for all* $k \in \mathbb{N}$ (Henderson et al. [10]),

and remains NP-hard if
- $H = (P_6, P_5 + P_2)$ *for* $r = 2$, *or*
- $H = 2P_4$ *for all* $r \geq 5$ (Hajebi et al. [9]).

(iii) *Max-Weight List r-Colorable Induced Subgraph on H-free graphs can be solved in polynomial time if*
- $H = P_5$ (Lokshtanov et al. [13]),
- $H = P_5 + kP_1$ *for all* $k \in \mathbb{N}$ (Henderson et al. [10]),

and remains NP-hard if
- $H = (P_6, P_5 + P_2)$ *for* $r \geq 2$, *or*
- $H = P_4 + P_2$ *for all* $r \geq 5$ (Couturier et al. [6]).

Recently, Chudnovsky et al. [4] obtained the following complete complexity dichotomy for List r-Coloring when $r \geq 5$. Assuming $\mathsf{P} \neq \mathsf{NP}$, List r-Coloring ($r \geq 5$) can be solved in polynomial time on H-free graphs if and only if H is an induced subgraph of either kP_3 or $P_5 + kP_1$, for some $k \in \mathbb{N}$. Their main result toward this was showing that List r-Coloring ($r \geq 1$) can be solved in polynomial time on kP_3-free graphs, for any $k \in \mathbb{N}$, and this was obtained building on a very technical result of Hajebi et al. [9, Theorem 5.1].

Motivated by the quest for a complexity dichotomy, Agrawal et al. [1] posed very recently as an open problem to classify the computational complexity of Odd Cycle Transversal on $(P_3 + P_2)$-free graphs, the unique minimal open case stemming from Theorem 1.

It should also be mentioned that classifying the complexity of Max-Weight Independent Set on H-free graphs when H is a linear forest is a notorious open problem in algorithmic graph theory (see [5] for the state of the art). Note however that Max-Weight Independent Set substantially differs from Odd Cycle Transversal on H-free graphs, in the sense that it is polynomial-time solvable on claw-free graphs.

In this paper, we also consider the distance-d generalizations of Max-Weight Independent Set and List r-Coloring, defined as follows. For $d \geq 2$, a *distance-d independent set* of a graph G is a set of vertices of G pairwise at distance at least d in G. For fixed $d \geq 2$, Max-Weight Distance-d Independent Set (also known as d-Scattered Set) is the problem to find a maximum-weight distance-d independent set of an input graph G.

Trivially, for every fixed $d \geq 2$, Max-Weight Distance-d Independent Set is easy on P_{d+1}-free graphs, and the following hardness results are known.

▶ **Theorem 2.** *The following hold for Distance-d Independent Set on H-free graphs:*
- **(i)** *It is* NP*-hard if H contains $2P_{(d+1)/2}$, for every fixed odd $d \geq 3$* (Eto et al. [8])*;*
- **(ii)** *Assuming ETH, it admits no $2^{o(|V(H)|+|E(H)|)}$-time algorithm if H contains a cycle or a claw, for every fixed $d \geq 3$* (Bacsó et al. [2])*.*

For $d \geq 2$, a (d, r)-*coloring* of a graph G is an assignment of colors to the vertices of G using at most r colors such that no two distinct vertices at distance less than d receive the same color. Thus, a $(2, r)$-coloring is nothing but an r-coloring. Similarly as above, for fixed $d \geq 2$ and $r \geq 1$, we define (d, r)-Coloring as the problem of determining whether a given graph G has a (d, r)-coloring. List (d, r)-Coloring is defined similarly but we require in addition that every vertex u must receive a color from some given list $L(u) \subseteq \{1, \ldots, r\}$.

Sharp [18] provided the following complexity dichotomy: For fixed $d \geq 3$, (d, r)-Coloring is polynomial-time solvable for $r \leq \lfloor 3d/2 \rfloor$ and NP-hard for $r > \lfloor 3d/2 \rfloor$.

Our results. We prove three main algorithmic results for kP_3-free graphs. The first result, proven in Section 3, concerns Max-Weight List r-Colorable Induced Subgraph.

▶ **Theorem 3.** *Let $r \geq 1$ be a fixed integer. For every $k \in \mathbb{N}$, Max-Weight List r-Colorable Induced Subgraph can be solved in polynomial time on kP_3-free graphs.*

Theorem 3 has several interesting consequences. First, it immediately implies that Odd Cycle Transversal can be solved in polynomial time on kP_3-free graphs, for every $k \in \mathbb{N}$, thus solving a generalized version of the aforementioned open problem of Agrawal et al. [1]. Our result for Odd Cycle Transversal also complements the polynomial-time algorithms for Feedback Vertex Set and Even Cycle Transversal on kP_3-free graphs of Paesani et al. [16]. Second, Theorem 3 generalizes the recent result of Chudnovsky et al. [4] that List r-Coloring can be solved in polynomial time on kP_3-free graphs for every $r, k \in \mathbb{N}$. Although partially inspired by their approach, as we explain below, our proof of the more general Theorem 3 has the advantage of being considerably shorter and self-contained.

Theorem 3 also makes considerable progress toward a complete complexity dichotomy for Max-Weight List r-Colorable Induced Subgraph and Odd Cycle Transversal on H-free graphs. Indeed, paired with the recent result of [10], it *completely* settles the complexity of Max-Weight List r-Colorable Induced Subgraph on H-free graphs for $r \geq 5$ (see Theorem 1 and the discussion preceding it):

▶ **Theorem 4.** *Let $r \geq 5$ be a fixed integer. Assuming* P $\neq$ NP*, Max-Weight List r-Colorable Induced Subgraph on H-free graphs is polynomial-time solvable if and only if H is an induced subgraph of either kP_3 or $P_5 + kP_1$, for some $k \geq 1$.*

Moreover, paired with the results of [3, 7, 10] mentioned above, Theorem 3 leaves the case $H = k_4P_4 + k_3P_3 + k_2P_2 + k_1P_1$, with $k_4 \geq 1$ and $k_4 + k_3 \geq 2$, as the only remaining open case toward a complete complexity dichotomy for Odd Cycle Transversal on H-free graphs.

We then consider the distance-d generalizations of Max-Weight Independent Set and List r-Coloring, for $d \geq 6$, and prove the following two results.

▶ **Theorem 5.** *Let $d \geq 6$ be a fixed integer. For every $k \in \mathbb{N}$, Max-Weight Distance-d Independent Set can be solved in polynomial time on kP_3-free graphs.*

▶ **Theorem 6.** *Let $d \geq 6$ and $r \geq 1$ be fixed integers. For every $k \in \mathbb{N}$, List (d, r)-Coloring can be solved in polynomial time on kP_3-free graphs.*

Paired with the aforementioned result of Eto et al. [8], Theorem 5 completely settles the computational complexity of Max-Weight Distance-d Independent Set on kP_3-free graphs, except for the *only* remaining open case $d = 4$.

Technical overview. We now explain our approach toward Theorem 3, which combines ideas from [14, 15] and [4]. It is instructive to first consider the special case of Odd Cycle Transversal (by complementation, Max-Weight 2-Colorable Induced Subgraph) on kP_2-free graphs, where a very simple algorithm can be obtained. Indeed, kP_2-free graphs have polynomially many (inclusion-wise) maximal independent sets, and these can be enumerated in polynomial time. Consequently, a maximum-weight induced bipartite subgraph can be found by exhaustively enumerating all pairs of maximal independent sets [3]. However, it is easily seen that even P_3-free graphs (i.e., graphs whose connected components are cliques) do not have polynomially many maximal independent sets. But it turns out that a weaker property is enough for our purposes: Admitting a polynomial family of "well-behaved" vertex sets such that every maximal independent set is contained in one of these sets. In our case, "well-behaved" means inducing a P_3-free subgraph. The intuition is that, given such a family, we can efficiently guess the color classes, each of which will be a disjoint union of cliques, and then match vertices to the possible color classes.

The following key notion, which we dub amiable family, was first introduced by Lozin and Mosca [15]. For a graph G, a family $\mathcal{S} \subseteq 2^{V(G)}$ of subsets of $V(G)$ is an *amiable family* if it satisfies the following two properties:

- Each member of $\mathcal{S}$ induces a P_3-free subgraph in G;
- Each (inclusion-wise) maximal independent set of G is contained in some member of $\mathcal{S}$.

Lozin and Mosca [15] showed that, when $k = 2$, every kP_3-free graph G admits an amiable family of size polynomial in $|V(G)|$ and which can be computed in polynomial time. Later, Lozin [14] observed how such property in fact holds for every $k \geq 2$ (see Lemma 8 for a formal statement). Given an amiable family $\mathcal{S}$ of polynomial size of a kP_3-free graph G, we would like to exhaustively solve Max-Weight List r-Colorable Induced Subgraph on every possible r-tuple consisting of members of $\mathcal{S}$. More precisely, let $(S_1, \ldots, S_r) \in \mathcal{S}^r$ be an r-tuple of members of $\mathcal{S}$. We would like to find a maximum-weight induced subgraph of $G[\bigcup_{i\in[r]} S_i]$ which admits an r-coloring respecting the given r-list assignment and such that, for $i = 1, \ldots, r$, all vertices colored i are contained in S_i. To do this, we then extend an idea of Chudnovsky et al. [4] as follows. We reduce our problem to that of finding a maximum-weight matching in an auxiliary bipartite graph where one partition class Y consists of $\bigcup_{i\in[r]} S_i$, the other class X consists of the connected components of the subgraphs $G[S_i]$, for $i = 1, \ldots, r$, and there is an edge between $y \in Y$ and $x \in X$ if and only if y belongs to the connected component x. Since each weighted matching problem can be solved in polynomial time using the Hungarian method (see, e.g., [17, Theorem 17.3]) and we build $|\mathcal{S}|^r$ auxiliary problems, which is a polynomial in $|V(G)|$, a solution to Max-Weight List r-Colorable Induced Subgraph can be found in polynomial time.

In order to prove Theorems 5 and 6, we consider the following distance-d generalization of the notion of amiable family. For a graph G, a family $\mathcal{S} \subseteq 2^{V(G)}$ of subsets of $V(G)$ is a *distance-d amiable family* if it satisfies the following properties:

- Each member of $\mathcal{S}$ induces a P_3-free subgraph in G;
- For each $S \in \mathcal{S}$, the connected components of $G[S]$ are pairwise at distance at least d in G;
- Each (inclusion-wise) maximal distance-d independent set of G is contained in some member of $\mathcal{S}$.

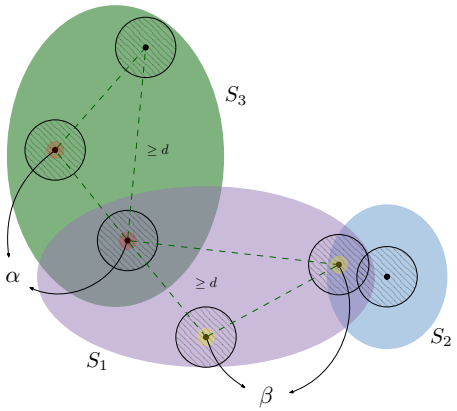


Figure 1 Visualization for distance-d amiable family $\mathcal{S} = \{S_1, S_2, S_3\}$. Circles represent cliques and α, β are maximal distance-d independent sets. Dashed lines depict paths of lengths at least d.

Clearly, a distance-2 amiable family is nothing but an amiable family. Our main technical contribution is the following Lemma 7, proven in Section 4. Although the algorithm for Lemma 7 is inspired by the case $d = 2$ and hence by the work of Lozin and Mosca [15], its proof of correctness is much more involved and requires genuinely new ideas.

▶ **Lemma 7.** *Let $d \geq 6$ be a fixed integer. For every $k \in \mathbb{N}$, every kP_3-free graph admits a distance-d amiable family of size $|V(G)|^{O(k)}$, which can be computed in time $|V(G)|^{O(k)}$.*

Equipped with Lemma 7, we immediately obtain Theorem 5. Indeed, in order to solve Max-Weight Distance-d Independent Set on a kP_3-free graph G, we simply find in polynomial time a distance-d amiable family $\mathcal{S}$ of G as above and, for each member $S \in \mathcal{S}$, find a max-weight independent set in the P_3-free graph $G[S]$. The latter can be clearly done in polynomial time, thus proving Theorem 5. The proof of Theorem 6 is similar to that of Theorem 3, the only difference being the use of a distance-d amiable family for $d \geq 6$, and hence omitted.

A remark about our approach, which combines ideas of Lozin and Mosca [14, 15] and Chudnovsky et al. [4], is in place. The proof of Chudnovsky et al. [4] that List r-Coloring ($r \geq 1$) is polynomial-time solvable on kP_3-free graphs generalizes the earlier result of Hajebi et al. [9] that List 5-Coloring is polynomial-time solvable on kP_3-free graphs, by replacing the second step in the proof of Hajebi et al. (see [9, Theorem 4.3]) with a significantly simpler argument (the aforementioned reduction to a bipartite matching problem) that, in addition, works for all $r \in \mathbb{N}$. However, their result still relies on the very technical first step of Hajebi et al. (see [9, Theorem 5.1]). Our approach can be viewed as a step further in the direction of simplifying and generalizing, as exemplified by our Theorems 3 and 6, which extend in different ways the main result of Chudnovsky et al. [4] by means of an arguably elegant and self-contained proof.

Proofs of statements marked with "$\star$" are omitted due to space constraints.

2 Preliminaries

We denote the set of positive integers by $\mathbb{N}$. For every $n \in \mathbb{N}$, we let $[n] := \{1, \ldots, n\}$. Given a set A, we denote by A^r the set of all ordered r-tuples of elements of A, i.e., $A^r = \{(a_1, \ldots, a_r) : a_i \in A \text{ for } i = 1, \ldots, r\}$.

All graphs in our paper are finite and simple. The *empty graph* is the graph with no vertices. Let now G be a graph. For $X \subseteq V(G)$, we denote the subgraph of G *induced* by X as $G[X]$, that is $G[X] = (X, \{uv : u, v \in X \text{ and } uv \in E(G)\})$. We use $N_G(X)$ to denote the neighbors in $V \setminus X$ of vertices in X. For disjoint sets $X, Y \subseteq V(G)$, we say that X is *complete* to Y if every vertex in X is adjacent to every vertex in Y, and X is *anticomplete* to Y if there are no edges between X and Y. For a subset $X \subseteq V(G)$, the *anti-neighborhood* of X, denoted by $\mathcal{A}(X)$, is the subset of vertices in $V(G) \setminus X$ which are anticomplete to X. With a slight abuse of notation, if $X = \{v_1, \ldots, v_i\}$, we denote $\mathcal{A}(X) = \mathcal{A}(\{v_1, \ldots, v_i\})$ by $\mathcal{A}(v_1, \ldots, v_i)$. Given two subsets $X, Y \subseteq V(G)$, an X, Y*-path* in G is a path in G which has one end in X, the other end in Y, and whose inner vertices belong to neither X nor Y. For vertices $u, v \in V(G)$, we denote by $d_G(u, v)$ the distance between u and v in G, i.e., the length of a shortest u, v-path in G. If no such path exists, we let $d_G(u, v) = \infty$. Moreover, for $d \in \mathbb{N}$, we let $N_G^{\geq d}(v) = \{u \in V(G) : d_G(v, u) \geq d\}$ and $N_G^{\leq d}(v) = \{u \in V(G) \setminus \{v\} : d_G(v, u) \leq d\}$. Given two subsets $X, Y \subseteq V(G)$, the *distance* between X and Y in G is the quantity $d_G(X, Y) = \min_{x \in X, y \in Y} d_G(x, y)$, i.e., the length of a shortest path in G between a vertex in X and a vertex in Y. Given $u \in V(G)$ and $Q \subseteq V(G)$, we say that u is connected to Q in G if there exists a u, Q-path in G (possibly of length 0).

A *clique* of a graph is a set of pairwise adjacent vertices and an *independent set* is a set of pairwise non-adjacent vertices. A *matching* of a graph is a set of pairwise non-adjacent edges. A *connected component* of G is a maximal connected subgraph of G. For convenience, we will often view a connected component as its vertex set rather than the subgraph itself. For this reason, we will say for example that "a connected component is a clique" rather than "a connected component is a complete subgraph". Throughout the paper, we will repeatedly make use of the fact that every connected component of a P_3-free graph is a clique.

3 The proof of Theorem 3

In this section we prove Theorem 3. As mentioned in the introduction, the first step is to obtain a polynomial-time algorithm for finding an amiable family (necessarily of polynomial size) of a kP_3-free graph. The second step consists then in reducing Max-Weight List r-Colorable Induced Subgraph to polynomially many auxiliary weighted matching problems. We do this in Lemma 9. We finally combine these two steps and prove Theorem 3.

▶ **Lemma 8** ($\star$). *For every $k \in \mathbb{N}$, every kP_3-free graph G admits an amiable family of size $|V(G)|^{O(k)}$, which can be computed in time $|V(G)|^{O(k)}$.*

▶ **Lemma 9.** *Let $r \geq 1$ and $d \geq 2$ be fixed integers. Let G be a graph with weight function $w \colon V(G) \to \mathbb{Q}_+$, L an r-list assignment of G, and $\mathcal{S}$ a distance-d amiable family of G. Given an r-tuple $(S_1, \ldots, S_r) \in \mathcal{S}^r$, there exists an $O(((r+1)|V(G)|)^3)$-time algorithm that finds a maximum-weight induced subgraph H of $G[\bigcup_{i \in [r]} S_i]$ which admits a (d, r)-coloring $\phi \colon V(H) \to [r]$ satisfying the following:*

1. *For every $v \in V(H)$, $\phi(v) \in L(v)$;*
2. *For every $i \in [r]$, $\{v \in V(H) \colon \phi(v) = i\} \subseteq S_i$.*

Proof. Consider an r-tuple $(S_1, \ldots, S_r) \in \mathcal{S}^r$. For every $i \in [r]$, let c_i be the number of connected components of $G[S_i]$ and let $S_i^1, \ldots, S_i^{c_i}$ be an arbitrary ordering of the connected components of $G[S_i]$. By definition of distance-d amiable family, each such connected component of $G[S_i]$ is a clique and any two of them are pairwise at distance at least d in G.

The first step of our algorithm consists in preprocessing the graph $G[\bigcup_{i\in[r]} S_i]$ as follows. For every $i \in [r]$, if there exists a vertex $v \in S_i$ such that $i \notin L(v)$, then we remove v from S_i, that is, we set $S_i = S_i \setminus \{v\}$. Observe that this preprocessing is safe. Indeed, if H is an induced subgraph of $G[\bigcup_{i\in[r]} S_i]$ for which there exists a (d,r)-coloring ϕ satisfying both 1 and 2, then surely $\phi(v) \neq i$ if $v \in V(H)$.

We next show that finding such an induced subgraph of $G[\bigcup_{i\in[r]} S_i]$ of maximum weight boils down to finding a maximum-weight matching in an auxiliary weighted bipartite graph B constructed as follows. The graph B has bipartition $X \cup Y$ and edge set $E(B)$, where

$$X = \left\{x_{S_i^j} : i \in [r], j \in [c_i]\right\},\ Y = \left\{y_v : v \in \bigcup_{i\in[r]} S_i\right\} \text{ and } E(B) = \bigcup_{i=1}^{r}\bigcup_{j=1}^{c_i}\left\{y_v x_{S_i^j} : v \in S_i^j\right\}.$$

Moreover, for each $v \in \bigcup_{i\in[r]} S_i$, every edge of B incident to y_v is assigned the weight $w(v)$. Note that $|V(B)| = |\bigcup_{i\in[r]} S_i| + \sum_{i\in[r]} c_i \leq |V(G)| + r|V(G)| = (r+1)|V(G)|$.

▷ Claim 10 (⋆). Let $m \in \mathbb{Q}_+$. The graph $G[\bigcup_{i\in[r]} S_i]$ has an induced subgraph H with $w(V(H)) = m$ and which admits a (d,r)-coloring satisfying both 1 and 2 if and only if B has a matching of weight m.

Now, by Claim 10, our problem reduces to finding a maximum-weight matching in the bipartite graph B, which can be done in $O(|V(B)|^3)$-time using the Hungarian method (see [17, Theorem 17.3]). Since $|V(B)| \leq (r+1)|V(G)|$, this completes the proof of Lemma 9. ◀

We can finally sketch the proof of Theorem 3.

Proof of Theorem 3. Given a kP_3-free graph G and an r-list assignment L of G, the following algorithm outputs, in polynomial time, an induced subgraph H of G admitting a coloring that respects L and such that H is of maximum weight among all such subgraphs of G.

1. Compute an amiable family $\mathcal{S}$ of G of size $|V(G)|^{O(k)}$.
2. For each r-tuple $(S_1, \ldots, S_r) \in \mathcal{S}^r$, find a maximum-weight induced subgraph H of $G[\bigcup_{i\in[r]} S_i]$ which admits a coloring $\phi\colon V(H) \to [r]$ satisfying 1 and 2 of Lemma 9.
3. Among all induced subgraphs computed in Step 2, output one of maximum weight. ◀

4 Distance-d amiable families: The proof of Lemma 7

In this section we prove Lemma 7, which is used for the proof of Theorem 6. To make our inductive proof of Lemma 7 work, we show in fact something more general, which requires the following definition. Given a graph G and a subset $F \subseteq V(G)$, a subset $S \subseteq V(G)$ is *F-avoiding* if $S \cap F = \varnothing$. By extension, a family $\mathcal{S} \subseteq 2^{V(G)}$ is *F-avoiding* if each member of $\mathcal{S}$ is F-avoiding. For a graph G, a family $\mathcal{S} \subseteq 2^{V(G)}$ of subsets of $V(G)$ is an *F-avoiding distance-d amiable family* if it satisfies the following properties:

- $\mathcal{S}$ is F-avoiding;
- Each member of $\mathcal{S}$ induces a P_3-free subgraph in G;
- For each $S \in \mathcal{S}$, the connected components of $G[S]$ are pairwise at distance at least d in G;
- Each (inclusion-wise) maximal F-avoiding distance-d independent set of G is contained in some member of $\mathcal{S}$.

Note that a distance-d amiable family of G is nothing but an F-avoiding distance-d amiable family of G for $F = \varnothing$. As we shall see, our proof of Lemma 7 in fact shows that, for every kP_3-free graph G and every $F \subseteq V(G)$, the graph G admits an F-avoiding distance-d amiable family of size $|V(G)|^{O(k)}$ and which can be computed in time $|V(G)|^{O(k)}$.

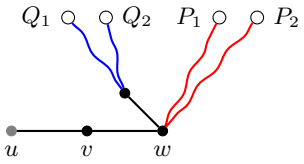


Figure 2 The case $d \leq 5$ (paths in blue are of length $d-3$, those in red, $d-2$).

Before formally proving Lemma 7, let us discuss why our approach fails for $3 \leq d \leq 5$ (recall however that the failure for $d \in \{3,5\}$ is to be expected given the hardness results in Theorem 2). The algorithm for computing a distance-d amiable family is, in essence, similar to the one for computing an amiable family (so $d=2$): it enumerates all P_3's uvw in the graph, "guesses" which vertex in the path is in the independent set and recursively calls the algorithm on (roughly) the anti-neighborhood of $\{u,v,w\}$, which is $(k-1)P_3$-free. The main difference (and difficulty) is in ensuring that the family computed recursively satisfies the distance requirement: the connected components of each member in this family could in principle be closer in the original graph. Consider, for instance, the case where u is picked in the independent set, and there are four connected components Q_1, Q_2, P_1, P_2 in a member of the recursively computed family with shortest paths between Q_1 and Q_2, and P_1 and P_2 as shown in Figure 2. Then, for $d \leq 5$, Q_1 and Q_2 can end up at a distance less than d, while P_1 and P_2 are at a distance at least d. Since there is a priori no way of "distinguishing" these two cases, we have to take d "large enough". The precise argument appears in the proof of Lemma 7 (Claim 12). We finally restate Lemma 7.

▶ **Lemma 7.** *Let $d \geq 6$ be a fixed integer. For every $k \in \mathbb{N}$, every kP_3-free graph admits a distance-d amiable family of size $|V(G)|^{O(k)}$, which can be computed in time $|V(G)|^{O(k)}$.*

Proof of Lemma 7. To prove the lemma, we provide, for every $d \geq 6$ and every $k \geq 1$, an algorithm Λ_k^d which takes as input a kP_3-free graph G, together with an arbitrary ordering of $V(G)$, and a subset $F \subseteq V(G)$, and outputs an F-avoiding distance-d amiable family $\Lambda_k^d(G,F)$ of G. The pseudo-code of the algorithm is given in Algorithm 1. Note that, if the input graph G is the empty graph, Λ_k^d correctly outputs $\Lambda_k^d(G) = \{\varnothing\}$.

In the following, we prove three claims (Claims 11–13) which altogether show that, for every $d \geq 6$ and every $k \geq 1$, if G is a kP_3-free graph and $F \subseteq V(G)$, then the family $\Lambda_k^d(G,F)$ is indeed an F-avoiding distance-d amiable family. We will then show that $\Lambda_k^d(G,F)$ has size $|V(G)|^{O(k)}$ and that the algorithm Λ_k^d has running time $|V(G)|^{O(k)}$. Taking $F = \varnothing$, will prove the lemma.

Given a graph G on n vertices and an ordering $v_1, \ldots, v_n$ of $V(G)$, we let $G_i = G[\{v_1, \ldots, v_i\}]$. Moreover, an induced path in G with vertex set $\{x_1, x_2, \ldots, x_\ell\}$ and edge set $\{x_1x_2, x_2x_3, \ldots x_{\ell-1}x_\ell\}$ is denoted by listing its vertices in the natural order $x_1x_2 \cdots x_\ell$.

▷ Claim 11 (⋆). For every $d \geq 6$ and every $k \geq 1$, if G is a kP_3-free graph and $F \subseteq V(G)$, then $\Lambda_k^d(G,F)$ is F-avoiding.

▷ Claim 12. For every $d \geq 6$ and $k \geq 1$, if G is a kP_3-free graph, $F \subseteq V(G)$, and $S \in \Lambda_k^d(G,F)$, then $G[S]$ is P_3-free and its connected components are pairwise at distance at least d in G.

Proof of Claim 12. For fixed $d \geq 6$, we proceed by induction on k. The statement holds for $k=1$, since for every P_3-free graph G and every $F \subseteq V(G)$, $\Lambda_1^d(G) = \{V(G) \setminus F\}$. Suppose now that $k > 1$ and that the statement holds for $k-1$. Let G be an n-vertex kP_3-free

Algorithm 1 Λ_k^d.

Input: A kP_3-free graph G, an arbitrary ordering $v_1, \ldots, v_n$ of $V(G)$, and a subset $F \subseteq V(G)$.
Output: An F-avoiding distance-d amiable family of G.

1: Initialize $\mathcal{S} = \{\varnothing\}$.
2: **for** $i = 1, \ldots, n$ **do**
3: **if** $v_i \notin F$ **then**
4: **for** every member $S \in \mathcal{S}$ **do**
5: **if** $G[S \cup \{v_i\}]$ is P_3-free and its connected components are pairwise at distance at least d in G **then**
6: Set $S = S \cup \{v_i\}$.
7: **for** every induced P_3 uvw in G_i such that $u \notin F$ **do**
8: Compute $\mathcal{C} := \Lambda_{k-1}^d(G[N_G^{\geq 4}(u)], (F \cap N_G^{\geq 4}(u)) \cup (N_G^{\geq 4}(u) \cap N_G^{\leq d-1}(u)))$.
9: **for** every $C \in \mathcal{C}$ **do**
10: Set $\mathcal{S} = \mathcal{S} \cup \{C \cup \{u\}\}$.
11: **for** every induced P_3 uvw in G_i such that $v \notin F$ **do**
12: Compute $\mathcal{C} := \Lambda_{k-1}^d(G[N_G^{\geq 4}(v)], (F \cap N_G^{\geq 4}(v)) \cup (N_G^{\geq 4}(v) \cap N_G^{\leq d-1}(v)))$.
13: **for** every $C \in \mathcal{C}$ **do**
14: Set $\mathcal{S} = \mathcal{S} \cup \{C \cup \{v\}\}$.
15: **return** $\Lambda_k^d(G, F) := \mathcal{S}$.

graph and let $F \subseteq V(G)$ as in input. For every $i \in [n]$, let $\mathcal{S}^i$ be the state of the family $\mathcal{S}$ at the end of the i-th iteration of the main loop. We show by induction on i that each member of $\mathcal{S}^i$ induces a P_3-free subgraph of G whose connected components are pairwise at distance at least d in G. The case $i = 1$ trivially holds, since either $v_1 \in F$ in which case $\mathcal{S}^1 = \{\varnothing\}$, or $v_1 \notin F$ in which case $\mathcal{S}^1 = \{\{v_1\}\}$. Thus, suppose that $i > 1$ and that the statement holds for $i - 1$.

Consider $S \in \mathcal{S}^i$. If $S \in \mathcal{S}^{i-1}$ then, by the induction hypothesis, $G[S]$ is P_3-free and its connected components are pairwise at distance at least d in G. Thus, we may assume that $S \notin \mathcal{S}^{i-1}$. This implies that S is added to $\mathcal{S}^i$ in one of the three inner loops during the i-th iteration of the main loop. If S is added in line 6 as an extension of a member of $\mathcal{S}^{i-1}$, then the statement holds by construction. Suppose next that S is added to $\mathcal{S}^i$ in line 10. Hence, there exist an induced P_3 uvw in G_i such that $u \notin F$ and a set $C \in \Lambda_{k-1}^d(G[N_G^{\geq 4}(u)], (F \cap N_G^{\geq 4}(u)) \cup (N_G^{\geq 4}(u) \cap N_G^{\leq d-1}(u)))$ such that $S = C \cup \{u\}$. Observe now that since G is kP_3-free and $(N_G(v) \cup N_G(w)) \cap N_G^{\geq 4}(u) = \varnothing$, the graph $G[N_G^{\geq 4}(u)]$ is $(k-1)P_3$-free, and thus, by the induction hypothesis on $k - 1$, every member of $\Lambda_{k-1}^d(G[N_G^{\geq 4}(u)], (F \cap N_G^{\geq 4}(u)) \cup (N_G^{\geq 4}(u) \cap N_G^{\leq d-1}(u)))$ induces a P_3-free subgraph of G whose connected components are pairwise at distance at least d in $G[N_G^{\geq 4}(u)]$. We claim that then, the connected components of $G[C]$ are in fact pairwise at distance at least d in G. Suppose, to the contrary, that there exist two connected components Q_1 and Q_2 of $G[C]$ such that $d_G(Q_1, Q_2) \leq d - 1$. Since the distance in $G[N_G^{\geq 4}(u)]$ between Q_1 and Q_2 is at least d, every shortest path in G from Q_1 to Q_2 contains at least one vertex of $V(G) \setminus N_G^{\geq 4}(u)$; let P be such a shortest path and let $x \in V(G) \setminus N_G^{\geq 4}(u)$ be an arbitrary vertex on P. Clearly, $d_G(u, x) \leq 3$. Now, by Claim 11, $C \cap ((F \cap N_G^{\geq 4}(u)) \cup (N_G^{\geq 4}(u) \cap N_G^{\leq d-1}(u))) = \varnothing$ which implies, in particular, that $C \subseteq N_G^{\geq d}(u)$. It follows that for $j \in [2]$,

$$d \leq d_G(u, Q_j) \leq d_G(u, x) + d_G(x, Q_j) \leq 3 + d_G(x, Q_j) \text{ and thus,}$$
$$d_G(Q_1, Q_2) = d_G(Q_1, x) + d_G(x, Q_2) \geq 2(d - 3).$$

Since, by assumption, $d_G(Q_1, Q_2) \leq d-1$, we conclude that $2(d-3) \leq d-1$, that is, $d \leq 5$, a contradiction. Thus, the connected components of $G[C]$ are pairwise at distance at least d in G. Since $C \subseteq N_G^{\geq d}(u)$, as previously observed, and $G[C]$ is P_3-free, it follows that $S = C \cup \{u\}$ induces a P_3-free graph whose connected components are pairwise at distance at least d in G. We conclude similarly in the case where S is added to $\mathcal{S}^i$ in line 14. ◁

▷ Claim 13. For every $d \geq 6$ and $k \geq 1$, if G is a kP_3-free graph and $F \subseteq V(G)$, then every F-avoiding distance-d independent set of G is contained in a member of $\Lambda_k^d(G, F)$.

Proof of Claim 13. For fixed $d \geq 6$, we proceed by induction on k. The statement clearly holds for $k = 1$, since for every P_3-free graph G and every $F \subseteq V(G)$, $\Lambda_1^d(G, F) = \{V(G) \setminus F\}$. Suppose now that $k > 1$ and that the statement holds for $k-1$. Let G be a kP_3-free graph and let $F \subseteq V(G)$ as in input. For every $i \in [n]$, let $\mathcal{S}^i$ be the state of the family $\mathcal{S}$ at the end of the i-th iteration of the main loop. Furthermore, given a distance-d independent set I of G such that $I \subseteq V(G_i)$, we say that I is *G_i-compatible* if for every $u \in I$ and every connected component Q of G_i such that u is connected to Q in G, there exists a shortest u, Q-path $P_{u,Q}$ in G such that $N_G^{\leq 2}(u) \cap V(P_{u,Q}) \subseteq V(G_i)$. We now show, by induction on i, that for every G_i-compatible F-avoiding distance-d independent set I of G such that $I \subseteq V(G_i)$, there exists $S \in \mathcal{S}^i$ such that $I \subseteq S$. Note that this is enough to prove our statement for k, since any F-avoiding distance-d independent set of $G = G_n$ is surely G_n-compatible.

The base case $i = 1$ trivially holds, since either $v_1 \in F$ and $\mathcal{S}^1 = \{\varnothing\}$ in which case $\varnothing$ is the only G_1-compatible F-avoiding distance-d independent of G_1, or $v_1 \notin F$ and $\mathcal{S}^1 = \{\{v_1\}\}$ in which case $\{v_1\}$ is the only maximal G_1-compatible F-avoiding distance-d independent set of G_1. Thus, suppose that $i > 1$ and that the statement holds for $i-1$.

Consider a G_i-compatible F-avoiding distance-d independent set I of G such that $I \subseteq V(G_i)$. If $I \subseteq V(G_{i-1})$ and I is G_{i-1}-compatible then, by the induction hypothesis, there exists $S \in \mathcal{S}^{i-1}$ such that $I \subseteq S$. Observe now that, by construction, there then exists $S' \in \mathcal{S}^i$ such that $S \subseteq S'$ and hence $I \subseteq S'$. Thus, assume that this does not hold. This implies that either $I \not\subseteq V(G_{i-1})$, or $I \subseteq V(G_{i-1})$ but I is not G_{i-1}-compatible. We distinguish these two cases and argue that we can always find $S \in \mathcal{S}^i$ such that $I \subseteq S$.

Case 1. $I \not\subseteq V(G_{i-1})$.
Hence, $v_i \in I$. Since I is F-avoiding, $v_i \notin F$. We claim that $I \setminus \{v_i\}$ is G_{i-1}-compatible. Indeed, since I is G_i-compatible, for every $u \in I \setminus \{v_i\}$ and every connected component Q of G_i such that u is connected to Q in G, there exists a shortest u, Q-path $P_{u,Q}$ in G such that $N_G^{\leq 2}(u) \cap V(P_{u,Q}) \subseteq V(G_i)$. But v_i is at distance at least $d \geq 6$ from u in G and so $v_i \notin (N_G^{\leq 2}(u) \cap V(P_{u,Q}))$, that is, $N_G^{\leq 2}(u) \cap V(P_{u,Q}) \subseteq V(G_{i-1})$ which proves our claim. By the induction hypothesis, there exists $S \in \mathcal{S}^{i-1}$ such that $I \setminus \{v_i\} \subseteq S$. If $G[S \cup \{v_i\}]$ is P_3-free and its connected components are pairwise at distance at least d in G, then $S \cup \{v_i\} \in \mathcal{S}^i$ by construction (cf. line 6) and $I \subseteq S \cup \{v_i\}$. Thus, we may assume that this does not hold. Hence, either $G[S \cup \{v_i\}]$ contains at least one induced P_3, or $G[S \cup \{v_i\}]$ is P_3-free but at least two of its connected components are at distance strictly less than d in G.

Case 1.1. $G[S \cup \{v_i\}]$ contains at least one induced P_3.
By Claim 12, $G[S]$ is P_3-free and thus, any induced P_3 in $G[S \cup \{v_i\}]$ must contain v_i. We distinguish two cases. Suppose first that there exist a connected component Q of $G[S]$ and a vertex $u \in Q$ such that v_i is adjacent to u but not complete to Q, say $w \in Q$ is nonadjacent to v_i. Note that at some point during the i-th iteration of the main loop, the induced P_3 v_iuw is considered in the second inner loop, since $v_i \notin F$. Furthermore, since G is kP_3-free and $(N_G(u) \cup N_G(w)) \cap N_G^{\geq 4}(v_i) = \varnothing$, the graph $G[N_G^{\geq 4}(v_i)]$ is $(k-1)P_3$-free. Thus, by the induction hypothesis on $k-1$ and since $I \setminus \{v_i\} \subseteq N_G^{\geq d}(v_i) \setminus F$, there exists

$C \in \Lambda^d_{k-1}(G[N^{\geq 4}_G(v_i)], (F \cap N^{\geq 4}_G(v_i)) \cup (N^{\geq 4}_G(v_i) \cap N^{\leq d-1}_G(v_i)))$ such that $I \setminus \{v_i\} \subseteq C$. But then, $C \cup \{v_i\} \in \mathcal{S}^i$ and $I \subseteq C \cup \{v_i\}$. The case where, for every connected component Q of $G[S]$ the vertex v_i is either complete or anticomplete to Q, is handled similarly.

Case 1.2. $G[S \cup \{v_i\}]$ is P_3-free but at least two of its connected components are at distance strictly less than d in G.

By Claim 12, the connected components of $G[S]$ are pairwise at distance at least d in G and so there must exist a connected component Q of $G[S \cup \{v_i\}]$ at distance strictly less than d in G to the connected component of $G[S \cup \{v_i\}]$ containing v_i. Now, since v_i is connected to Q in G and $v_i \in I$, the assumption that I is G_i-compatible implies that there exists a shortest v_i, Q-path $P_{v_i,Q}$ in G such that $N^{\leq 2}_G(v_i) \cap V(P_{v_i,Q}) \subseteq V(G_i)$. Let $u_1, u_2 \in V(G_i)$ be the two vertices on $P_{v_i,Q}$ at distance one and two, respectively, from v_i. Note that at some point during the i-th iteration of the main loop, the induced P_3 $v_iu_1u_2$ is considered in the second inner loop, since $v_i \notin F$. Moreover, since G is kP_3-free and $(N_G(u_1) \cup N_G(u_2)) \cap N^{\geq 4}_G(v_i) = \varnothing$, the graph $G[N^{\geq 4}_G(v_i)]$ is $(k-1)P_3$-free. Thus, by the induction hypothesis on $k-1$ and since $I \subseteq N^{\geq d}_G(v_i) \setminus F$, there exists $C \in \Lambda^d_{k-1}(G[N^{\geq 4}_G(v_i)], (F \cap N^{\geq 4}_G(v_i)) \cup (N^{\geq 4}_G(v_i) \cap N^{\leq d-1}_G(v_i)))$ such that $I \setminus \{v_i\} \subseteq C$. But then, $C \cup \{v_i\} \in \mathcal{S}^i$ and $I \subseteq C \cup \{v_i\}$.

Case 2. $I \subseteq V(G_{i-1})$ but I is not G_{i-1}-compatible.

Hence, there must exist $u \in I$ and a connected component Q of G_i to which u is connected in G such that $v_i \in N^{\leq 2}_G(u) \cap V(P_{u,Q})$, where $P_{u,Q}$ is a shortest path in G from u to Q given by the G_i-compatibility of I. Let $u_1, u_2 \in V(G_i)$ be the vertices on $P_{u,Q}$ at distance one and two, respectively, from u. Note that at some point during the i-th iteration of the main loop, the induced P_3 uu_1u_2 is considered in the second inner loop, since $v_i \notin F$. Moreover, since G is kP_3-free and $(N_G(u_1) \cup N_G(u_2)) \cap N^{\geq 4}_G(u) = \varnothing$, the graph $G[N^{\geq 4}_G(u)]$ is $(k-1)P_3$-free. Thus, by the induction hypothesis on $k-1$ and since $I \subseteq N^{\geq d}_G(u) \setminus F$, there exists $C \in \Lambda^d_{k-1}(G[N^{\geq 4}_G(u)], (F \cap N^{\geq 4}_G(u)) \cup (N^{\geq 4}_G(u) \cap N^{\leq d-1}_G(u)))$ such that $I \setminus \{u\} \subseteq C$. But then, $C \cup \{u\} \in \mathcal{S}^i$ and $I \subseteq C \cup \{u\}$.

Since in all cases we found $S \in \mathcal{S}^i$ with $I \subseteq S$, this concludes the proof of Claim 13. ◁

It follows now from Claims 11–13 that, for every $d \geq 6$ and every $k \geq 1$, if G is a kP_3-free graph and $F \subseteq V(G)$, then the family $\Lambda^d_k(G, F)$ is indeed an F-avoiding distance-d amiable family. It remains to show that $\Lambda^d_k(G, F)$ has size at most $|V(G)|^{O(k)}$ and that the running time of the algorithm Λ^d_k is $|V(G)|^{O(k)}$. To this end, let

$$f_d(n, k) = \max\{|\Lambda^d_k(G, F)| : |V(G)| \leq n,\ F \subseteq V(G) \text{ and } G \text{ is } kP_3\text{-free}\}.$$

Clearly, $f_d(n, 1) = 1$ for every $n \in \mathbb{N}$. We claim that, for every $n \in \mathbb{N}$ and $k > 1$, $f_d(n, k) \leq 2n^4 \cdot f_d(n, k-1)$. Indeed, for every n-vertex kP_3-free graph G and every $F \subseteq V(G)$, a member of $\Lambda^d_k(G, F)$ can only be created during an i-th iteration of the main loop (for some $i \in [n]$) in one of the inner loops from an induced P_3 of G_i and some set resulting from a call to Λ^d_{k-1}. Since for each $i \in [n]$ there are at most i^3 such copies of P_3 in G_i, at most $2i^3 \cdot f_d(n, k-1)$ new members are added in the i-th iteration of the main loop. It follows that $f_d(n, k) \leq \sum_{i \in [n]} 2i^3 \cdot f_d(n, k-1) \leq 2n^4 \cdot f_d(n, k-1)$ and thus $f_d(n, k) \leq 2^{k-1} n^{4(k-1)}$.

Similarly, for every $d \geq 6$ and every $n, k \in \mathbb{N}$, if $T_d(n, k)$ denotes the running time of the algorithm Λ^d_k on an n-vertex kP_3-free graph, then clearly $T_d(n, 1) = O(n)$. Furthermore, we obtain the following recurrence for $T_d(n, k)$, where we use the fact that checking if an n-vertex graph is P_3-free can be done in $O(n^3)$ time and determining if the connected components of an n-vertex graph are pairwise at distance at least d can be done in $O(n^3)$ time (using an all-pair-shortest-path algorithm):

$$T_d(n, k) \leq cn \cdot (f_d(n, k) \cdot n^3 + 2n^3 \cdot (T_d(n, k-1) + f_d(n, k-1))) \leq 2cn^4 \cdot T_d(n, k-1) + O(n^{4k}),$$

for some constant $c > 0$. We conclude that $T_d(n, k) \leq n^{O(k)}$. ◀

References

1 Akanksha Agrawal, Paloma T Lima, Daniel Lokshtanov, Paweł Rzążewski, Saket Saurabh, and Roohani Sharma. Odd Cycle Transversal on P_5-free graphs in polynomial time. *ACM Transactions on Algorithms*, 21(2):16:1–16:14, 2025.

2 Gábor Bacsó, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Zsolt Tuza, and Erik Jan van Leeuwen. Subexponential-time algorithms for Maximum Independent Set in P_t-free and broom-free graphs. *Algorithmica*, 81(2):421–438, 2019.

3 Nina Chiarelli, Tatiana R Hartinger, Matthew Johnson, Martin Milanič, and Daniël Paulusma. Minimum connected transversals in graphs: new hardness results and tractable cases using the price of connectivity. *Theoretical Computer Science*, 705:75–83, 2018. doi:10.1016/J.TCS.2017.09.033.

4 Maria Chudnovsky, Sepehr Hajebi, and Sophie Spirkl. List-k-Coloring H-free graphs for all $k > 4$. *Combinatorica*, 44(5):1063–1068, 2024.

5 Maria Chudnovsky, Rose McCarty, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Sparse induced subgraphs in P_6-free graphs. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 5291–5299. SIAM, 2024. doi:10.1137/1.9781611977912.190.

6 Jean-François Couturier, Petr A Golovach, Dieter Kratsch, and Daniël Paulusma. List coloring in the absence of a linear forest. *Algorithmica*, 71:21–35, 2015. doi:10.1007/S00453-013-9777-0.

7 Konrad K Dabrowski, Carl Feghali, Matthew Johnson, Giacomo Paesani, Daniël Paulusma, and Paweł Rzążewski. On cycle transversals and their connected variants in the absence of a small linear forest. *Algorithmica*, 82:2841–2866, 2020. doi:10.1007/S00453-020-00706-6.

8 Hiroshi Eto, Fengrui Guo, and Eiji Miyano. Distance-d independent set problems for bipartite and chordal graphs. *Journal of Combinatorial Optimization*, 27(1):88–99, 2014.

9 Sepehr Hajebi, Yanjia Li, and Sophie Spirkl. Complexity dichotomy for List-5-Coloring with a forbidden induced subgraph. *SIAM Journal on Discrete Mathematics*, 36(3):2004–2027, 2022. doi:10.1137/21M1443352.

10 Cicely Henderson, Evelyne Smith-Roberge, Sophie Spirkl, and Rebecca Whitman. Maximum k-colourable induced subgraphs in $(P_5 + rK_1)$-free graphs. *arXiv*, 2024. arXiv:2410.08077.

11 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103. Springer US, 1972. doi:10.1007/978-1-4684-2001-2_9.

12 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

13 Daniel Lokshtanov, Paweł Rzążewski, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Maximum Partial List H-Coloring on P_5-free graphs in polynomial time. *arXiv*, 2024. arXiv:2410.21569.

14 Vadim V. Lozin. From matchings to independent sets. *Discrete Applied Mathematics*, 231:4–14, 2017. doi:10.1016/J.DAM.2016.04.012.

15 Vadim V. Lozin and Raffaele Mosca. Maximum regular induced subgraphs in $2P_3$-free graphs. *Theoretical Computer Science*, 460:26–33, 2012.

16 Giacomo Paesani, Daniël Paulusma, and Paweł Rzążewski. Feedback Vertex Set and Even Cycle Transversal for H-free graphs: Finding large block graphs. *SIAM Journal on Discrete Mathematics*, 36(4):2453–2472, 2022. doi:10.1137/22M1468864.

17 Alexander Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2004.

18 Alexa Sharp. Distance coloring. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Algorithms – ESA 2007, 15th Annual European Symposium*, volume 4698 of *Lecture Notes in Computer Science*, pages 510–521. Springer, 2007. doi:10.1007/978-3-540-75520-3_46.

Min-Max Correlation Clustering via Neighborhood Similarity

Nairen Cao
New York University, NY, USA

Steven Roche
Boston College, Chestnut Hill, MA, USA

Hsin-Hao Su
Boston College, Chestnut Hill, MA, USA

Abstract

We present an efficient algorithm for the min-max correlation clustering problem. The input is a complete graph where edges are labeled as either positive $(+)$ or negative $(-)$, and the objective is to find a clustering that minimizes the ℓ_∞-norm of the disagreement vector over all vertices.

We address this problem with an efficient $(3+\epsilon)$-approximation algorithm that runs in nearly linear time, $\tilde{O}(|E^+|)$, where $|E^+|$ denotes the number of positive edges. This improves upon the previous best-known approximation guarantee of 4 by Heidrich, Irmai, and Andres [37], whose algorithm runs in $O(|V|^2 + |V|D^2)$ time, where $|V|$ is the number of nodes and D is the maximum degree in the graph (V, E^+).

Furthermore, we extend our algorithm to the massively parallel computation (MPC) model and the semi-streaming model. In the MPC model, our algorithm runs on machines with memory sublinear in the number of nodes and takes $O(1)$ rounds. In the streaming model, our algorithm requires only $\tilde{O}(|V|)$ space, where $|V|$ is the number of vertices in the graph.

Our algorithms are purely combinatorial. They are based on a novel structural observation about the optimal min-max instance, which enables the construction of a $(3+\epsilon)$-approximation algorithm using $O(|E^+|)$ neighborhood similarity queries. By leveraging random projection, we further show these queries can be computed in nearly linear time.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms; Theory of computation → Approximation algorithms analysis

Keywords and phrases Min Max Correlation Clustering, Approximate algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.41

Related Version *Full Version*: `https://arxiv.org/abs/2502.12519` [17]

Funding Supported by NSF CCF-2008422.

1 Introduction

In the correlation clustering problem, we are given a complete graph where each edge is labeled as either "+" or "-". A "+" edge indicates that the two vertices are *similar*, while a "-" edge indicates they are *dissimilar*. For any partition of the graph, an edge is considered to be in disagreement if it is a negative edge and its endpoints belong to the same cluster, or if it is a positive edge and its endpoints belong to different clusters. It is typical to assume that the input graph is $G^+ = (V, E^+)$, where E^+ is the set of positive edges, and to obtain time bounds in terms of properties of G^+, such as $m = |E^+|$ and D, the maximum degree of G^+. This is because, in many practical applications, the number of positive edges is often much smaller than the number of negative edges [23].


© Nairen Cao, Steven Roche, and Hsin-Hao Su;
licensed under Creative Commons License CC-BY 4.0



Given a clustering (partition), the disagreement $\rho(v)$ for any node v is defined as the number of edges incident to v that are in disagreement with respect to the clustering. The goal of the correlation clustering problem is to find a clustering that minimizes an objective function capturing the disagreement of edges.

Puleo and Milenkovic [41] introduced the objective of minimizing the ℓ_p-norm of the disagreements over the vertices, which is defined as $\left(\sum_{v\in V}\rho(v)^p\right)^{1/p}$. This objective generalizes the correlation clustering problem proposed by Bansal, Blum, and Chawla [8], which corresponds to the case $p = 1$, where the goal is to minimize the total number of disagreements. Significant progress has been made on the ℓ_1-norm objective [21, 3, 22, 27, 26], Cao, Cohen-Addad, Lee, Li, Newman, and Vogl [14] present a 1.437-approximate algorithm for ℓ_1-norm objective. For the case $p = \infty$, it corresponds to the min-max correlation clustering problem, where the goal is to minimize the maximum disagreements over all vertices. While the $p = 1$ case captures scenarios in which minimizing overall disagreements is desired, the $p = \infty$ case caters to situations where the quality of every individual needs to be ensured – for example, in community detection problems where no "antagonists" are allowed, with an antagonist referring to an individual whose properties are inconsistent with those of too many other members.

Compared to the $p = 1$ case where significant progress has been made, the $p = \infty$ case remains relatively unexplored. Puleo and Milenkovic [41] proposed an algorithm that achieves a 48-approximation ratio. Their approach uses the standard metric linear programming formulation followed by a rounding algorithm. Later, Charikar, Gupta, and Schwartz [20] improved this result to a 7-approximation using the same framework, and Kalhan, Makarychev, and Zhou [39] further reduced it to a 5-approximation. Davies, Moseley, and Newman [29] designed a combinatorial algorithm that achieves a 40-approximation ratio with a runtime of $O(n^2 \log n)$, where $n = |V|$ is the number of vertices in the graph. The best known approximation ratio to date is 4, achieved by Heidrich, Irmai, and Andres [37], who also used a combinatorial approach with a runtime of $O(n^2 + nD^2)$, where D is the maximum degree of the G^+.

Concerning efficiency, the ultimate goal of an efficient algorithm is to achieve a running time that is nearly linear in $m = |E^+|$. However, none of the aforementioned algorithms has yet achieved such a goal. Recently, Cao, Li, and Ye [16] proposed an nearly linear-time algorithm that achieves a 63.3-approximation ratio. While their algorithm is fast, the approximation ratio is far from optimal. This naturally leads to the question:

> Can we design a nearly linear algorithm for min-max correlation clustering with a small approximation ratio?

We give a nearly linear algorithm for the problem that achieves a $(3+\epsilon)$-approximation. Moreover, it can be made exactly 3-factor approximation with additional running time.

▶ **Theorem 1.** *Let $G = (V, E^+)$ be a min-max correlation clustering instance, $\epsilon > 0$ be a small constant, and* OPT *be the value of the optimal solution. There exist:*

1. *A randomized $O(m \log^2 n/\epsilon^2)$-time algorithm that outputs a clustering $\mathcal{C}$ with* $\mathrm{obj}(\mathcal{C}) \le (3+\epsilon)\cdot \mathrm{OPT}$ *w.h.p.*[1]
2. *A deterministic $O(mD \log n)$-time sequential algorithm that outputs a clustering $\mathcal{C}$ with* $\mathrm{obj}(\mathcal{C}) \le 3 \cdot \mathrm{OPT}$*, where D is the maximum degree of $G^+ = (V, E^+)$.*

[1] *With high probability*, which refers to with probability at least $1 - 1/n^c$ for a sufficiently large constant c.

1.1 Semi-Streaming and MPC Settings

For large graphs, it can be infeasible to store the graph G entirely on a single machine and then compute the solution. A substantial body of research has focused on massively parallel computation (MPC) algorithms for the $p = 1$ case [10, 40, 31, 11, 25, 7, 12, 15, 28]. Very recently, Cao, Cohen-Addad, Lee, Li, Lolck, Newman, Thorup, Vogl, Yan, and Zhang [13] gave an $O(1)$-round MPC algorithm that achieves a $(1.437 + \epsilon)$-approximation ratio.

Moreover, several constant-round semi-streaming algorithms with $n \operatorname{polylog} n$ space complexity have been developed [23, 2, 25, 9]. The first single-pass semi-streaming algorithm was given by Ahn, Cormode, Guha, McGregor, and Wirth [2], and it was later improved by multiple works [7, 9, 18, 28]. Very recently, a breakthrough result of Assadi, Khanna, and Putterman [5] showed that it is possible to obtain a $(\alpha_{\text{best}} + o(1))$-approximation in a single pass using $n \operatorname{polylog} n$ space, where α_{best} denotes the best approximation ratio achievable by polynomial-time sequential algorithms. Currently, it is known that $\alpha_{\text{best}} \leq 1.437$ [14].

In contrast, less work in such directions have been done for the $p = \infty$ setting. For the MPC setting, the only known work is by Cao, Ye, and Li [16], who proposed an algorithm that achieves a 63.3-approximation ratio in $O(\log^3 n)$ rounds and another algorithm that achieves a 360-approximation ratio in $O(1)$ rounds. For the semi-streaming setting, to our knowledge, we are not aware of any existing work on semi-streaming algorithms. Most of the aforementioned streaming algorithms for $p = 1$ were specifically tailored for this case, and the sparsification technique of [5] does not seem to generalize obviously to other values of p.

In addition to our sequential algorithm, we give a constant-round MPC algorithm and a single-pass semi-streaming algorithm that achieve a $(3 + \epsilon)$-approximation for the problem:

▶ **Theorem 2.** *Let $G = (V, E^+)$ be a min-max correlation clustering instance, $\epsilon > 0$ be a small constant, and* OPT *be the value of the optimal solution. In the following models, there exist randomized algorithms that output a clustering $\mathcal{C}$ with* $\text{obj}(\mathcal{C}) \leq (3 + \epsilon) \cdot \text{OPT}$ *w.h.p.:*

1. *(MPC model) An $O(1)$-round algorithm using $O(n^\delta)$ memory per machine and total memory $O(m \log n / \epsilon^2)$ for any constant $0 < \delta < 1$.*
2. *(Semi-streaming model) A single-pass streaming algorithm that uses $O(n \log n / \epsilon^2)$ space.*

1.2 Technical Overview

Better Approximation. Our first main technical contribution is a newly achieved approximation factor of 3. Given a guess for the optimal objective value ϕ, if $OPT \leq \phi$, [37] observed that if the neighborhoods of u and v share at least 2ϕ elements, then they must belong to the same cluster in the optimal solution. Similarly, if neighborhoods of u and v differ by more than 2ϕ elements, then they must belong to different clusters in the optimal solution.

Furthermore, they observed that these properties can be used to determine the clusters for vertices of degrees at least 4ϕ. Specifically, if $d(x) \geq 4\phi$, then for every other vertex y, either $|N[x] \cap N[y]| \geq 2\phi$ or $|N[x] \Delta N[y]| \geq 2\phi$. Here, $N[x] = N(x) \cup \{x\}$ represents the closed neighborhood of vertex x. The remaining vertices can then be placed in singleton clusters, as the disagreements per vertex will be upper bounded by their degrees, 4ϕ. Therefore, a clustering of disagreements upper bounded by 4ϕ can be constructed, resulting in a 4-approximation algorithm.

To achieve a 3-approximation, we first observe that if two vertices x and y have degrees greater than 3ϕ, then it is also the case either $|N[x] \cap N[y]| \geq 2\phi$ or $|N[x] \Delta N[y]| \geq 2\phi$ holds. In other words, whether x and y belong to the same cluster is uniquely determined in the optimal solution. Therefore, the clustering induced on the high-degree vertices (vertices with $d > 3\phi$) is uniquely determined.

Now the question lies in the placement of the low-degree vertices, that is, vertices with degrees upper-bounded by 3ϕ. It is unclear whether they should be placed in singleton clusters, as it is possible that they need to be included in the same cluster with certain high-degree vertices. Otherwise, the disagreements associated with the high-degree vertices could become too large.

We show that the low-degree vertices can be placed in the high-degree clusters to achieve a maximum disagreements of 3ϕ, provided $OPT \leq \phi$. A key structural result we show is that if a low-degree vertex w belongs to some cluster C in an optimal solution, then no vertex v outside C can have similar neighborhood with w, i.e., $|N[v] \Delta N[w]| \leq 2\phi$. Using this structural result, we show the following algorithm constructs a clustering with maximum disagreement upper bounded by 3ϕ (presented slightly differently here than in the main body for the sake of intuition):

1. Form clusters for high-degree vertices based on whether $|N[u] \Delta N[v]| \leq 2\phi$ for all high-degree vertices u, v (abort if any inconsistency occurs). 2. Choose an arbitrary vertex u in each cluster and have it send proposal messages to low-degree neighboring vertices whose neighborhoods are similar to u. 3. For each low-degree vertex that receives at least one proposal, pick one arbitrary proposal and join the cluster containing the vertex that sent it. 4. Place all low-degree vertices that do not receive a proposal into singleton clusters.

Efficient Implementations. The remaining question is how such an algorithm can be implemented efficiently, particularly in time (and total memory) nearly linear in $|E^+|$. A main technical challenge lies in Step 1. To implement it within the aforementioned time bound, we can only afford to conduct similarity tests (i.e. to test whether $|N[u] \Delta N[v]| \leq 2\phi$) for $O(|E^+|)$ times. However, it is possible for two vertices that are endpoints of a negative edge to have similar neighborhoods and thus need to be placed in the same cluster to achieve a good clustering.

Using the structural result, we further show that two high-degree vertices u and v are in the same cluster in the optimal solution if and only if there are at least $\phi + 1$ disjoint paths of length 2 connecting u and v in E_{sim}, where $E_{\text{sim}} \subseteq E^+$ consists of all the edges in E^+ whose endpoints have similar neighborhoods. This property enables us to develop efficient algorithms for the sublinear MPC model and the sequential model.

The remaining question lies in how to find E_{sim} efficiently. To this end, for each vertex u, we treat its neighborhood set $N[u]$ as a point in an n-dimensional space. Then, we apply the (discrete) random projection technique [1] developed for the Johnson-Lindenstrauss transform [38] to reduce the dimension to $O(\log n/\epsilon^2)$ while preserving the ℓ_2 distance (up to a $(1 \pm \epsilon)$ factor) between the points. For 0/1 vectors, the square of the ℓ_2 distance is exactly the symmetric difference. Since the dimension is $O(\log n/\epsilon^2)$, it takes $O(\log n/\epsilon^2)$ time to compute the difference. To our knowledge, this is the first time that random projection techniques have been applied to computing efficient solutions for correlation clustering and problems alike. This may be of independent interest, as neighborhood similarity is known to be used in various tools such as *almost-clique decompositions* [36, 19, 33, 34, 32, 4, 30, 6, 24, 7].

Single-Pass Semi-Streaming. While the above techniques are sufficient for getting our MPC and sequential algorithms, the single-pass semi-streaming algorithm introduces additional technical difficulties. The main difficulty for a single-pass semi-streaming to work here lies in Step 2, where an arbitrarily chosen high-degree vertex in each cluster proposes to its neighbors who have similar neighborhoods. For convenience, we call the chosen high-degree vertices *pivots* here.

To be able to do this, we need to memorize the neighbors of all the chosen vertices using $n\,\mathrm{polylog}(n)$ space. Suppose that $\mathrm{OPT} \le \phi$, it can be shown that each cluster in the optimal clustering containing at least one low-degree vertex has size of $\Theta(\phi)$. As a result, any vertex from these clusters has degree at most $O(\phi)$. Since we pick a pivot per cluster, $O(\phi \cdot (n/\phi))$ is the space we need to store the neighbors of the pivots.

However, we do not know beforehand how the clusters of high-degree vertices look like, so the pivots cannot be chosen at the beginning of the stream. Without knowing what the pivots are beforehand, it is difficult to store their neighbors in the same pass.

As a result, we sample each vertex (both high-degree and low-degree vertices) independently with probability $O(\log n/d(v))$ so that w.h.p. each cluster in the optimal clustering has $O(\log n)$ sampled vertices. Then we store the neighbors of all the sampled vertices. This poses another problem: low-degree vertices may be chosen as pivots. However, a low-degree vertex is exempted from our structural result – using it as a pivot may steal vertices from other clusters in an optimal clustering.

To resolve this, we do the following. For each sampled vertex y, we first try to recover the high-degree portion L of the cluster containing y in the optimal solution. We construct a candidate set $\mathrm{Cand}(L)$ that contains vertices that would not be added to other clusters. When using y as a pivot, we restrict it to consider only the intersection with the candidate set, $N[y] \cap \mathrm{Cand}(L)$ to ensure that it does not steal vertices from other clusters.

Roughly speaking, the candidate set $\mathrm{Cand}(L)$ contains all the low-degree vertices that have similar neighborhood with every vertex in L but have different neighborhood with every vertex in any other cluster L' (as a result, $\mathrm{Cand}(L) \cap \mathrm{Cand}(L') = \varnothing$). We show such a modification does not affect the approximation ratio. Furthermore, since the candidate sets are defined based on similarity of neighborhoods, they can be constructed by the aforementioned dimension reduction technique, which takes $O(n \log n/\epsilon^2)$ space.

2 Preliminaries

▶ **Definition 3.** *Given $u \in G^+ = (V, E^+)$, let $N(u)$ denote the neighbors of $u \in G^+$. Define $N[u] = N(u) \cup \{u\}$. $d(u) = |N(u)|$ is the number of neighbors of $u \in G^+$. We use $N_H(u)$,$N_H[u]$ and $d_H(u)$ to denote the the corresponding quantities in graph H when $H \neq G^+$.*

▶ **Definition 4.** *Let A and B be sets. The symmetric difference between A and B, $A\Delta B$, is defined as $A\Delta B = (A \setminus B) \cup (B \setminus A)$.*

In general, we say A and B are similar if $|A\Delta B|$ is small.

▶ **Lemma 5** (Triangle Inequality [35])**.** *Let A, B, C be sets. Then: $|A\Delta C| \le |A\Delta B| + |B\Delta C|$.*

▶ **Definition 6.** *Given any clustering $\mathcal{C}$ and any vertex u, $\mathcal{C}_u$ is defined to be the cluster of $\mathcal{C}$ containing u.*

▶ **Definition 7.** *Given clustering $\mathcal{C}$, define $\rho_{\mathcal{C}}(x)$ as the number of disagreements incident to vertex x. More precisely:*

$$\rho_{\mathcal{C}}(x) = |\mathcal{C}_x \setminus N[x]| + |N[x] \setminus \mathcal{C}_x| = |N[x]\Delta\mathcal{C}_x| = |N(x) \,\Delta\, \mathcal{C}_x| - 1$$

▶ **Definition 8.** *Given a clustering $\mathcal{C}$, the objective value of $\mathcal{C}$, $\mathrm{obj}(\mathcal{C}) = \max_u \rho_{\mathcal{C}}(u)$, is defined as the maximum incident disagreements over every vertex.*

The MPC Model. In the MPC model, computation proceeds in synchronous parallel *rounds* across multiple machines. Each machine has memory S. At the beginning of the computation, data is arbitrarily partitioned across the machines. During each round, machines process data locally, exchange messages with other machines, and send or receive messages of total size S. The efficiency of an algorithm in this model is measured by the number of rounds required for the algorithm to terminate and the size S of the memory available to each machine.

In this paper, we focus on the most practical and challenging regime, also known as the *strictly sublinear regime*, where each machine has $S = O(n^{\delta})$ local memory. Here, n represents the number of vertices, and $0 < \delta < 1$ is an arbitrary constant. Under this assumption, the input assigned to each machine and the messages exchanged during any round are of size $O(n^{\delta})$.

The Semi-Streaming Model. In the semi-streaming model, the input is a stream of edges in E^{+}. We are allowed to use $n \operatorname{polylog} n$ space, where the space complexity is the number of words used and a word consists of $O(\log n)$ bits. A solution is expected to be output at the end of the stream.

3 Algorithm

▶ **Definition 9.** *For any $0 \le \eta < 1$, an η-similarity query $\Delta_\eta(u, v, t)$ returns:*

$$\Delta_\eta(u,v,t) = \begin{cases} 0, & \text{if } |N[u] \Delta N[v]| > (1+\eta)t. \\ 0 \text{ or } 1, & \text{if } t < |N[u] \Delta N[v]| \le (1+\eta)t. \\ 1, & \text{if } |N[u] \Delta N[v]| \le t. \end{cases}$$

▶ **Definition 10.** *(Shorthand for η-similarity query) For readability, we use $|N[u] \Delta N[v]| \le_\eta t$ to denote that an η-similarity query $\Delta_\eta(u, v, t)$ has been conducted and returned 1. We use $|N[u] \Delta N[v]| \not\le_\eta t$ to denote that the query $\Delta_\eta(u, v, t)$ returned 0.*

The parameter η was introduced to accommodate the error induced by an approximate test of whether $|N[u] \Delta N[v]| \le 2\phi$. For the sake of simplicity, we urge the readers to assume $\eta = 0$ when reading for the first time. When $\eta = 0$, $|N[u] \Delta N[v]| \le_\eta 2\phi$ if and only if $|N[u] \Delta N[v]| \le 2\phi$.

▶ **Definition 11.** *Define $V_{\text{low}} \leftarrow \{w \in V \mid d(w) \le (3+\eta)\phi\}$ and $V_{\text{high}} \leftarrow V \setminus V_{\text{low}}$, where we call the vertices in V_{low} and V_{high} as low-degree and high-degree vertices, respectively.*

Algorithm Description. Algorithm 1 takes two parameters ϕ and $0 \le \eta < 1$, where ϕ is a guess on the upper bound of OPT and η is an error control parameter. The goal of the algorithm is to output a solution of value $(3+\eta)\phi$ provided OPT $\le \phi$. Note that we use η instead of ϵ here to indicate that it can be set to zero (at the cost of computing the more expensive exact neighborhood similarity).

The algorithm works as follows: first, we form a clustering of high-degree vertices $\mathcal{L}$ based on the similarity between the neighborhood of vertices. If two high-degree vertices u and v have similar neighborhood (i.e. $N[u] \Delta N[v] \le_\eta 2\phi$) then they will be placed in the same cluster.

Once the high-degree clusters are formed, we go through each cluster $L_i \in \mathcal{L}$. For each L_i, we pick an arbitrary pivot $u_i \in L_i$. For each neighbor w of u_i that remains unclustered (i.e. in V_i), we include it in $R(u_i)$ if w and u_i have similar neighborhoods (i.e. $N[w] \Delta N[u_i] \le_\eta 2\phi$). Then we set our cluster C_i to be $L_i \cup R(u_i)$. Then we update the unclustered vertices to be $V_{i+1} = V_i \setminus R(u_i)$.

Algorithm 1 ClusterPhi($G^+ = (V, E^+), \phi, \eta$).
Input: A graph G^+ and parameters ϕ and $0 \le \eta < 1$.
Output: A clustering $\mathcal{C}$ with $\mathrm{obj}(\mathcal{C}) \le (3+\eta)\phi$ or "OPT > ϕ"

function ClusterPhi($G^+ = (V, E^+), \phi, \eta$)
  $V_{\text{low}} \leftarrow \{w \mid d(w) \le (3+\eta)\phi\}, V_{\text{high}} \leftarrow V \setminus V_{\text{low}}$
  for $u, v \in V_{high}$ **do** ▷ uv not necessarily in E^+.
    if $|N[u] \Delta N[v]| \le_\eta 2\phi$ **then**
      $E' \leftarrow E' \cup \{uv\}$.
    end if
  end for
  $V_1 = V_{\text{low}}$
  Let $\mathcal{L} = \{L_i\}_{i=1}^{|\mathcal{L}|}$ be the partition formed by the connected components in (V_{high}, E').
  for i from 1 to $|\mathcal{L}|$ **do**
    Choose any node $u_i \in L_i$.
    Compute $R(u_i) = \{w \in V_i \cap N(u_i) \mid |N[w] \Delta N[u_i]| \le_{\eta/2} 2\phi\}$
    $C_i \leftarrow L_i \cup R(u_i)$
    $V_{i+1} \leftarrow V_i \setminus R(u_i)$
  end for
  Let $\mathcal{C} = \{C_i\}_{i=1}^{|\mathcal{L}|} \cup \bigcup_{v \in V_{|\mathcal{L}|+1}} \{\{v\}\}$
  if for some i there exists $u \in C_i$ such that $\rho_{\mathcal{C}}(u) > (3+\eta)\phi$ **then**
    return "OPT > ϕ"
  else
    return $\mathcal{C}$.
  end if
end function

Once we went through every L_i, there might be still some unclustered low-degree vertices (i.e. those in $V_{|\mathcal{L}|+1}$). For each such vertex, we put it in a singleton cluster. Then, we check if the clustering we have obtained has an objective value at most $(3+\eta)\phi$ or not. If it does, then we are done. If not, we conclude that OPT > ϕ, so we would need to set our guess of ϕ larger.

▶ **Theorem 12.** *Suppose that* OPT $\le \phi$*, Algorithm 1 outputs a clustering* $\mathcal{C}$ *with* $\mathrm{obj}(\mathcal{C}) \le (3+\eta)\phi$.

Proof. Let $\mathcal{C}^*$ be an optimal solution so $\mathrm{obj}(\mathcal{C}^*) \le \phi$. In the next subsections, we show the following:

1. (High-Degree Nodes Clustering). For any $u \in V_{\text{high}}$, $\mathcal{L}_u \cap V_{\text{high}} = \mathcal{C}^*_u \cap V_{\text{high}}$.
2. (No Stealing on Low-Degree Nodes). For each $i \in [1, |\mathcal{L}|]$, let C_i^* be the cluster in $\mathcal{C}^*$ such that $L_i \cap V_{\text{high}} = C_i^* \cap V_{\text{high}}$. We have $C_i^* \cap V_i = C_i^* \cap V_{\text{low}}$. That is, those low-degree vertices in C_i^* are not taken by other clusters C_j for $j < i$.
3. (Low-Degree Nodes Inclusion). For any $i \in [1, |\mathcal{L}|]$, $C_i^* \cap N(u_i) \subseteq C_i$.
4. (Closeness). For any $i \in [1, |\mathcal{L}|]$, $|N[u_i] \Delta C_i| \le \phi$ and $|C_i^* \Delta C_i| \le \phi$.

Once we have shown the above, we can see that $\mathrm{obj}(\mathcal{C}) \le (3+\eta)\phi$ as follows. Consider a component C_i of $\mathcal{C}$. If C_i is a singleton $\{v\}$ with $d(v) \le (3+\eta)\phi$, then obviously, $\rho_{\mathcal{C}}(v) \le (3+\eta)\phi$. Otherwise, C_i contains some vertex x with $d(x) > (3+\eta)\phi$. By Item 1, there must exist C_i^* such that $C_i \cap V_{\text{high}} = C_i^* \cap V_{\text{high}}$.

Let v be any vertex in C_i. We will show that $\rho_{\mathcal{C}}(v) \le 3\phi$. Suppose that $v \in C_i^* \cap C_i$, we have:

$$\begin{aligned}\rho_{\mathcal{C}}(v) = |N[v] \Delta C_i| &\le |N[v] \Delta C_i^*| + |C_i^* \Delta C_i| && \text{by Lemma 5}\\ &= \rho_{\mathcal{C}^*}(v) + |C_i^* \Delta C_i| \le \phi + \phi = 2\phi\end{aligned}$$

Otherwise, if $v \notin C_i^* \cap C_i$ then it must be the case that $v \in C_i \setminus C_i^*$. In such a case, v must be a vertex in $R(u_i)$ added to C_i in Line 12 to Line 13. This implies $|N[v] \Delta N[u_i]| \le_{\eta/2} 2\phi$ and thus $|N[v] \Delta N[u_i]| \le (1 + \eta/2) \cdot 2\phi$. Therefore:

$$\begin{aligned}\rho_{\mathcal{C}}(v) = |N[v] \Delta C_i| &\le |N[v] \Delta N[u_i]| + |N[u_i] \Delta C_i| && \text{by Lemma 5}\\ &\le (1 + \eta/2) \cdot 2\phi + \phi = (3 + \eta)\phi\end{aligned}$$

◀

▶ Remark 13. To get a $(3 + \epsilon)$-approximation algorithm, $O(\log n)$ calls of Algorithm 1 are sufficient, as we can perform a binary search on ϕ in the range of $[0, n]$ with $\eta = \epsilon$ and take the solution output by the algorithm with the smallest ϕ. For a 3-approximation algorithm, it can also be achieved within $O(\log n)$ calls of Algorithm 1 by setting $\eta = 0$.

We prove each of the four items in the proof in the following subsections. Note that the no-stealing property of low-degree vertices (Theorem 17 in Section 3.2) is the main structural result, which not only leads to a guarantee on the approximation ratio, but it is also crucial in the design of efficient algorithms in the subsequent sections.

3.1 High-Degree Nodes Clustering

Recall that ϕ is our guess of the upper bound of the optimal solution. In this subsection, we show if ϕ is indeed such an upper bound, then there is a unique way to form clustering on high-degree nodes in the optimal solution. Our algorithm clusters the high-degree nodes using the unique way. The following two lemmas are observed by [37].

▶ **Lemma 14.** *Suppose that $|N[x] \cap N[y]| > t$. If there is a clustering $\mathcal{C}'$ such that x and y are in different clusters, then* $\mathrm{obj}(\mathcal{C}') > t/2$.

Proof.

$$t < |N[x] \cap N[y]| \le |N[x] \cap N[y] \setminus \mathcal{C}'_y| + |N[x] \cap N[y] \setminus \mathcal{C}'_x| \le \rho_{\mathcal{C}'}(y) + \rho_{\mathcal{C}'}(x)$$

Therefore, $\mathrm{obj}(\mathcal{C}') \ge \max(\rho_{\mathcal{C}'}(x), \rho_{\mathcal{C}'}(y)) \ge (\rho_{\mathcal{C}'}(x) + \rho_{\mathcal{C}'}(y))/2 > t/2$. ◀

▶ **Lemma 15.** *Suppose that $|N[x] \Delta N[y]| > t$. If there is a clustering $\mathcal{C}'$ such that x and y are in the same clusters, then* $\mathrm{obj}(\mathcal{C}') > t/2$.

Proof. Let C be the cluster containing x and y. By Lemma 5, we have:

$$t < |N[x] \Delta N[y]| \le |N[x] \Delta C| + |N[y] \Delta C| = \rho_{\mathcal{C}'}(x) + \rho_{\mathcal{C}'}(y)$$

Therefore, $\mathrm{obj}(\mathcal{C}') \ge \max(\rho_{\mathcal{C}'}(x), \rho_{\mathcal{C}'}(y)) \ge (\rho_{\mathcal{C}'}(x) + \rho_{\mathcal{C}'}(y))/2 > t/2$. ◀

▶ **Lemma 16.** *Let $\mathcal{C}^*$ be a clustering with* $\mathrm{obj}(\mathcal{C}^*) \le \phi$. *For any $u \in V_{\mathrm{high}}$, $\mathcal{L}_u \cap V_{\mathrm{high}} = \mathcal{C}_u^* \cap V_{\mathrm{high}}$.*

Proof. Suppose on the contrary that $\mathcal{L}_u \cap V_{\text{high}} \neq \mathcal{C}^*_u \cap V_{\text{high}}$. Then there exists some node x such that $x \in (\mathcal{L}_u \cap V_{\text{high}}) \Delta (\mathcal{C}^*_u \cap V_{\text{high}})$.

Suppose that there exists a node x such that $x \in \mathcal{L}_u \cap V_{\text{high}}$ but $x \notin \mathcal{C}^*_u \cap V_{\text{high}}$. Then there must exist $y \in \mathcal{C}^*_u$ such that $(x, y) \in E'$. This implies $|N[x] \Delta N[y]| \leq_\eta 2\phi$ and so $|N[x] \Delta N[y]| \leq (1+\eta)2\phi$. Therefore,

$$\begin{aligned} |N[x] \cap N[y]| &= (d(x) + 1 + d(y) + 1 - |N[x] \Delta N[y]|)/2 \\ &> ((3+\eta)\phi + 1 + (3+\eta)\phi + 1 - (1+\eta)2\phi)/2 \\ &= (6\phi + 2 - 2\phi)/2 = 2\phi + 1 \end{aligned}$$

By Lemma 14, $\text{obj}(\mathcal{C}^*) > \phi$, a contradiction.

Otherwise, it must be the case that $x \in \mathcal{C}^*_u \cap V_{\text{high}}$ but $x \notin \mathcal{L}_u \cap V_{\text{high}}$. In this case, there exists $y \in \mathcal{C}^*_u$ such that $(x, y) \notin E'$. This implies that $|N[x] \Delta N[y]| \not\leq_\eta 2\phi$, which in turns implies $|N[x] \Delta N[y]| > 2\phi$. By Lemma 15, $\text{obj}(\mathcal{C}^*) > \phi$, a contradiction. ◀

3.2 No Stealing on Low-Degree Nodes

In the algorithm, low-degree nodes (i.e. nodes with degree at most $(3+\eta)\phi$) are added to the clusters formed by high-degree nodes iteratively. In this subsection, we show that if a low-degree node degree node y belongs to C^*_i for some i, then it will not be included in C_j for $j < i$. This implies that $y \in V_i$, the candidate set of vertices to be added C_i. Then in the next subsection, we show that it will be added to C_i.

▶ **Theorem 17.** *Let $\mathcal{C}^*$ be a clustering with obj$(\mathcal{C}^*) \leq \phi$. Let u and v be vertices of degree greater than $(3+\eta)\phi$ where $\mathcal{C}^*_u \neq \mathcal{C}^*_v$, and w be a vertex with $d(w) \leq (3+\eta)\phi$. If $w \in \mathcal{C}^*_u$ then $|N[v] \Delta N[w]| > (1+\eta)2\phi$ and so $|N[v] \Delta N[w]| \not\leq_\eta 2\phi$.*

A representative case that illustrates the intuition of why the theorem holds is when $|N[u] \cap N[v] \cap N[w]| \leq \phi$, i.e. the three sets have small intersections. Since they have small intersection, together with the fact that u and v have degrees greater than $(3+\eta)\phi$, it cannot be the case that both the symmetric differences $N[u] \Delta N[w]$ and $N[v] \Delta N[w]$ are small, as illustrated in Figure 1. We will refer the case that $|N[u] \cap N[v] \cap N[w]| \leq \phi$ as the *easy case*.

When the three sets have a large intersection, with a more sophisticated argument on the relations among $N[u], N[v]$, and $N[w]$, it can also be shown that $N[w]$ and $N[u]$ will not intersect a lot. We will refer to the case that $|N[u] \cap N[v] \cap N[w]| > \phi$ as *the hard case*.

3.2.1 The easy case

We will first show the theorem for the easy case as follows.

▶ **Lemma 18.** *Let $\mathcal{C}^*$ be a clustering with obj$(\mathcal{C}^*) \leq \phi$. Let u and v be vertices of degree greater than $(3+\eta)\phi$ where $\mathcal{C}^*_u \neq \mathcal{C}^*_v$, and w be a vertex with $d(w) \leq (3+\eta)\phi$. If $w \in \mathcal{C}^*_u$ and $|N[v] \cap N[u] \cap N[w]| \leq \phi$, then $|N[v] \Delta N[w]| > (1+\eta)2\phi$ and so $|N[v] \Delta N[w]| \not\leq_\eta 2\phi$.*

Proof. We will show that $|N[v] \Delta N[w]| + |N[u] \Delta N[w]|$ is greater than $(1+\eta)4\phi$. Figure 1 gives a high level illustration of why this should be true. To show this formally, first note that:

$$\begin{aligned} &|N[v] \cap N[w] \cap N[u]| + |N[w] \setminus N[u]| + |N[v] \setminus N[w]| \\ &\geq |N[v] \cap N[w] \cap N[u]| + |(N[v] \cap N[w]) \setminus N[u]| + |N[v] \setminus N[w]| \\ &= |N[v] \cap N[w]| + |N[v] \setminus N[w]| = |N[v]| \end{aligned}$$

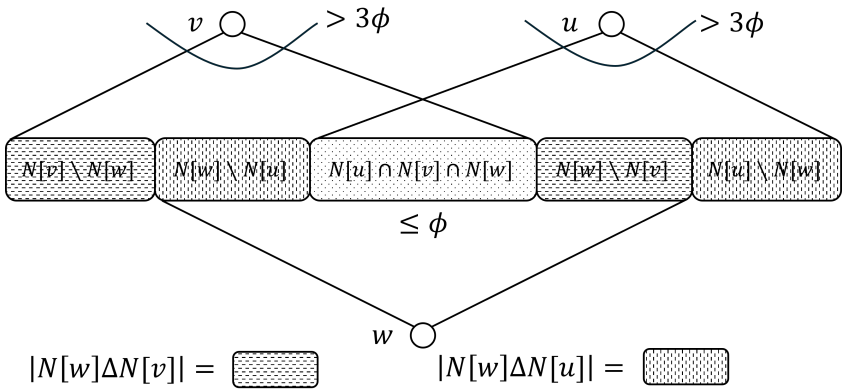


Figure 1 A pictorial illustration of the proof of Lemma 18 when $\eta = 0$. It shows that if $N[u] \cap N[v] \cap N[w]$ is small, then either $N[w]\Delta N[v]$ or $N[w]\Delta N[u]$ has to be large, since $|N[u]|$ and $|N[v]|$ are more than 3ϕ.

Re-arranging, we have:

$$\begin{aligned} |N[w] \setminus N[u]| + |N[v] \setminus N[w]| &\geq |N[v]| - |N[v] \cap N[w] \cap N[u]| \\ &> (3+\eta)\phi - \phi = (2+\eta)\phi \end{aligned} \tag{1}$$

By the same reasoning, we have

$$|N[w] \setminus N[v]| + |N[u] \setminus N[w]| > (2+\eta)\phi \tag{2}$$

Now consider the following:

$$\begin{aligned} &|N[v]\Delta N[w]| + |N[u]\Delta N[w]| \\ &= |N[v] \setminus N[w]| + |N[w] \setminus N[v]| + |N[u] \setminus N[w]| + |N[w] \setminus N[u]| \\ &> (2+\eta)\phi + (2+\eta)\phi \qquad \text{by (1) and (2)} \\ &= (4+2\eta)\phi \end{aligned}$$

Therefore, assume to the contrary that $|N[v]\Delta N[w]| \leq_\eta 2\phi$ and so $|N[v]\Delta N[w]| \leq (1+\eta)2\phi$. Then, it must be the case that $|N[u]\Delta N[w]| > (4+2\eta)\phi - |N[v]\Delta N[w]| \geq 2\phi$. By the fact that $w \in \mathcal{C}_v^*$ and Lemma 15, $\text{obj}(\mathcal{C}^*) > \phi$, a contradiction. ◀

3.2.2 The hard case

Now we consider the case where $|N[w] \cap N[u] \cap N[v]| > \phi$. To illustrate the idea of proof, suppose that $|N[w] \cap N[u] \cap N[v]| = \phi + t$ for some $t > 0$. If we follow the same argument as Lemma 18, we would only be able to show that $|N[w]\Delta N[v]| + |N[w]\Delta N[u]| > (4+2\eta)\phi - 2t$ as opposed to $(4+2\eta)\phi$.

However, if this is the case, we can show that $|N[w]\Delta N[u]| \leq 2(\phi - t)$, as stated in Lemma 19. This would make $|N[w]\Delta N[v]| > (1+\eta)2\phi$.

The high-level idea of the proof of Lemma 19 is illustrated in Figure 2. First, we observe that at most ϕ vertices in $|N[u] \cap N[v] \cap N[w]|$ can be contained in $\mathcal{C}_u^*$; otherwise the vertex v would have disagreements greater than ϕ. This implies the contribution of disagreement from $N[u] \cap N[w]$ to u (and to w) is already at least t. If the symmetric difference $|N[u]\Delta N[w]|$ is too large, i.e. larger than $2(\phi - t)$, then it would force the disagreements of either u or w to be too large.

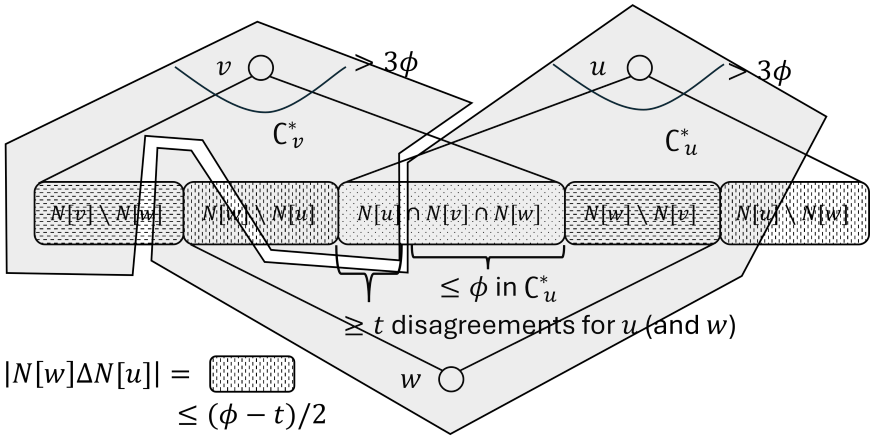


Figure 2 A pictorial illustration of the proof of Lemma 19 when $\eta = 0$. This means that $N[u] \cap N[v] \cap N[w]$ must contribute at least t disagreements to both u and v. Additionally, each element in $|N[u]\Delta N[w]|$ contributes a disagreement to at least one of u or w, so $|N[u]\Delta N[w]|$ cannot exceed $(\phi - t)/2$; otherwise, the disagreement of u or w would exceed ϕ.

▶ **Lemma 19.** *Let $\mathcal{C}^*$ be a clustering with $obj(\mathcal{C}^*) \leq \phi$. Let u and v be vertices of degree greater than $(3+\eta)\phi$ where $\mathcal{C}^*_u \neq \mathcal{C}^*_v$, and $w \in \mathcal{C}^*_u$ be a vertex with $d(w) \leq (3+\eta)\phi$. Suppose that $|N[w] \cap N[v] \cap N[u]| = \phi + t$ for some $t > 0$. Then, $|N[w]\Delta N[u]| \leq 2(\phi - t)$.*

Proof. Let $S = N[w] \cap N[v] \cap N[u]$. First we claim that at most ϕ vertices in S can be in $\mathcal{C}^*_u$. This in turns implies that vertices in S contributes at least t disagreements to vertex u and vertex w. To see why this holds, suppose to the contrary that $|S \cap \mathcal{C}^*_u| > \phi$. Since $S \subseteq N[v]$ and $\mathcal{C}^*_v \neq \mathcal{C}^*_u$, it must be the case that $\rho_{\mathcal{C}^*}(v) \geq |N[v] \setminus C^*_v| \geq |S \setminus C^*_v| \geq |S \cap \mathcal{C}^*_u| > \phi$, a contradiction. Therefore, we conclude that:

$$|S \setminus \mathcal{C}^*_u| \geq t \tag{3}$$

Now note that the disagreements of u and w are at least:

$$\rho_{\mathcal{C}^*}(u) \geq |S \setminus \mathcal{C}^*_u| + |(N[w] \setminus N[u]) \cap \mathcal{C}^*_u| + |(N[u] \setminus N[w]) \setminus \mathcal{C}^*_u|$$
$$\rho_{\mathcal{C}^*}(w) \geq |S \setminus \mathcal{C}^*_u| + |(N[u] \setminus N[w]) \cap \mathcal{C}^*_u| + |(N[w] \setminus N[u]) \setminus \mathcal{C}^*_u|$$

Using the fact $\rho_{\mathcal{C}^*}(u), \rho_{\mathcal{C}^*}(w) \leq \phi$ and summing up the above two inequalities, we have:

$$\begin{aligned}
2\phi &\geq \rho_{\mathcal{C}^*}(u) + \rho_{\mathcal{C}^*}(w) \\
&\geq 2|S \setminus \mathcal{C}^*_u| + |(N[w] \setminus N[u]) \cap \mathcal{C}^*_u| + |(N[u] \setminus N[w]) \setminus \mathcal{C}^*_u| + \\
&\qquad |(N[u] \setminus N[w]) \cap \mathcal{C}^*_u| + |(N[w] \setminus N[u]) \setminus \mathcal{C}^*_u| \\
&= 2|S \setminus \mathcal{C}^*_u| + |(N[w]\Delta N[u]) \cap \mathcal{C}^*_u| + |(N[u]\Delta N[w]) \setminus \mathcal{C}^*_u| \\
&= 2|S \setminus \mathcal{C}^*_u| + |(N[w]\Delta N[u])| \\
&\geq 2t + |(N[w]\Delta N[u])|
\end{aligned}$$

Therefore, $|N[w]\Delta N[v]| \leq 2(\phi - t)$. ◀

▶ **Lemma 20.** *Let $\mathcal{C}^*$ be a clustering with $obj(\mathcal{C}^*) \leq \phi$. Let u and v be vertices of degree greater than $(3+\eta)\phi$ where $\mathcal{C}^*_u \neq \mathcal{C}^*_v$, and w be a vertex with $deg(w) \leq (3+\eta)\phi$. If $w \in \mathcal{C}^*_u$ and $|N[v] \cap N[u] \cap N[w]| > \phi$, then $|N[v]\Delta N[w]| > (1+\eta)2\phi$ and so $|N[v]\Delta N[w]| \not\leq_\eta 2\phi$.*

Proof. By (1) in the proof of Lemma 18, we have:

$$\begin{aligned}|N[w] \setminus N[u]| + |N[v] \setminus N[w]| &\geq |N[v]| - |N[v] \cap N[w] \cap N[u]| \\ &> (3+\eta)\phi - (\phi + t) = (2+\eta)\phi - t \\ |N[w] \setminus N[v]| + |N[u] \setminus N[w]| &\geq |N[u]| - |N[v] \cap N[w] \cap N[u]| \\ &> (3+\eta)\phi - (\phi + t) = (2+\eta)\phi - t\end{aligned}$$

Therefore,

$$\begin{aligned}|N[v] \Delta N[w]| + |N[u] \Delta N[w]| &= |N[v] \setminus N[w]| + |N[w] \setminus N[v]| \\ &\quad + |N[u] \setminus N[w]| + |N[u] \setminus N[w]| \\ &> (4+2\eta)\phi - 2t\end{aligned}$$

By Lemma 19, we have $|N[u] \Delta N[w]| \geq 2(\phi - t)$, therefore:

$$|N[v] \Delta N[w]| > (4+2\eta)\phi - 2t - 2(\phi - t) = (1+\eta)2\phi$$

Hence, $|N[v] \Delta N[w]| \not\leq_\eta 2\phi$. ◀

Lemma 18 and Lemma 20 complete the proof of Theorem 17. Theorem 17 immediately implies the following:

▶ **Corollary 21.** *Let C_i^* be the cluster in $\mathcal{C}^*$ such that $L_i \cap V_{\text{high}} = C_i^* \cap V_{\text{high}}$. For any i, $C_i^* \cap V_i = C_i^* \cap V_{\text{low}}$.*

3.3 Low-Degree Nodes Inclusion

Here, we show that all the vertices in $\mathcal{C}_i^*$ have similar neighborhoods with u_i. As a result, if they are neighbors of u_i, they will be added to C_i.

▶ **Lemma 22.** *Let C_i^* be the cluster in $\mathcal{C}^*$ such that $L_i \cap V_{\text{high}} = C_i^* \cap V_{\text{high}}$. For any i, $C_i^* \cap N(u_i) \subseteq C_i$.*

Proof. If $w \in C_i^* \cap V_{\text{high}}$, then $w \in C_i$ because $C_i = L_i \cup R(u_i)$ and $L_i \cap V_{\text{high}} = C_i^* \cap V_{\text{high}}$ by Lemma 16. Now it remains to consider the case $w \in C_i^* \cap V_{\text{low}} \cap N(u_i)$. It suffices for us to show that $w \in R(u_i)$, where $R(u_i) = \{w \in V_i \cap N(u_i) \mid |N[w] \Delta N[u_i]| \leq_\eta 2\phi\}$, as $R(u_i)$ will be added to C_i.

By Corollary 21, we have $w \in C_i^* \cap V_i \cap N(u_i)$, which means w is not going to be stolen by other clusters. To show that $w \in C_i$, now it suffices to show that $|N[w] \Delta N[u_i]| \leq_\eta 2\phi$ always holds. This is indeed true, because:

$$\begin{aligned}|N[w] \Delta N[u_i]| &\leq |N[w] \Delta C_i^*| + |C_i^* \Delta N[u_i]| && \text{by Lemma 5} \\ &\leq \phi + \phi \leq 2\phi && u_i, w \in C_i^*, \rho_{\mathcal{C}^*}(w), \rho_{\mathcal{C}^*}(u_i) \leq \phi\end{aligned}$$ ◀

3.4 Closeness

In the following, we will show that the cluster we constructed C_i will be similar to C_i^* and $N[u_i]$. Intuitively, this holds because the low-degree part of C_i will be sandwiched between the low-degree part of $C_i^* \cap N[u_i]$ and $N[u_i]$, i.e. $C_i^* \cap N[u_i] \cap V_{\text{low}} \subseteq C_i \cap V_{\text{low}} \subseteq N[u_i] \cap V_{\text{low}}$. Also, we know that $N[u_i]$ and C_i^* are close (i.e. $|N[u_i] \Delta C_i^*| \leq \phi$), so somehow C_i cannot be too far away from C_i^* and $N[u_i]$. Note that the high-degree parts of C_i and C_i^* coincide.

▶ **Lemma 23.** *Let C_i^* be the cluster in $\mathcal{C}^*$ such that $L_i \cap V_{\text{high}} = C_i^* \cap V_{\text{high}}$. For any i, $|N[u_i] \Delta C_i| \leq \phi$ and $|C_i^* \Delta C_i| \leq \phi$.*

Proof. We first show that $|N[u_i] \Delta C_i| \leq \phi$.

$$\begin{aligned}
|V_{\text{low}} \cap (N[u_i] \Delta C_i)| &= |V_{\text{low}} \cap (N[u_i] \setminus C_i)| && (V_{\text{low}} \cap C_i) \subseteq N[u_i] \\
&\leq |V_{\text{low}} \cap (N[u_i] \setminus C_i^*)| && \text{by Lemma 22} \\
&\leq |V_{\text{low}} \cap (N[u_i] \Delta C_i^*)| && (4)
\end{aligned}$$

Moreover, since $C_i^* \cap V_{\text{high}} = C_i \cap V_{\text{high}}$, it must be the case that

$$|(N[u_i] \Delta C_i) \cap V_{\text{high}}| = |(N[u_i] \Delta C_i^*) \cap V_{\text{high}}| \quad (5)$$

Therefore,

$$\begin{aligned}
&|N[u_i] \Delta C_i| \\
&= |(N[u_i] \Delta C_i) \cap V_{\text{low}}| + |(N[u_i] \Delta C_i) \cap V_{\text{high}}| \\
&\leq |(N[u_i] \Delta C_i^*) \cap V_{\text{low}}| + |(N[u_i] \Delta C_i^*) \cap V_{\text{high}}| && \text{by (4) and (5)} \\
&= |N[u_i] \Delta C_i^*| \leq \phi && u_i \in C_i^* \text{ and } \rho_{C^*}(u_i) \leq \phi
\end{aligned}$$

Now we show that $|C_i^* \Delta C_i| \leq \phi$.

$$\begin{aligned}
&|C_i^* \Delta C_i| \\
&= |V_{\text{low}} \cap (C_i^* \Delta C_i)| && C_i^* \cap V_{\text{high}} = C_i \cap V_{\text{high}} \\
&= |V_{\text{low}} \cap (C_i^* \setminus C_i)| + |V_{\text{low}} \cap (C_i \setminus C_i^*)| \\
&= |V_{\text{low}} \cap (C_i^* \cap N[u_i] \setminus C_i)| + |V_{\text{low}} \cap (C_i^* \setminus N[u_i] \setminus C_i)| \\
&\qquad + |V_{\text{low}} \cap (C_i \setminus C_i^*)| \\
&= |V_{\text{low}} \cap (C_i^* \cap N[u_i] \setminus C_i)| + |V_{\text{low}} \cap (C_i^* \setminus N[u_i])| \\
&\qquad + |V_{\text{low}} \cap (C_i \setminus C_i^*)| && V_{\text{low}} \cap C_i \subseteq V_{\text{low}} \cap N[u_i] \\
&= |V_{\text{low}} \cap (C_i^* \setminus N[u_i])| + |V_{\text{low}} \cap (C_i \setminus C_i^*)| && C_i^* \cap N[u_i] \subseteq C_i \\
&\leq |V_{\text{low}} \cap (C_i^* \setminus N[u_i])| + |V_{\text{low}} \cap (N[u_i] \setminus C_i^*)| && V_{\text{low}} \cap C_i \subseteq V_{\text{low}} \cap N[u_i] \\
&= |V_{\text{low}} \cap (C_i^* \Delta N[u_i])| \leq |C_i^* \Delta N[u_i]| \leq \phi
\end{aligned}$$

◀

4 Efficient Algorithms for the Sequential and MPC Models

To make the algorithm efficient in the sequential and MPC settings, we need to address several challenges. First, we need to compute the graph E' for high-degree nodes. Second, we need to be able to conduct approximate neighborhood similarity queries efficiently. We will address these issues one by one.

4.1 Computing the Clustering for High-Degree Nodes

In this section, we show that although E' may be significantly larger than E^+, it is sufficient to make only $O(|E^+|) = O(m)$ neighborhood similarity queries to identify the high-degree clusters. Algorithm 2 describes how to form a clustering $\mathcal{F}$ for high-degree vertices by conducting $O(m)$ similarity queries.

Algorithm 2 HighDegreeClustering$(G^+ = (V, E^+), \phi, \eta)$.
Input: A graph G^+, a parameter ϕ, a parameter $0 \le \eta < 1$.
Output: A clustering $\mathcal{F} = \{L_i\}_{i=1}^{|\mathcal{L}|}$ of V_{high} such that $\mathcal{F} = \mathcal{L}$.

1: **function** HighDegreeClustering$(G^+ = (V, E^+), \phi, \eta)$
2: Let $G_{\text{sim}} = (V, E_{\text{sim}})$, where $E_{\text{sim}} = \{uv \in E^+ \mid |N[u] \Delta N[v]| \le_\eta 2\phi\}$.
3: Assign each vertex V with an unique ID, $\text{ID}(v)$.
4: Every vertex $v \in V$ sets $\min(v) \leftarrow \min_{x \in N_{G_{\text{sim}}}[v] \cap V_{\text{high}}} \text{ID}(x)$
5: Every vertex $v \in V$ sends a token of value $\min(v)$ to the neighbor of v in G_{sim}.
6: If a vertex $v \in V_{\text{high}}$ receives at least $\phi + 1$ tokens of the same value, set $cluster(v)$ to be the minimum such value.
7: Let $\mathcal{F}$ be a clustering where each cluster is formed by vertices with the same $cluster$ values.
8: **end function**

Algorithm Description. In Algorithm 2, we first perform $O(m)$ neighborhood queries to construct $G_{\text{sim}} = (V, E_{\text{sim}})$, where $E_{\text{sim}} \subseteq E^+$ consists of those endpoints who have similar neighborhoods. Then we assign every vertex u with a unique identifier, $\text{ID}(u)$. By sending the ID of every high-degree vertex to its neighbors in G_{sim}, every vertex u can learn ID of the high-degree vertex with the smallest ID in its closed neighborhood in G_{sim} (Line 4) and then set $\min(u)$ to be such an ID. Now, every vertex u (including low-degree vertices) sends a token with value $\min(u)$ to all the neighbors of u in G_{sim}. Then, if a vertex u recieved at least $\phi + 1$ tokens of the same value, it will set its cluster ID, $cluster(u)$, to be the minimum value that occurs at least $\phi + 1$ times. Finally, we assign all high-degree vertices with the same cluster ID to be in the same cluster.

In Lemma 24, we show that if OPT $\le \phi$, then the output $\mathcal{F}$ is exactly $\mathcal{L}$. The proof is based on showing that if two high-degree vertices u, v are in the same cluster in $\mathcal{C}^*$, then there will be at least $\phi + 1$ disjoint paths of length two in G_{sim}.

▶ **Lemma 24.** *efficienthighdeg Suppose that* OPT $\le \phi$. *Algorithm 2 outputs a clustering* $\mathcal{F}$ *such that* $\mathcal{F} = \mathcal{L}$.

Proof. Let $\mathcal{C}^*$ be a clustering with $\text{obj}(\mathcal{C}^*) \le \phi$. Let u be a vertex in V_{high}. Let u_* be the vertex with the minimum ID in $\mathcal{L}_u$. We will show that

1. Every vertex $x \in \mathcal{L}_u \cap V_{\text{high}}$ will receive at least $\phi + 1$ tokens of values $\text{ID}(u_*)$ in Line 6.
2. In addition, if x received at least $\phi + 1$ tokens of value T, then $T \ge \text{ID}(u_*)$.

Once these two points are established, every vertex $x \in \mathcal{L}_u \cap V_{\text{high}}$ will be assigned $\texttt{cluster}(x) = \text{ID}(u_*)$. Consequently, this ensures that $\mathcal{F}_u = \mathcal{L}_u$. Now we start with the first point, since $x, u_* \in \mathcal{C}^*_u$, $|N[u_*] \Delta N[x]| \le 2\phi$; for otherwise $\text{obj}(\mathcal{C}^*) > \phi$ by Lemma 15. This implies:

$$\begin{aligned} |N[u_*] \cap N[x]| &= (|N[x]| + |N[u_*]| - |N[u_*] \Delta N[x]|)/2 \\ &\ge (3\phi + 2 + 3\phi + 2 - 2\phi)/2 = 2\phi + 2 \end{aligned}$$

Also, since $u_* \in \mathcal{C}^*_u$ and by definition of $\rho_{\mathcal{C}^*}(u_*)$, we have:

$$\phi \ge \rho_{\mathcal{C}^*}(u_*) \ge |N[u_*]| - |(\mathcal{C}^*_u \cap N[u_*])| \tag{6}$$

Therefore,

$$\begin{aligned}|(\mathcal{C}_u^* \cap N[u_*]) \cap N[x]| &= |N[x] \cap N[u_*]| - |N[x] \cap (N[u_*] \setminus (\mathcal{C}_u^* \cap N[u_*]))| \\ &\geq |N[x] \cap N[u_*]| - |N[u_*] \setminus (\mathcal{C}_u^* \cap N[u_*])| \\ &\geq (2\phi + 2) - \phi \qquad \text{by (6)} \\ &\geq \phi + 2\end{aligned}$$

Hence:

$$|(\mathcal{C}_u^* \cap N[u_*]) \cap N(x)| \geq |(\mathcal{C}_u^* \cap N[u_*]) \cap N[x]| - 1 \geq \phi + 1$$

Note that all vertices in $(\mathcal{C}_u^* \cap N[u_*])$ must have their $\min(\cdot)$ values equal to $\mathrm{ID}(u_*)$. This is because if $v \in (\mathcal{C}_u^* \cap N[u_*])$ and $v \in V_{\text{high}}$, then $(N_{G_{\text{sim}}}(v) \cap V_{\text{high}}) \subseteq \mathcal{C}_u^*$. If $v \in (\mathcal{C}_u^* \cap N[u_*])$ and $v \notin V_{\text{high}}$, then it must be the case that $v \in N_{G_{\text{sim}}}(u_*)$. Moreover, $N_{G_{\text{sim}}}[v] \cap V_{\text{high}}$ cannot contain any cluster nodes outside $\mathcal{C}_u^*$ by Theorem 17. Thus, $\min(v) = \mathrm{ID}(u_*)$. This implies x will receive at least $\phi + 1$ tokens with value $\mathrm{ID}(u_*)$.

Next we show that if there is a value T such that x receives from at least $\phi + 1$ tokens, then $T \geq \mathrm{ID}(u_*)$. Suppose to the contrary that $T < \mathrm{ID}(u_*)$. First note that if $v \in N(x)$ has $\min(v) = T$, then it cannot be the case that $v \in V_{\text{high}}$. Otherwise, u_*, v, and a vertex whose ID is T, would all be in $\mathcal{C}_u^*$, which contradict with the fact $\mathrm{ID}(u_*)$ is the smallest ID in $\mathcal{L}_u = \mathcal{C}_u^* \cap V_{\text{high}}$. Therefore, it must be the case that vertices $v \in N(x)$ with $\min(v) = T$ are all in V_{low}.

Now note that if $v \in \mathcal{C}_u^*$ then by Theorem 17, v can only be connected to vertices in $\mathcal{C}_u^*$ in G_{sim}. In this case, $\min(v) \geq \mathrm{ID}(u_*)$ so $\min(v) \neq T$ by the assumption of $\mathrm{ID}(u_*)$. So the only possibility for v to have $\min(v) = T$ is when $v \notin \mathcal{C}_u^*$. However, as $\rho_{\mathcal{C}^*}(x) \leq \phi$, there are at most ϕ neighbors of x not in $\mathcal{C}_x^*$. This implies x will receive less than $\phi + 1$ tokens whose value equals to T. ◀

4.2 Approximate Neighborhood Similarity Testing by Random Projection

We show that for $0 < \eta < 1$, w.h.p. the queries $\Delta_\eta(x, y, 2\phi)$ for all $xy \in E^+$ can be answered in $O(m \log n/\eta^2)$ time and thus G_{sim} can be constructed in $O(m \log n/\eta^2)$ time.

▶ **Definition 25.** *Given a vertex x, let $\vec{N}[x] \subseteq \{0,1\}^n$ denote the characteristic vector of $N[x]$.*

▶ **Lemma 26.** *Given $\epsilon > 0$, let $k = C \cdot (\log n/\epsilon^2)$ for some large enough constant $C > 0$. Let A be a $k \times n$ matrix where each entry is drawn from $\{-1, +1\}$ uniformly at random. W.h.p. for every two vertices x and y, we have:*

$$(1-\epsilon) \cdot |N[x] \Delta N[y]| \leq ||A \cdot \vec{N}[x] - A \cdot \vec{N}[y]||_2^2/k \leq (1+\epsilon) \cdot |N[x] \Delta N[y]|$$

Proof. By the Johnson-Lindenstrauss lemma [1, 38], w.h.p. for every x and y, we have:

$$(1-\epsilon) \cdot ||\vec{N}[x] - \vec{N}[y]||_2^2 \leq ||A \cdot \vec{N}[x] - A \cdot \vec{N}[y]||_2^2/k \leq (1+\epsilon) \cdot ||\vec{N}[x] - \vec{N}[y]||_2^2$$

The lemma follows by observing that $|N[x] \Delta N[y]| = ||\vec{N}[x] - \vec{N}[y]||_2^2$. ◀

▶ **Lemma 27.** *Let $\epsilon = \Theta(\eta)$ be such that $(1+\eta) = (1+\epsilon)/(1-\epsilon)$, $k = O(\log n/\epsilon^2)$, and set:*

$$\Delta_\eta(x, y, t) = \begin{cases} 0 & \text{if } ||A \cdot \vec{N}[x] - A \cdot \vec{N}[y]||_2^2/((1+\epsilon)k) > t \\ 1 & \text{otherwise} \end{cases}$$

W.h.p. the above implementation returns a correct answer for $\Delta_\eta(x, y, t)$.

Proof. It suffices to show that if $|N[x]\Delta N[y]| \leq t$ then w.h.p. we will set $\Delta_\eta(x, y, t)$ to be 0 and if $|N[x]\Delta N[y]| > (1+\eta)t$, then w.h.p. we will set $\Delta_\eta(x, y, t)$ to be 1.

If $|N[x]\Delta N[y]| \leq t$, then w.h.p.

$$||A \cdot \vec{N}[x] - A \cdot \vec{N}[y]||_2^2/((1+\epsilon)k) \leq |N[x]\Delta N[y]| \leq t$$

Thus, $\Delta_\eta(x, y, t)$ returns 1.

On the other hand, if $|N[x]\Delta N[y]| > (1+\eta)t$ then w.h.p.

$$||A \cdot \vec{N}[x] - A \cdot \vec{N}[y]||_2^2/((1+\epsilon)k) \geq \frac{1-\epsilon}{1+\epsilon} \cdot (|N[x]\Delta N[y]|) > \frac{1-\epsilon}{1+\epsilon}(1+\eta) \cdot t = t$$

Thus, $\Delta_\eta(x, y, t)$ returns 0. ◀

We have presented the key ingredients that lead to efficient sequential and MPC algorithms. The details of their implementations are presented in the full version [17]. For the semi-streaming algorithm, different challenges arise. The necessary modifications and details are also included in the full version.

References

1. Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003. doi:10.1016/S0022-0000(03)00025-4.
2. Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. *Algorithmica*, 83(7):1980–2017, 2021. doi:10.1007/S00453-021-00816-9.
3. Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. doi:10.1145/1411509.1411513.
4. Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta+1)$ vertex coloring. In *Proc. 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 767–786, 2019. doi:10.1137/1.9781611975482.48.
5. Sepehr Assadi, Sanjeev Khanna, and Aaron Putterman. Correlation clustering and (de)sparsification: Graph sketches can match classical algorithms. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 417–428, 2025. doi:10.1145/3717823.3718194.
6. Sepehr Assadi, Pankaj Kumar, and Parth Mittal. Brooks' theorem in graph streams: A single-pass semi-streaming algorithm for Δ-coloring. *TheoretiCS*, 2, 2023. doi:10.46298/THEORETICS.23.9.
7. Sepehr Assadi and Chen Wang. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. In *13th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 215, pages 10:1–10:20, 2022. doi:10.4230/LIPICS.ITCS.2022.10.
8. Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004. doi:10.1023/B:MACH.0000033116.57574.95.
9. S. Behnezhad, M. Charikar, W. Ma, and L. Tan. Almost 3-approximate correlation clustering in constant rounds. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 720–731, 2022.
10. Guy E Blelloch, Jeremy T Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 308–317, 2012. doi:10.1145/2312005.2312058.
11. Mélanie Cambus, Davin Choo, Havu Miikonen, and Jara Uitto. Massively parallel correlation clustering in bounded arboricity graphs. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing (DISC)*, volume 209 of *LIPIcs*, pages 15:1–15:18, 2021. doi:10.4230/LIPICS.DISC.2021.15.

12 Mélanie Cambus, Fabian Kuhn, Etna Lindy, Shreyas Pai, and Jara Uitto. A single-pass semi-streaming algorithm for $(3+\varepsilon)$-approximate correlation clustering. In *Proc. 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2024. to appear.

13 Nairen Cao, Vincent Cohen-Addad, Euiwoong Lee, Shi Li, David Rasmussen Lolck, Alantha Newman, Mikkel Thorup, Lukas Vogl, Shuyi Yan, and Hanwen Zhang. Solving the correlation cluster lp in sublinear time. In *Proc. 57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1154–1165, 2025. doi:10.1145/3717823.3718181.

14 Nairen Cao, Vincent Cohen-Addad, Euiwoong Lee, Shi Li, Alantha Newman, and Lukas Vogl. Understanding the cluster linear program for correlation clustering. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1605–1616, 2024. doi:10.1145/3618260.3649749.

15 Nairen Cao, Shang-En Huang, and Hsin-Hao Su. Breaking 3-factor approximation for correlation clustering in polylogarithmic rounds. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4124–4154. SIAM, 2024. doi:10.1137/1.9781611977912.143.

16 Nairen Cao, Shi Li, and Jia Ye. Simultaneously approximating all norms for massively parallel correlation clustering. In *Proc. ICALP*, 2025. doi:10.4230/LIPIcs.ICALP.2025.40.

17 Nairen Cao, Steven Roche, and Hsin-Hao Su. Min-max correlation clustering via neighborhood similarity. *arXiv preprint arXiv:2502.12519*, 2025. doi:10.48550/arXiv.2502.12519.

18 Sayak Chakrabarty and Konstantin Makarychev. Single-pass pivot algorithm for correlation clustering. keep it simple! *arXiv preprint arXiv:2305.13560*, 2023. doi:10.48550/arXiv.2305.13560.

19 Yi-Jun Chang, Wenzheng Li, and Seth Pettie. An optimal distributed $(\Delta+1)$-coloring algorithm? In *Proc. 50th ACM Symposium on Theory of Computing (STOC)*, pages 445–456, 2018. doi:10.1145/3188745.3188964.

20 Moses Charikar, Neha Gupta, and Roy Schwartz. Local guarantees in graph cuts and clustering. In *Integer Programming and Combinatorial Optimization (IPCO)*, pages 136–147, 2017. doi:10.1007/978-3-319-59250-3_12.

21 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005. doi:10.1016/J.JCSS.2004.10.012.

22 Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 219–228, 2015. doi:10.1145/2746539.2746604.

23 Flavio Chierichetti, Nilesh Dalvi, and Ravi Kumar. Correlation clustering in mapreduce. In *Proc. 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 641–650, 2014. doi:10.1145/2623330.2623743.

24 Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 2069–2078, 2021. URL: http://proceedings.mlr.press/v139/cohen-addad21b.html.

25 Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2069–2078. PMLR, 2021. URL: http://proceedings.mlr.press/v139/cohen-addad21b.html.

26 Vincent Cohen-Addad, Euiwoong Lee, Shi Li, and Alantha Newman. Handling correlated rounding error via preclustering: A 1.73-approximation for correlation clustering. In *64rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2023. to appear. doi:10.1109/FOCS57990.2023.00065.

27 Vincent Cohen-Addad, Euiwoong Lee, and Alantha Newman. Correlation clustering with sherali-adams. In *63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 651–661. IEEE, 2022. doi:10.1109/FOCS54457.2022.00068.

28 Vincent Cohen-Addad, David Rasmussen Lolck, Marcin Pilipczuk, Mikkel Thorup, Shuyi Yan, and Hanwen Zhang. Combinatorial correlation clustering. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1617–1628, 2024. doi:10.1145/3618260.3649712.

29 Sami Davies, Benjamin Moseley, and Heather Newman. Fast combinatorial algorithms for min max correlation clustering. In *International Conference on Machine Learning*, pages 7205–7230. PMLR, 2023. URL: https://proceedings.mlr.press/v202/davies23a.html.

30 Manuela Fischer, Magnús M. Halldórsson, and Yannic Maus. Fast distributed brooks' theorem. In *Proc. ACM-SIAM Symposium on Discrete Algorithms SODA*, pages 2567–2588, 2023. doi:10.1137/1.9781611977554.CH98.

31 Manuela Fischer and Andreas Noever. Tight analysis of parallel randomized greedy MIS. *ACM Trans. Algorithms*, 16(1):6:1–6:13, 2020. doi:10.1145/3326165.

32 Maxime Flin, Mohsen Ghaffari, Magnús M. Halldórsson, Fabian Kuhn, and Alexandre Nolin. A distributed palette sparsification theorem. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4083–4123. SIAM, 2024. doi:10.1137/1.9781611977912.142.

33 Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Tigran Tonoyan. Efficient randomized distributed coloring in CONGEST. In *Proc. 53rd ACM Symposium on Theory of Computing (STOC)*, pages 1180–1193, 2021. doi:10.1145/3406325.3451089.

34 Magnús M. Halldórsson, Fabian Kuhn, Alexandre Nolin, and Tigran Tonoyan. Near-optimal distributed degree+1 coloring. In *Proc. 54th ACM Symposium on Theory of Computing (STOC)*, pages 450–463, 2022. doi:10.1145/3519935.3520023.

35 Paul R. Halmos. *Naive Set Theory*. Van Nostrand Reinhold, Princeton, NJ, 1960.

36 David G. Harris, Johannes Schneider, and Hsin-Hao Su. Distributed (Δ+1)-coloring in sublogarithmic rounds. *J. ACM*, 65(4):19:1–19:21, 2018. Preliminary version appeared in *Proc. 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 465–478, 2016. doi:10.1145/3178120.

37 Holger SG Heidrich, Jannik Irmai, and Bjoern Andres. A 4-approximation algorithm for min max correlation clustering. In *AISTATS*, pages 1945–1953, 2024.

38 William B. Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into hilbert space. *Contemporary mathematics*, 26:189–206, 1984. URL: https://api.semanticscholar.org/CorpusID:117819162.

39 Sanchit Kalhan, Konstantin Makarychev, and Timothy Zhou. Correlation clustering with local objectives. In *Advances in Neural Information Processing Systems (NuerIPS)*, volume 32, 2019.

40 Xinghao Pan, Dimitris Papailiopoulos, Samet Oymak, Benjamin Recht, Kannan Ramchandran, and Michael I. Jordan. Parallel correlation clustering on big graphs. In *Proc. 28th International Conference on Neural Information Processing Systems (NIPS)*, pages 82–90, 2015. URL: https://proceedings.neurips.cc/paper/2015/hash/b53b3a3d6ab90ce0268229151c9bde11-Abstract.html.

41 Gregory J. Puleo and Olgica Milenkovic. Correlation clustering and biclustering with locally bounded errors. *IEEE Trans. Inf. Theory*, 64(6):4105–4119, 2018. doi:10.1109/TIT.2018.2819696.

(Multivariate) k-SUM as Barrier to Succinct Computation

Geri Gokaj
Karlsruhe Institute of Technology, Germany

Marvin Künnemann
Karlsruhe Institute of Technology, Germany

Sabine Storandt
University of Konstanz, Germany

Carina Truschel
University of Konstanz, Germany

Abstract

How does the time complexity of a problem change when the input is given *succinctly* rather than explicitly? We study this question for several geometric problems defined on a set X of N points in $\mathbb{Z}^d$. As succinct representation, we choose a sumset (or Minkowski sum) representation: Instead of receiving X explicitly, we are given sets A, B of n points that define X as $A + B = \{a + b \mid a \in A, b \in B\}$.

We investigate the fine-grained complexity of this succinct version for several $\tilde{O}(N)$-time computable geometric primitives. Remarkably, we can tie their complexity tightly to the complexity of corresponding k-SUM problems. Specifically, we introduce as All-ints 3-SUM(n, n, k) the following multivariate, multi-output variant of 3-SUM: given sets A, B of size n and set C of size k, determine for all $c \in C$ whether there are $a \in A$ and $b \in B$ with $a + b = c$. We obtain the following results:

1. Succinct closest L_∞-pair requires time $N^{1-o(1)}$ under the 3-SUM hypothesis, while succinct furthest L_∞-pair can be solved in time $\tilde{O}(n)$.
2. Succinct bichromatic closest L_∞-Pair requires time $N^{1-o(1)}$ iff the 4-SUM hypothesis holds.
3. The following problems are fine-grained equivalent to All-ints 3-SUM(n, n, k): succinct skyline computation in 2D with output size k and succinct batched orthogonal range search with k given ranges. This establishes conditionally tight $\tilde{O}(\min\{nk, N\})$-time algorithms for these problems. We obtain further connections with All-ints 3-SUM(n, n, k) for succinctly computing independent sets in unit interval graphs.

Thus, (Multivariate) k-SUM problems precisely capture the barrier for enabling sumset-succinct computation for various geometric primitives.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fine-grained complexity theory, sumsets, additive combinatorics, succinct inputs, computational geometry

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.42

Funding *Geri Gokaj*: Research supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 462679611.
Carina Truschel: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 251654672 – TRR 161.

Acknowledgements We thank all reviewers for their in-depth review and feedback.



© Geri Gokaj, Marvin Künnemann, Sabine Storandt, and Carina Truschel;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Consider an algorithmic problem on an input set X of N points in $\mathbb{Z}^d$. How does the time complexity of the problem change when the input is given *succinctly* rather than explicitly? In this paper, we investigate this question when the succinct representation of X is chosen as a *sumset* (or *Minkowski sum*) representation $X = A + B \coloneqq \{a + b \mid a \in A, b \in B\}$.

Our motivation is threefold:

1. **Complexity on structured input:** How does the time complexity of the problem change when we are supplied with structural information about the input? As a case in point, this question has received considerable interest for the fundamental task of sorting. Indeed, the well-known Sorting $A + B$ problem studied by Fredman [20], and originally posed by Berlekamp[1], asks to determine whether we can compute, given A, B with $|A| = |B| = n$, the sorted order of $A + B$ in time $o(n^2 \log n)$. Since the succinct representation gives a large partial order essentially for free – simply presort A, B in time $O(n \log n)$ –, this question essentially boils to whether or not a partial solution structure helps significantly in determining the full output. The Sorting $A + B$ problem has several connections to problems in computational geometry such as polygon containment, computing the Minkowski sum of orthogonal polygons and minimizing the Hausdorff distance under translation for two segment sets, as described by the works of Barequet and Har-Peled [6], and Barrera [7].
   Another example that has received considerable attention is the *Pareto Sum* problem (see, e.g., [27, 24]), which asks to compute the skyline of $X = A + B$ (see Section 4 for a formal definition). It is only natural to ask analogous questions for other numerical/geometric problems. In particular, a succinct sumset representation reveals additive structure of the problem, which may or may not simplify the algorithmic task considerably.
2. **Algorithmic applications (sumset-compressed computational geometry):** Let $T(N)$ denote the optimal running time for solving a problem of interest on an input set of N points in $\mathbb{Z}^d$. For favorable inputs, it may be possible to find *much smaller* sets A, B with $X = A + B$ – in principle compressions to size $O(\sqrt{N})$ are possible. If we were able to solve the problem of interest in time $O(T(|A| + |B|))$ rather than $T(|X|)$ (aka *decompress-and-solve* running time), we could obtain significantly faster algorithms that reduce the running time from $T(N)$ to $O(T(\sqrt{N}))$. In fact, even small asymptotic improvements over $T(N)$ might be algorithmically interesting.
3. ***k*-SUM as precise barrier for succinct computations:** Our setup offers another perspective on the k-SUM problem. Specifically, for various problems, we shall prove that refuting a corresponding k-SUM hypothesis is *equivalent* to solving the sumset-succinct version of the problem faster than via decompress-and-solve. That is, despite its simple definition (asking a very basic question about sumsets), the k-SUM problem captures much more general primitives on succinct inputs. We believe that this yields valuable insights into the true expressiveness of k-SUM and the plausibility of the corresponding hardness hypothesis.

1.1 Our setup

For any algorithmic problem g defined on an input set $X \subseteq \mathbb{Z}^d$, we consider the following sumset-succinct version as *succinct-g*:

[1] In the literature, the name Sorting $X + Y$ is more common. However, throughout our paper X will denote the explicit input and $A + B$ its succinct representation.

Problem g		**Problem** succinct-g	
Input:	$X \subseteq \mathbb{Z}^d$ possibly additional inputs z	**Input:**	$A, B \subseteq \mathbb{Z}^d$ possibly additional inputs z
Output:	$g(X, z)$	**Output:**	$g(X, z)$ where $X = A + B$

As a convention, we set $N := |X|$, $n := |A|$ and $m := |B|$. For ease of presentation, we will generally assume that $n = m$, but our results can be adapted to the general case. Throughout the paper, we will assume that all input numbers are chosen from a polynomial range, i.e., $X \subseteq \{N^c, \ldots, N^c\}^d$ for some arbitrarily large constant c.

Let us consider a few (near-)linear time solvable problems g. We observe that their succinct versions may have very different time complexities:

A succintly easy problem: Selection

Consider the problem Selection: Given $X \subseteq \mathbb{Z}$ and $1 \leq k \leq N$, return the k-th largest element in X. Selection is well known to be solvable in time $O(N)$; see [8, 15]. Its succinct version succinct-Selection has received considerable attention as well [30, 29, 28, 19] and can be solved significantly faster than decompress-and-solve, specifically, in time $\tilde{O}(n)$.[2]

A succinctly hard problem: Skyline computation

Consider the problem of skyline computation (aka non-dominance filtering or Pareto front computation), see, e.g. [32, 14, 22]: Given $X \subseteq \mathbb{Z}^d$, compute the set Z of non-dominated points, i.e., all $x \in X$ for which there is no $x' \in X$ with $x' \neq x$ and $x \leq x'$ component-wise. This problem can be solved in near-linear time $\tilde{O}(N)$, see, e.g., [32]. Its succinct version is known as Pareto Sum and naturally occurs in several multi-objective optimization algorithms, see Hespe et al. [27]. Notably, a conditional lower bound of $N^{1-o(1)}$ that holds when the output size is $\Theta(\sqrt{N})$ has been proven under the $(\min, +)$-convolution hypothesis [21] and the 3-SUM hypothesis [24]. We shall investigate this problem in much more detail below.

On reductions to k-SUM

The 3-SUM problem[3] is one of the first problems used for conditional lower bounds. Initially introduced to explain various quadratic-time barriers observed in computational geometry [23], it has since been used to show quadratic-time hardness for a wealth of problems from various fields [41, 35, 5, 34, 13]. Its generalization, the k-SUM problem[4], has led to further conditional lower bounds beyond quadratic-time solvable problems [18, 4, 1, 2]. For a more comprehensive overview, we refer to the survey of Williams [43].

An important aspect of our investigations is that we establish *fine-grained equivalences* with k-SUM problems, which requires the somewhat unusual direction of reducing certain problems *to* k-SUM. Conveniently, we may exploit for this task a recent completeness theorem [24]: This work introduces general classes of problems which can be reduced to k-SUM, thereby simplifying the arguments needed to reduce sumset-related

[2] The authors consider a slight variant in which $A + B$ is viewed as a multiset (or $|A + B| = |A| \cdot |B|$).
[3] Given sets A, B, C do there exist $a \in A, b \in B, c \in C$ s.t $a + b + c = 0$.
[4] The k-SUM problem asks, given sets $A_1, \ldots, A_k$ of n numbers, whether there exist $a_1 \in A_1, \ldots, a_k \in A_k$ such that $\sum_{i=1}^{k} a_i = 0$. The k-SUM hypothesis states that for no $\epsilon > 0$ there exists a $O(n^{\lceil k/2 \rceil - \epsilon})$ time algorithm that solves k-SUM.

problems to k-SUM. Specifically, we shall need the existential fragment of the class $\mathsf{FOP}_{\mathbb{Z}}$: It consists of problems whose inputs are sets $A_1 \subseteq \mathbb{Z}^{d_1}, \ldots, A_k \subseteq \mathbb{Z}^{d_k}$, and free variables $t_1 \ldots t_\ell \in \mathbb{Z}$, and the task is to determine whether there exist $a_1 \in A_1 \ldots a_k \in A_k$ such that $\varphi(a_1[1], \ldots, a_1[d_1], \ldots, a_k[1], \ldots, a_k[d_k], t_1 \ldots, t_\ell)$ holds, where φ is a fixed, but arbitrary linear arithmetic formula over the vector entries of $a_1 \ldots a_k$ and the free variables. (In particular, the dimensions $d_1, \ldots, d_k$ are constant.) Each such problem is near-linear time reducible to k-SUM. We will frequently argue in our reductions that subtasks can be expressed as an existential $\mathsf{FOP}_{\mathbb{Z}}$ formula and exploit the completeness theorem.

Further related work

The authors in [9] explored connections of the $X + Y$ sorting problem with the $(\min, +)$-convolution problem. Other works consider the fine-grained complexity of other notions of succinct inputs, such as representing sequences succinctly as grammar-based compressions [2, 1], or succinctly defined graphs using the notion of factored graphs [25].

1.2 Our results

We investigate the succinct versions of many well-studied geometric primitives solvable in near-linear time. While some turn out to be easy, many others can be tightly connected to the k-SUM problem. In fact, we believe our results may serve as template to classify further problems according to their succinct complexity.

Some succinctly easy problems

For context, we observe that there is a large number of non-trivial geometric primitives (beyond selection) for which fast succinct computation is indeed possible.

As perhaps most prominent and well-known example, consider the convex hull problem: Given $A, B \subseteq \mathbb{Z}^2$, compute the convex hull $\mathsf{conv}(X)$ of $X = A + B$. By a folklore identity, $\mathsf{conv}(A + B) = \mathsf{conv}(A) + \mathsf{conv}(B)$, with complexity $|\mathsf{conv}(A)| + |\mathsf{conv}(B)|$. This yields $\tilde{O}(n)$-time algorithms for succinctly computing convex hulls, see also [17]. This fact leads us to the following general observation.

▶ **Observation 1.** *If we can compute $g(X)$ by a $T(m)$-time algorithm that receives as input only the convex hull of X of size m, then succinct-$g(X)$ can be solved in time $\tilde{O}(T(n))$.*

While the observation is simple, it leads to an interesting class of problems which can be solved in near-linear time in the size of the sumset compression. Some examples of problems which can be efficiently solved using only the convex hull of the input are the diameter of points, the maximum bounding box, the distance between two convex hulls and many more, described by Shamos in [36]. All these problems can be solved with a known technique originally introduced by Shamos [37, 36], which has subsequently been coined *rotating calipers* and further explored by Toussaint [39].

On Closest vs. Furthest Pair

Consider the problem Closest L_∞-Pair: Given $X \subseteq \mathbb{Z}^d$, determine $\min_{x_1, x_2 \in X, x_1 \neq x_2} \|x_1 - x_2\|_\infty$. Its natural maximization analogue is Furthest L_∞-Pair: Given $X \subseteq \mathbb{Z}^d$, determine $\max_{x_1, x_2 \in X} \|x_1 - x_2\|_\infty$. Both problems are solvable in near-linear time $\tilde{O}(N)$ whenever d is constant [22, 10].

However, we observe that the succinct versions differ substantially in their time complexity: Succinct-Furthest L_∞-Pair can be solved in time $\tilde{O}(n)$, while for succinct-Closest L_∞-Pair we obtain a 3-SUM-based lower bound of $N^{1-o(1)}$ already in 1D, using the fact that the EqDist problem is 3-SUM-hard [6]; see the full version of the paper.

Naturally, we may also consider the bichromatic case of the Closest L_∞-Pair problem: Given $X, Y \subseteq \mathbb{Z}^d$, determine $\min_{x\in X, y\in Y} \|x-y\|_\infty$. Here, the natural succinct version receives sets $A, B, C, D \subseteq \mathbb{Z}^d$ as inputs, and the task is to solve the problem for $X = A + B$ and $Y = C + D$. We prove that this succinct-Bichromatic L_∞-Pair problem is *fine-grained equivalent* to 4-SUM. The reduction to 4-SUM follows from observing that the decision problem can be formulated as an existential $\mathsf{FOP}_\mathbb{Z}$ formula, which can be reduced to 4-SUM [24]. It remains an interesting question whether the *monochromatic* succinct Closest-L_∞-Pair is fine-grained equivalent to 3-SUM (our results place it between 3-SUM and 4-SUM). For further fine-grained complexity results on Closest and Furthest Pair in higher dimensions; see e.g., [42] and [31].

Multivariate 3-SUM

The remaining problems that we study will exhibit a tight connection to a multivariate version of the 3-SUM problem. Specifically, we introduce the following problem:

▶ **Definition 2** (All-ints 3-SUM(n, n, k))**.** *Given sets $A, B \subseteq \mathbb{Z}$ of size n, and a set $C \subseteq \mathbb{Z}$ of size k. Determine for each c in C if there exists $(a, b) \in A \times B$ with $a + b = c$.*

It is easy to see that this problem can be solved in time $\tilde{O}(\min\{kn, n^2 + k\})$: The $O(kn)$ bound follows by solving, for each $c \in C$, the corresponding 2-SUM problem of detecting $a \in A, b \in B$ with $a + b = c$, which is well-known to be solvable in time $O(n)$. The $\tilde{O}(n^2 + k)$ upper bound follows from computing $A+B$ explicitly, and solving the corresponding dictionary problem for each $c \in C$. This time bound turns out to be conditionally tight; see Section 3: For any $k = \Theta(n^\alpha)$ with $0 < \alpha \leq 1$, we show using standard techniques that this problem cannot be solved in time $O((nk)^{1-\epsilon})$ with $\epsilon > 0$ unless the 3-SUM hypothesis is false. Note that this establishes a conditionally tight answer for all k: If $k = \Theta(n^\alpha)$ with $\alpha > 1$, then we obtain a $n^{2-o(1)}$ conditional lower bound via padding, and a $\Omega(k)$ lower bound follows from the input size of the problem.

Perhaps surprisingly, this problem naturally captures the precise fine-grained complexity of various succinct geometric primitives. Note that when $k = n$, All-ints 3-SUM(n, n, k) is well known to be subquadratic equivalent to 3-SUM [40], but a fine-grained equivalence for $k = \Theta(n^\alpha)$ with $\alpha < 1$ does not appear to follow from standard techniques. We will thus make use of a slight generalization of the completeness theorem for existential $\mathsf{FOP}_\mathbb{Z}$ formulas, which is implicit in [24]. Consider the following problems: Given $A \subseteq \mathbb{Z}^{d_1}, B \subseteq \mathbb{Z}^{d_2}, C \subseteq \mathbb{Z}^{d_3}$ of sizes n, n and k, respectively, compute for each $c \in C$, whether there exist $a \in A, b \in B$ such that a linear arithmetic formula $\varphi(a[1], \ldots, a[d_1], b[1], \ldots, b[d_2], c[1], \ldots, c[d_3])$ over the entries in the vectors a, b, c holds. For every linear arithmetic formula φ, we can reduce this problem to $\tilde{O}(1)$ instances of All-ints 3-SUM(n, n, k). A standalone proof of this fact can be found in the full version of the paper.

Succint orthogonal range queries

Consider a *batched* (or *offline*) version of answering orthogonal range queries over X, which we call Batched Orthogonal Range Searching: Given $X \subseteq \mathbb{Z}^d$ and orthogonal ranges $r_1, \ldots, r_k$, determine for each $j = 1, \ldots, k$ whether $X \cap r_j$ is empty. This problem can be solved in

time $\tilde{O}(N+k)$ for constant d, see, e.g. [12]. We observe that succinct-Batched Orthogonal Range Searching is fine-grained equivalent to All-ints 3-SUM(n,n,k). To reduce from All-ints 3-SUM(n,n,k), use A,B as succinct input sets for a 1-dimensional instance of succinct-Batched Orthogonal Range Searching and for $C=\{c_1,\ldots,c_k\}$ choose $r_i=[c_i,c_i]$. The other direction follows from observing that we may represent the succinct Orthogonal Range Searching problem as a suitable All-ints-$\mathsf{FOP}_{\mathbb{Z}}$ formula. The details can be found in the full version of the paper.

Succinct skyline (Pareto Sum)

Recall the prominent geometric problem of skyline computation (aka non-dominance filtering or maxima of point sets computation), see, e.g. [32, 14, 22]: Given $X\subseteq\mathbb{Z}^d$, compute the set Z of non-dominated points, i.e., all $x\in X$ for which there is no $x'\in X$ with $x'\neq x$ and $x\leq x'$ component-wise. This (explicit) problem can be solved in time $O(N\log k)$ [32], where $k=|Z|$ denotes the output size. For $d=2$, Hespe et al. show how to compute its succinct variant, the Pareto Sum, in time $O(n\log n+kn)$. We prove that computing all k non-dominated points of $X=A+B$ is fine-grained equivalent to All-ints 3-SUM(n,n,k).

Starting point of our reduction from All-ints 3-SUM(n,n,k) to the Pareto Sum is the subquadratic reduction from 3-SUM to Pareto Sum given in [24], which uses Conv3-SUM as an intermediate step. Notably, however, our multivariate setting poses a perhaps unexpected challenge: The standard reductions from 3-SUM to Conv3-SUM [35, 11] do not readily transfer to a multivariate setting. Instead, we introduce a variant of Conv3-SUM that we call All-ints-MixedConv3-SUM(n,n,k)[5]:

▶ **Definition 3** (All-ints-MixedConv3-SUM). *Given sequences $A[0],\ldots,A[k-1]$ and $B[0],\ldots,B[k-1]$, where each set $A[i],B[j]$ is a* set *of $O(n/k)$ integers, and a sequence of integers $C[0],\ldots,C[k-1]$, determine for each $0\leq i<k$ whether there exists some j and elements $a\in A[j],b\in B[i-j]$ such that $a+b=C[i]$.*

This problem can easily be solved in time $O(kn)$. We carefully adapt the reduction from 3-SUM to convolution 3-SUM given in [11] to reduce All-ints 3-SUM(n,n,k) to All-ints-MixedConv3-SUM(n,n,k). As a further technical challenge, our approach requires an equivalence between the problem of computing the complete Pareto Sum and computing the Pareto Sum inside a given query range. Specifically, in our lower bound, we need to allow as query ranges quadrants of the form $[x,\infty)\times[y,\infty)$; with this choice, even for sets A,B with a Pareto Sum as large as $\Omega(n^2)$ one may still give a query range with arbitrary small outputsize k, e.g., $k=O(1)$. Interestingly, using an involved construction detailed in Section 4.1, we show that this problem reduces to its specical case of computing the complete Pareto set under the promise that its size is $k+O(1)$. As a by-product, this also shows that for any $n\in\mathbb{N}$, there exist sets A,B of size n of incomparable vectors in $\mathbb{Z}^2$ with a Pareto Sum of constant size, which does not appear to have been proven before. After showing this equivalence, we reduce All-ints-MixedConv3-SUM to Pareto Sum, reminiscent of the reduction from Convolution 3-SUM to Pareto Sum in [24].

The reverse reduction from Pareto Sum to All-ints 3-SUM(n,n,k) is also technically interesting: while Hespe et al. [27] show how to compute each of the k output points successively in $\tilde{O}(n)$ time per point, unfortunately, we cannot follow this approach in our reduction (calling an All-ints 3-SUM(n,n,k)-oracle $\Omega(k)$ times would trivially exceed $\tilde{O}(kn)$

[5] The definition appears related in spirit (but different) to Factored 3-SUM defined in [16].

running time). Instead, we aim to identify the k output points essentially simultaneously. To this end, we perform a divide-and-conquer approach. We maintain a set of already identified Pareto Sum points S, as well as a list of up to k *active* x-intervals – an interval is deemed active if it contains at least one (unidentified) point of the Pareto Sum. At each level of the recursion, we split all active intervals into two halves. The key observation is the following: Given S and an arbitrary list of intervals $I_1, \ldots, I_s$, we can compute for each I_j whether it contains a point in the Pareto Sum (but not already in S) using only $\tilde{O}(1)$ All-ints $\mathsf{FOP}_{\mathbb{Z}}$ (n, n, s) queries, which in turn can be reduced to $\tilde{O}(1)$ All-ints 3-SUM(n, n, k) instances. After $O(\log n)$ levels of the recursion, each interval consists of at most a single integer, and can be resolved fully. The details can be found in Section 4.

Succinct independent set in unit interval graphs

Consider the maximum independent set problem (MIS) in unit interval graphs: Given a set of (open) unit intervals $\mathcal{I}$ in $\mathbb{Z}$, compute a maximum independent set S^*, i.e., a pairwise non-intersecting subset of intervals maximizing its size. A simple greedy algorithm is well known to solve the problem in time $O(n \log n)$, see, e.g. [26]. Taking the output size $k = |S^*|$ into account, this can be slightly improved to time $O(n \log k)$, see [38].

For studying its connection with All-ints 3-SUM(n, n, k), the technical work spent for the equivalence to Pareto Sum pays off: On the lower bound side, we establish a fine-grained reduction from All-ints 3-SUM(n, n, k) to succinct-MIS in unit interval graphs using a similar approach as for the Pareto Sum. In particular, we may reduce immediately from All-ints MixedConv3-SUM(n, n, k), and transfer conceptually similar arguments to the MIS setting.

On the upper bound side, we are not quite able to establish the reverse reduction. However, we show how to reduce computing a *maximal* independent set to All-ints 3-SUM(n, n, k), again exploiting careful parallel binary searches in a divide-and-conquer approach. It remains an interesting open problem to reduce the MIS problem in unit interval graphs to All-ints 3-SUM(n, n, k). The details can be found in the full version of the paper.

Summary

For the problems equivalent to All-ints 3-SUM(n, n, k), our results show that improving over decompress-and-solve running time $N^{1\pm o(1)}$ is possible whenever the output size k is small enough, specifically $k = O(N^{1/2-\epsilon}) = O(n^{1-\epsilon})$ for some $\epsilon > 0$. We refer to Figure 1 for an overview of the reductions shown. In the full version of the paper, we provide a table which summarizes runtimes for various problems, when defined succinctly and explicitly.

2 Preliminaries

We make use of basic tools in fine-grained complexity, for a comprehensive overview; see the survey [43]. For our purposes, the basic notion of *fine-grained reduction* is as follows. Consider problems P_1, P_2, with presumed time complexities T_1,T_2, we say there is a fine-grained reduction from P_1 to P_2 if for all $\epsilon > 0$ there exists $\delta > 0$ and an algorithm A solving P_1 with oracle access to P_2 such that if all calls to P_2 are replaced by an $O(T_2(n)^{1-\epsilon})$-time algorithm, then A runs in time $O(T_1(n)^{1-\delta})$. We say problems P_1, P_2 are fine-grained equivalent if there exist fine-grained reductions from P_1 to P_2 and P_2 to P_1. Throughout the paper, M will always denote a large number which we set as follows: With $A, B, C \subseteq \mathbb{Z}^d$, we let $M := 10 \cdot \max\{\|a\|_1 + \|b\|_1 + \|c\|_1 : a \in A, b \in B, c \in C\}$.

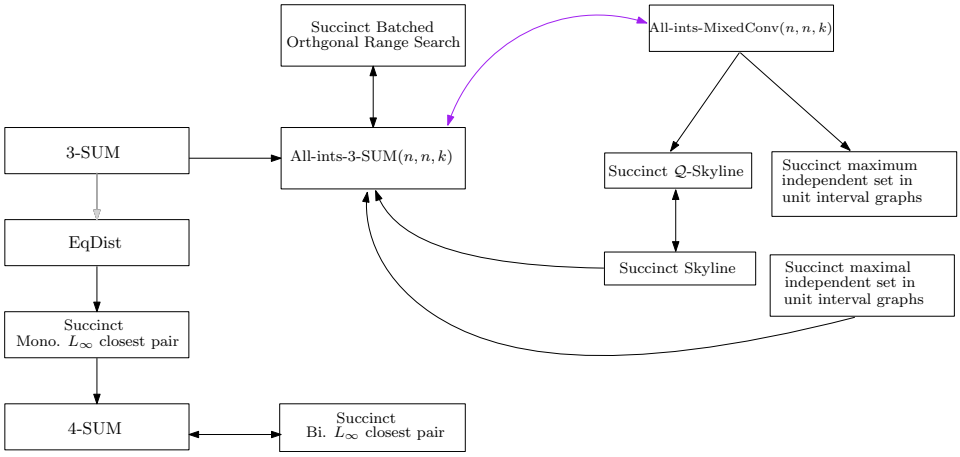


Figure 1 The above figure gives an overview of our most important results. The black arrows, denote the reductions, we have shown. The single gray arrow (reduction) from 3-SUM to the problem EqDist has been shown by Barequet and Har-Peled [6]. Furthermore, the violet arrow i.e the equivalence between All-ints 3-SUM(n, n, k) and All-intsMixedConv(n, n, k), holds only for the case, when $k \in \Theta(n^\alpha)$, where $\alpha \in (0, 1]$.

3 Multivariate 3-SUM variants

Let us recall two central problems for our results.

▶ **Definition 4** (All-ints 3-SUM(n, n, k))**.** *Given sets A, B of size n, and a set C of size k. Determine for each c in C if there exists $(a, b) \in A \times B$ with $a + b = c$.*

▶ **Definition 5** (All-ints-MixedConv3-SUM(n, n, k))**.** *Given sequences of sets $A[0] \dots A[k-1]$ and $B[0] \dots B[k-1]$ of size $O(n/k)$, and a sequence of integers $C[0], \dots, C[k-1]$, determine for every $C[i]$ if there exists an element $a \in A[j]$ and an element $b \in B[i-j]$ such that $a + b = C[i]$.*

In order to show an equivalence between All-ints 3-SUM(n, n, k) and All-ints-MixedConv3-SUM, we make use of a family of hash functions $h : [U] \to [m]$ defined by $h(x) := x \bmod p$ over a uniform random prime p selected in $[m/2, m)$. These hash functions satisfy the following properties.

- almost-linearity, that is $h(x + y) = h(x) + h(y)$ or $h(x + y) = h(x) + h(y) - p$.
- $O(\log(U))$-universality, that is $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] = O(\frac{\log(U)}{m})$.

Throughout, we assume that $A, B, C \subseteq [U]$ with $U = O(n^3)$ which can be achieved using a standard universe reduction. Furthermore, the above hash functions exhibit the following additional property.

▶ **Lemma 6** (Overfull buckets [11])**.** *Let $\mathcal{H}$ be a family of d-universal hash functions $[U] \to [m]$. Furthermore, let $h \in \mathcal{H}$ be chosen randomly. Let S be a set of n integers in $[U]$. We call x bad iff the number of elements equal to $h(x)$ (excluding x) exceeds a parameter t. The expected number of bad elements x in S is at most $\frac{dn}{m} \cdot \frac{1}{t} \cdot n$.*

We adapt the approach of Chan and He [11] (originally used to show a deterministic reduction from Convolution 3-SUM to 3-SUM) to show the following.

▶ **Lemma 7.** *Let $k = \Theta(n^\alpha)$, where $\alpha \in (0, 1]$. If we can solve All-ints-MixedConv3-SUM in time $O((nk)^{1-\epsilon})$ for some $\epsilon > 0$, then we can solve All-ints 3-SUM(n, n, k) in time $O((nk)^{1-\epsilon'})$ for some $\epsilon' > 0$.*

Proof. We reduce from All-ints 3-SUM(n, n, k). Let p be a prime number chosen from $[m/2, m)$ where $m = \frac{dk}{R}$ with $d = O(\log(U)) = O(\log(n))$ and R to be chosen later. Let $h(x) := x \bmod p$ and $C'[i] := \{c \in C : h(c) = i\}$, $A'[i] := \{a \in A : h(a) = i\}$, $B'[i] := \{b \in B : h(b) = i\}$. We call elements in $c \in C$ bad if $|C'[i]| > t$ where $i = h(c)$. Furthermore, we call elements $a \in A$ bad if $|A'[i]| > t \cdot n/m$ where $i = h(a)$. Similarly, we call elements $b \in B$ bad if $|B'[i]| > t \cdot n/m$ where $i = h(b)$.

The expected number of bad elements in C is $O\left(\frac{dk}{\frac{dk}{R}} \cdot \frac{1}{t}\right) \cdot k = O\left(\frac{Rk}{t}\right)$. The expected number of bad elements in A and B is $O\left(\frac{dn}{m} \cdot \frac{1}{\frac{n}{m} \cdot t} \cdot n\right) = O\left(\frac{nd}{t}\right)$.

By making use of Markov's inequality for the sets A, B, C separately, the probability that the number of bad A, B and C elements exceed 4 times its expectation is at most $1/4$ each. Thus by applying the union bound, there exists a hash function h, represented by some suitable prime p, where the number of bad elements for each set A, B, C is bounded by at most 4 times the number of expected bad elements.

We find such a hash function by going naively through all $O(m/\log(m))$ primes $p \in [m/2, m)$. The primes can be generated in time $\tilde{O}(m)$ by the sieve of Eratosthenes, and computing the bucket sizes takes time $O(m/\log(m)) \cdot O(n) = \tilde{O}(\frac{dkn}{R})$.

We proceed now as follows: For the bad elements in C, we perform a bruteforce 2-SUM algorithm, which needs time $O(n)$. There are at most $O(\frac{Rk}{t})$ many bad elements, thus this takes time $O(\frac{Rkn}{t})$.

For the bad elements in A and B, we also perform a simple bruteforce algorithm, which needs time $O(k)$, resulting in a runtime $O(k \cdot \frac{nd}{t}) = \tilde{O}(\frac{nk}{t})$.

For the remaining good elements in C, which occur at most t times, we proceed as follows: For each possible candidate in $[t]$, we perform an All-ints-MixedConv3-SUM instance for both cases of the almost-additiveness. For this, we get a runtime of $\tilde{O}(t \cdot (n \cdot \frac{k}{R})^{1-\epsilon})$. Concluding, we get a total runtime of

$$\tilde{O}\left(\frac{kn}{R}\right) + O\left(\frac{Rkn}{t}\right) + \tilde{O}\left(\frac{nk}{t}\right) + \tilde{O}\left(t \cdot (n \cdot \frac{k}{R})^{1-\epsilon}\right).$$

By setting $t = R^2$, we equate the first two terms, reducing to a runtime of:

$$\tilde{O}\left(\frac{kn}{R}\right) + \tilde{O}\left(\frac{nk}{R^2}\right) + \tilde{O}\left(R^2 \cdot \left(n \cdot \frac{k}{R}\right)^{1-\epsilon}\right),$$

which is bounded by $\tilde{O}\left(\frac{kn}{R}\right) + \tilde{O}\left(R^{1+\epsilon} \cdot (n \cdot k)^{1-\epsilon}\right)$. We balance the expression by setting $R = (nk)^{\epsilon/(2+\epsilon)}$, resulting in the final runtime $\tilde{O}((nk)^{1-\epsilon'})$, for an $\epsilon' > 0$. We remove the polylog factors by choosing an ϵ^* such that $0 < \epsilon^* < \epsilon'$, and get the desired runtime $O((nk)^{1-\epsilon^*})$. ◀

The reverse reduction is simpler and follows the standard reduction from convolution 3-SUM to 3-SUM.

▶ **Lemma 8.** *Let $k = \Theta(n^\alpha)$, where $\alpha \in (0, 1]$. If All-ints 3-SUM(n, n, k) can be solved in time $T(n, k)$, then we can solve All-ints MixedConv3-SUM(n, n, k) in time $O(T(n, k))$.*

Proof. We first convert the All-intsMixed-Conv3-SUM (n, n, k) problem into an All-ints 3-SUM (n, n, k) problem. Consider the newly created sets A', B', C', and a large number M as follows:

$$A' := \{a + iM : i \in \{0, \dots, k-1\} \wedge a \in A[i]\}$$
$$B' := \{b + iM : i \in \{0, \dots, k-1\} \wedge b \in B[i]\}$$
$$C' := \{c[i] + iM : i \in \{0, \dots, k-1\}\}$$

Due to the large choice of M it holds that there exist $a' \in A', b' \in B', c' \in C'$ such that $a' + b' = c'$ iff there exists $i, j, k \in \{0, \dots, k-1\}$ such that $i + j = k$ and there exists $a \in A[i], b \in B[j]$ such that $a + b = C[k]$. ◀

▶ **Lemma 9** (All-ints $\mathsf{FOP}_{\mathbb{Z}}$ queries [24]). *Let $A_1 \subseteq \mathbb{Z}^{d_1}, A_2 \subseteq \mathbb{Z}^{d_2}, A_3 \subseteq \mathbb{Z}^{d_3}$, with $|A_1| = k, |A_2| = |A_3| = n$, and $\varphi(a_1, a_2, a_3)$ be a quantifier free linear arithmetic formula. If we can solve All-ints 3-SUM(n, n, k) in time $T(n, k)$, then we can determine for each $a_1 \in A_1$, whether there exist $a_2 \in A_2, a_3 \in A_3$ such that $\varphi(a_1, a_2, a_3)$ is satisfied in time $\tilde{O}(T(n, k))$.*

The details can be found in the full version of the paper. We conclude with a simple 3-SUM lower bound for All-ints 3-SUM(n, n, k).

▶ **Lemma 10.** *Let $k = \Theta(n^\alpha)$ for $\alpha \in (0, 1]$. If we can solve All-ints 3-SUM (n, n, k) in time $O((nk)^{1-\epsilon})$ for some $\epsilon > 0$, then the 3-SUM hypothesis fails.*

Proof. We reduce from 3-SUM and separate the set C into $O(n/k)$ sets of size $O(k)$. We perform then $O(n/k)$ instances of All-ints 3-SUM(n, n, k) that run in time $O(n/n^\alpha) \cdot O((n \cdot n^\alpha)^{1-\epsilon}) = O(n^{2-\epsilon'})$ for some $\epsilon' > 0$. ◀

4 Output sensitive Skyline

We recall the two variants of the Pareto Sum problem.

▶ **Definition 11** (Succinct Skyline/Pareto Sum). *Given a succinctly defined set $X = A + B \subseteq \mathbb{Z}^2$, the skyline (aka Pareto front) of X consists of all non-dominated vectors in X. More formally compute the following set: $\mathsf{PS}(A, B) := \{x \in X : \text{There is no } x' \in X \setminus \{x\} : x \leq x'\}$, where we denote the output size by $k := |\mathsf{PS}(A, B)|$.*

▶ **Definition 12** ($\mathcal{Q}$-Succinct Skyline/$\mathcal{Q}$-Pareto Sum). *Given a succinctly defined set $X = A + B \subseteq \mathbb{Z}^2$, and a quadrant $\mathcal{Q} = [\alpha_1, \infty) \times [\alpha_2, \infty)$ the skyline (aka Pareto front) of X consists of all non-dominated vectors in X inside $\mathcal{Q}$. More formally compute the following set: $\mathsf{PS}_{\mathcal{Q}}(A, B) := \{x \in X = A + B : x \in \mathcal{Q} \text{ and there is no } x' \in X \setminus \{x\} : x \leq x'\}$, where again we denote the output size by $k := |\mathsf{PS}_{\mathcal{Q}}(A, B)|$.*

In the following section, we show that the above problems are fine-grained equivalent, which will be necessary to show the fine-grained equivalence between the Pareto Sum problem and All-ints 3-SUM(n, n, k).

4.1 On the equivalence of succinct skyline and $\mathcal{Q}-$succinct skyline

▶ **Theorem 13.** *If we can compute the Pareto Sum in time $T(n, k)$, then the $\mathcal{Q}$-Pareto Sum of an instance $A, B, \mathcal{Q}$ with $|A| = |B| = n$ and output size k can be computed in time $\mathcal{O}(T(n, k))$.*

Proof. Assume the quadrant to be given in the form $\mathcal{Q} = [L.x, \infty) \times [L.y, \infty)$. The goal is to construct a Pareto Sum instance where only Pareto points $c \in A + B$ with $c[0] \geq L.x$ and $c[1] \geq L.y$ are part of the output. The reduction consists of the following steps:

- Step 1: Transform the point coordinates in A, B such that the minimum coordinate value is at least 1 and adapt $L.x$ and $L.y$ accordingly. Define M larger than the largest possible point coordinate in $A + B$ and set $A' := A \cup A^+$ and $B' = B \cup B^+$ where $A^+ = \{(-5M + L.x, 7M), (-3M + L.x, 5M), (9M, -7M), (11M, -9M)\}$ and $B^+ = \{(-5M, 7M), (-M, 3M), (3M, -M)\}$.
- Step 2: Transform the point coordinates in A' and B' such that the minimum coordinate values are again at least 1 and adjust $L.y$ accordingly. Let M' be larger than the new maximum coordinate of any point in $A' + B'$. Augment A' with points A^{++} and B' with points B^{++}, which are derived from A^+, B^+ by swapping their x- and y-coordinates, respectively, and replacing M by M' and $L.x$ by (the transformed) $L.y$.
- Step 3: Compute the Pareto Sum C' of A', B'.
- Step 4: Delete all points with coordinates smaller than 1 or larger than M' from C', transform the coordinates of the remaining points in C' by reversing the transformations applied in Step 4, delete again elements with coordinate values smaller than 1 or larger than M and finally transform the coordinates once more by reversing the transformation applied in Step 2.

Applying the steps 1 and 2 takes time $\mathcal{O}(n)$ each. Step 3 takes time $T(\max\{|A'|, |B'|\}, |C'|)$ with $\max\{|A'|, |B'|\} = n+8$. Step 4 takes time in $\mathcal{O}(|C'|)$. Below, we argue that the returned result is correct and that $|C'| \in \mathcal{O}(k)$.

After the coordinate transformation in Step 1, the elements in A and B have coordinates in $\{1, \ldots, M-1\}$. Figure 2 illustrates the impact of adding A^+ and B^+ to A and B, respectively. Firstly, we observe that all elements $c = a + b$ with $a \in A^+$ or $b \in B^+$ – except for $A_2^+ + B_3^+$ – have either $c[0] < M$ or $c[1] < M$. Accordingly, none of these elements can dominate any element in $A + B$ as their maximum coordinate is bounded by M. The point $A_2^+ + B_3^+ = (L, 4M)$, however, dominates all points in $A + B$ with an x-coordinate of at most $L.x$. These points are all outside of $\mathcal{Q}$ based on our definition of $L.x$. Next, we observe that only 12 points $c = a + b$ with $a \notin A$ or $b \notin B$ are non-dominated, see again Figure 2. For cells having a colored point in Figure 2 we argue in the following that they dominate the corresponding colored ranges by comparing the x- and y-coordinates of the dominating point to the x- and y-coordinates of the dominated range.

- $A_1^+ + B_3^+ = (-2M + L, 6M)$ dominates $A_2^+ + B$ (yellow) since $-2M + L > -3M + L + 1$ and $-2M + L > -2M + L - 1$ for the x-coordinates and for the y-coordinates $6M > 6M - 1$ and $6M > 5M + 1$.
- $A_2^+ + B_2^+ = (-4M + L, 8M)$ dominates two ranges, namely $A_1^+ + B$ and $A + B_1^+$ (orange). The former is dominated since $-4M + L > -5M + L - 1$ and $-4M + L > -4M + L - 1$ hold for the x-coordinates and $8M > 8M - 1$ and $8M > 7M + 1$ for the y-coordinates. For the latter range we conclude that $-4M + L > -5M + 1$ and $-4M + L > -4M - 1$ for the x-coordinates and $8M > 8M - 1$ and $8M > 7M + 1$ for the y-coordinates.
- $A_2^+ + B_3^+ = (L, 4M)$ dominates $A + B_2^+$ (purple) since $L > -M + 1$ and $L > -1$ for the x-coordinates and $4M > 4M - 1$ and $4M > 3M + 1$ for the y-coordinates.
- $A_3^+ + B_1^+ = (4M, 0)$ dominates $A + B_3^+$ (light green) since $4M > 3M + 1$ and $4M > 4M - 1$ for the x-coordinates and $0 > -1$ and $0 > -M + 1$ for the y-coordinates.
- $A_3^+ + B_3^+ = (12M, -8M)$ dominates $A_4^+ + B$ (turquoise) since $12M > 11M + 1$ and $12M > 12M - 1$ for the x-coordinates and $-8M > -8M - 1$ and $-8M > -9M + 1$ for the y-coordinates.

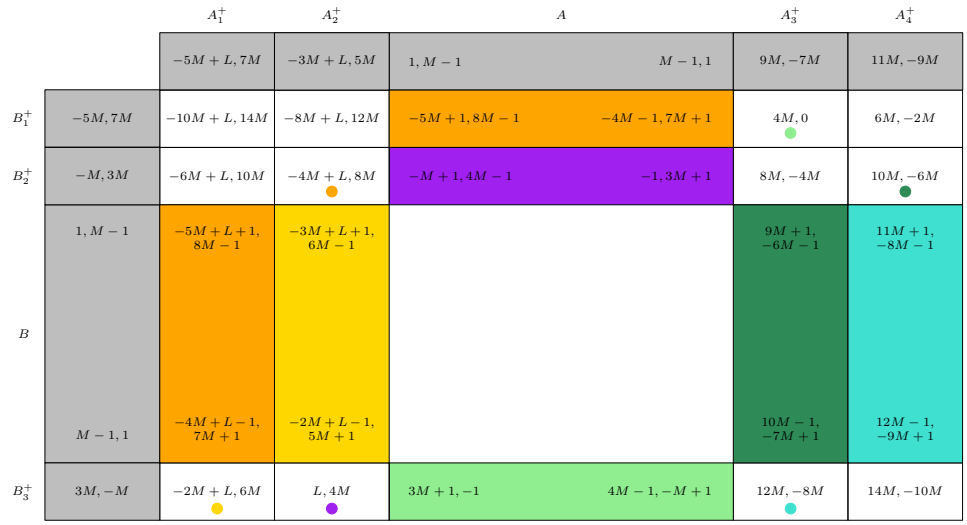


Figure 2 $\mathcal{Q}$-Pareto Sum instance augmented with A^+, B^+. The colored regions are dominated by the element in the cell with the dot of matching color. L refers to $L.x$. By definition, we have $L \leq M$.

- $A_4^+ + B_2^+ = (10M, -6M)$ dominates $A_3^+ + B$ (dark green) since $10M > 9M + 1$ and $10M > 10M - 1$ for the x-coordinates and $-6M > -6M - 1$ and $-6M > -7M + 1$ for the y-coordinates.

Step 2 repeats the very same process of coordinate transformation and augmentation, now with the goal to introduce a filter for the points with a y-coordinate smaller than $L.y$. The arguments from above apply again. Therefore, computing the Pareto Sum on A', B' in Step 3 results in C' of size at most $k + 24$. Thus, the filtering and back transformation in Step 4 takes time in $\mathcal{O}(k)$. Summing up over all steps, the running time is in $\mathcal{O}(n + k + T(n, k)) = \mathcal{O}(T(n, k))$.

After completing Step 4, C' is guaranteed to contain all points of the $\mathcal{Q}$-Pareto Sum C of A, B that are strictly inside $\mathcal{Q}$, as the points with an x-coordinate of at most $L.x$ and a y-coordinate of at most $L.y$ are dominated by the newly introduced elements in the augmented sumset, and Pareto points that result from the augmentation steps are filtered. ◀

We remark that the reduction in the other direction is trivial. Given an instance A, B of Pareto Sum, we can construct an instance of $\mathcal{Q}$-Pareto Sum with the same solution by enclosing all points in $A+B$ inside a large quadrant i.e $\mathcal{Q} = [-M, \infty) \times [-M, \infty)$. Accordingly, these two problems are fine-grained equivalent.

Moreover, our reduction technique can be leveraged to show that there are arbitrarily large Pareto Sum instances, where A and B consist of incomparable vectors, for which the output size k is constant. In prior work, it was only shown that for an instance with $|A| = 2$ and $|B| = 5$ an output size $k = 4 < \max(|A|, |B|)$ is possible [33]. But the question whether there exist sets A, B with $|A|, |B| = n$ and a Pareto Sum of size $k \in o(n)$ has been open. Given any Pareto Sum instance, we can define $L.x$ as the maximum x coordinate of any Pareto point in $A + B$. Using the augmentation described in Step 2 in the proof of Theorem 13, all Pareto points of $A + B$ will be dominated and only 12 new Pareto points will be introduced. Thus, we get the following corollary.

▶ **Corollary 14.** *For any $n \in \mathbb{N}$, there exist sets A, B of incomparable vectors with $|A| = |B| = n$ and output size of the Pareto Sum $k \in \mathcal{O}(1)$.*

4.2 A lower bound from All-ints 3-SUM(n, n, k)

▶ **Lemma 15.** *If we can compute the output-sensitive $\mathcal{Q}$-Pareto Sum of size $k = \Theta(n^\alpha)$ for $\alpha \in (0, 1]$ for sets A, B in time $O(T(n, k))$, then we can solve All-ints MixedConv3-SUM(n, n, k) in time $O(T(n, k))$.*

Proof. Consider an All-ints MixedConv3-SUM(n, n, k) instance given by array of sets $A[0, \ldots, k-1], B[0, \ldots, k-1]$ of size $O(n/k)$, and the array $C[0, \ldots, k-1]$. Let $A' := \{a + 100iM : i \in \{0, \ldots, k-1\} \wedge a \in A[i]\}$, $B' := \{b + 100iM : i \in \{0, \ldots, k-1\} \wedge b \in B[i]\}$ $C' := \{C[i] + 100iM : i \in \{0, \ldots, k-1\}\}$. It is clear that the newly constructed sets A', B' are of size at most n and the set C' of size at most k. Let $t := 2M$, and by the above construction, it holds that for an index $i \in \{0, \ldots, k-1\}$ there does not exist a $j \in \{0, \ldots, k-1\}$, where $j \leq i$ and $a \in A[j], b \in B[i-j]$ such that $a+b = C[i]$ if and only if for all $a' \in A', b' \in B'$ there exists $c' \in C'$, where one of the following two inequalities holds: $c' + 1 \leq a' + b' \leq c' + 1 + t$ or $c' - t - 1 \leq a' + b' \leq c' - 1$. Let us create the following set of the left side of the inequalities observed, that is $\tilde{C} := \bigcup_{c \in C'} \{c' + 1, c' - t - 1\}$.

We will construct the following sets in $\mathbb{Z}^2$, in order to integrate both above inequalities in a Pareto Sum problem.

$$A^* := \left\{ \begin{pmatrix} a \\ -a \end{pmatrix} : a \in A' \right\}, B^* := \left\{ \begin{pmatrix} b \\ -b \end{pmatrix} : b \in B' \right\}, C^* := \left\{ \begin{pmatrix} c + t \\ -c \end{pmatrix} : c \in \tilde{C} \right\}$$

We perform an output sensitive $\mathcal{Q}$-Pareto Sum computation on the following sets, with a quadrant $\mathcal{Q}$, which we determine in a bit.

$$H = A^* \cup \left(C^* + \left\{ \begin{pmatrix} 1000kM \\ -1000kM \end{pmatrix} \right\} \right), G = B^* \cup \left\{ \begin{pmatrix} -1000kM \\ 1000kM \end{pmatrix} \right\}$$

The sumset $H + G$ then consists of the following sumsets:

$$A^* + B^*,\ A^* + \left\{ \begin{pmatrix} -1000kM \\ 1000kM \end{pmatrix} \right\},\ C^* + B^* + \left\{ \begin{pmatrix} 1000kM \\ -1000kM \end{pmatrix} \right\},\ C^*.$$

We set the quadrant $\mathcal{Q} = [\min_{c \in C^*} c[0], \infty) \times [\min_{c \in C^*} c[1], \infty)$, thus forcing the output-sensitive $\mathcal{Q}$-Pareto Sum to ignore everything besides the points in $A^* + B^*$ and C^*.

Let us first argue that the points in C^* need to be part of the output-sensitive skyline. We argue based on the indices $i \in \{0, \ldots, k-1\}$, which separate the points in $A^* + B^*$ and C^* into clusters of points. For each $i, j \in \{0, \ldots, k-1\}$ the points in $A^* + B^*$ are of the form $\begin{pmatrix} a + 100iM + b + 100jM \\ -a - b - 100iM - 100jM \end{pmatrix} = \begin{pmatrix} (a+b) + (i+j)100M \\ -(a + b + (i+j)100M) \end{pmatrix}$. For each $i \in \{0, \ldots, k-1\}$ the two points originally induced by index i, now in C^* are of the form $\tilde{c}_i^1 := \begin{pmatrix} c[i] + 100iM + 1 + t \\ -(c[i] + 100iM + 1) \end{pmatrix}, \tilde{c}_i^2 := \begin{pmatrix} c[i] + 100iM - t - 1 + t \\ -(c[i] + 100iM - t - 1) \end{pmatrix} = \begin{pmatrix} c[i] + 100iM - 1 \\ -(c[i] + 100iM - t - 1) \end{pmatrix}$. For each $i \in \{0, \ldots, k-1\}$, we now observe the points which are included in the Pareto Sum. Clearly, the Pareto front, will contain $\tilde{c}_i^1$ as it is the point with the largest x-coordinate among the points landing at the cluster of points induced by index i. Furthermore, the point $\tilde{c}_i^2 = \begin{pmatrix} c[i] + 100iM - 1 \\ -(c[i] + 100iM - t - 1) \end{pmatrix}$ is contained in the Pareto Sum as it is the point with the highest y-coordinate among the points landing at the cluster of points induced by index i. Let us prove the following claim.

▷ Claim 16. The Pareto front of the cluster of points induced by index i will contain precisely the points $\{\tilde{c}_i^1, \tilde{c}_i^2\}$ if and only if there is no $j \in \{0, \ldots, k-1\}$ such that there exists an $a \in A[j]$ and $b \in B[i-j]$ with $a + b = C[i]$.

Proof. As argued before the points $\{\tilde{c}_i^1, \tilde{c}_i^2\}$, will always be in the Pareto Sum induced by index i. Assume there exists an index j such that there exists an $a \in A[j]$ and $b \in B[i-j]$ such that $a + b = C[i]$, then in the sumset $A^* + B^*$ there will be a point γ_i of the form:

$$\gamma_i = \begin{pmatrix} a + 100jM + b + 100(i-j)M \\ -(a + 100jM + b + 100(i-j)M) \end{pmatrix} = \begin{pmatrix} c[i] + 100iM \\ -(c[i] + 100iM) \end{pmatrix}.$$

The point $\tilde{c}_i^1 = \begin{pmatrix} c[i] + 100iM + 1 + t \\ -(c[i] + 100iM + 1) \end{pmatrix}$ does not dominate γ_i because of the y-coordinate. Furthermore, the point $\tilde{c}_i^2 = \begin{pmatrix} c[i] + 100iM - 1 \\ -(c[i] + 100iM - t - 1) \end{pmatrix}$ does not dominate γ_i because of the x-coordinate. Lastly, it is easy to see that the only points not dominated by $\tilde{c}_i^1$ or $\tilde{c}_i^2$ in the cluster of points induced by index i need to be of the form γ_i. Concluding, the Pareto front of the cluster of points induced by index i will contain precisely the points $\{\tilde{c}_i^1, \tilde{c}_i^2, \gamma_i\}$ if and only if there exists a $j \in \{0, \ldots, k-1\}$ and $a \in A[j]$ and $b \in B[i-j]$ with $a + b = C[i]$. ◁

After having proven the claim, it remains to scan through the Pareto Sum and identify whether the Pareto front induced by each index i consists of two points, concluding the reduction from All-ints MixedConv3-SUM(n, n, k). Lastly, notice that the size of the Pareto front evaluated is at most $O(k)$. ◀

By a simple padding argument we can now show the following lower bound.

▶ **Lemma 17.** *Let $\epsilon > 0$. There is no output-sensitive algorithm to compute the Pareto Sum of sets A, B of size n in time $O(n^{2-\epsilon})$ with guaranteed output size $\Theta(n^\alpha)$, where $1 \leq \alpha < 2$ and $k := n^\alpha$, unless All-ints 3-SUM(n, n, k) can be solved in time $O(n^{2-\epsilon'})$ for an $\epsilon' > 0$.*

Proof. Case $\alpha \leq 1$ is shown in Lemma 15, by making use of the fine-grained equivalence between All-ints 3-SUM(n, n, k) and All-ints-MixedConv-3-SUM(n, n, k).

We reduce from the case $\alpha = 1$, let $\beta = \alpha - 1$. We construct an artificial blowup of the Pareto Sum output set of size n^β. To this end, take a subset of B' of size n^β from the set B. Let the vectors in B' be numbered arbitrarily $b_1, \ldots, b_m$. Construct the following set:

$$\bar{B} := B \cup \underbrace{\bigcup_{i=1}^{m} \left\{ b_i + \begin{pmatrix} i \cdot M \\ -i \cdot M \end{pmatrix} \right\}}_{:=\tilde{B}}.$$

The Pareto Sum C will now consist of the original Pareto Sum of size n combined, with $A + \tilde{B}$ of size $n^{1+\beta}$. This holds due to the fact that all vectors in $A + \tilde{B}$ are incomparable to each other, and incomparable to the vectors $A + B$, due to the shift chosen for each vector. ◀

Let us now make use of the 3-SUM hardness of All-ints 3-SUM(n, n, k) to show the following lower bound, by a simple combination of Lemma 15 and Lemma 17.

▶ **Theorem 18.** *Let $k = \Theta(n^\alpha)$, where $\alpha \in (0, 2)$. There is no algorithm to solve output sensitive Pareto Sum of output size k, in time $O(\min\{n^2, nk\}^{1-\epsilon})$ for an $\epsilon > 0$, unless the 3-SUM hypothesis fails.*

Proof. We perform a simple case distinction depending on α, if $0 < \alpha \leq 1$ we apply Lemma 15 together with the fine-grained equivalence between All-ints 3-SUM(n, n, k) and All-ints MixedConv-3-SUM(n, n, k). In the case of $1 < \alpha < 2$, we similarly apply Lemma 17 together with the fact that All-ints 3-SUM(n, n, k) is 3-SUM hard. ◀

4.3 A reduction to All-ints 3-SUM(n, n, k)

▶ **Lemma 19.** *Given sets $A, B \subseteq \mathbb{Z}^2$ of size n and closed intervals $I_1 = [\alpha_1, \omega_1], \ldots, I_k = [\alpha_k, \omega_k]$. We can compute for all intervals I_j the highest point in $A + B$ with x-coordinate in I_j in time $\tilde{O}(T(n, k))$ if All-ints 3-SUM(n, n, k) can be solved in time $T(n, k)$.*

Proof. We consider a parallel binary search approach. Let $\gamma_1, \ldots, \gamma_k, \in [-M, M]$ and $C := \{(\alpha_i, \omega_i, \gamma_i) : i \in \{1, \ldots, k\}\}$. An application of All-ints$\mathsf{FOP}_{\mathbb{Z}}$ query to the set C on the formula

$$\exists a \in A \exists b \in B \exists c \in C : c[0] \leq a[0] + b[0] \wedge a[0] + b[0] \leq c[1] \wedge a[1] + b[1] \geq c[2],$$

gives us an answer for each interval I_j, where $j \in \{1, \ldots, k\}$, whether there exist points $a \in A, b \in B$ such that the x-coordinate of $a + b$ lies on the interval I_j and the y-coordinate of $a + b$ is higher than γ_j. By appropriately changing the value of γ_j, for each interval and constructing a new formula in a logarithmic number of steps, we can find the height y_j^* of the highest point in each interval I_j via a parallel binary search.

We now continue computing the x-coordinate of the highest point for each interval, which we can also compute by a binary search. Consider $\delta_1, \ldots, \delta_k$, where $\delta_i = \lfloor (\omega_i + \alpha_i)/2 \rfloor$ and construct the following set $C' = \{(\alpha_i, \omega_i, \delta_i, y_i^*) : i \in \{1, \ldots, k\}\}$. An application of All-ints $\mathsf{FOP}_{\mathbb{Z}}$ query to the set C' of the formula

$$\exists a \in A \exists b \in B \exists c' \in C' : c[0] \leq a[0]+b[0] \wedge a[0]+b[0] \leq c[1] \wedge a[0]+b[0] \geq c[2] \wedge a[1]+b[1] = c[3]$$

gives us an answer for each Interval I_j, whether there exists $a \in A, b \in B$ such that the x-coordinate of $a + b$ lies on or to the right of δ_j and the y-coordinate of $a + b$ is precisely the y-coordinate of the highest point in the interval I_j.

Thus, by appropriately changing the value of δ_i for each interval and constructing a new formula, in a logarithmic number of steps, we can find a corresponding x-coordinate to the highest y-coordinate in each interval via a parallel binary search. Notice that we can choose to binary search in such a way as to find for each interval the rightmost point, which equals the highest y-coordinate. Lastly, we can also deduce if there is no point $a + b$ inside an interval at all.

For the runtime notice that a parallel binary search on an All-ints $\mathsf{FOP}_{\mathbb{Z}}$ query runs in time $\tilde{O}(T(n, k)) \cdot O(\log U) = \tilde{O}(T(n, k))$ by Lemma 9, where $U \in O(poly(n))$ denotes the size of the universe. ◀

▶ **Theorem 20.** *If we can solve All-ints 3-SUM(n, n, k) in time $T(n, k)$, then we can compute an output sensitive Pareto Sum in time $\tilde{O}(T(n, k))$.*

Proof. We perform a divide and conquer approach, where we keep a collection of disjoint intervals $\mathcal{I}$, with the invariant that every interval $I \in \mathcal{I}$ contains at least one point the Pareto front of $A + B$ whose x-coordinate lies in I. Furthermore, keep a set S which stores the points of the skyline of $A + B$.

Start with the set of intervals given by one interval concluding the x-coordinate of all points, $\mathcal{I} := \{I_1 = [-M, M]\}$. We now state the general procedure.

Firstly, compute the highest points inside all intervals in $\mathcal{I}$ by an application of Lemma 19. In the beginning we directly, store this point in the set S as it must be part of the Pareto front of $A + B$.

Now, for each interval $I_j = [\alpha, \omega]$ in $\mathcal{I}$, we create two intervals $I_j^{(1)} := [\alpha, m-1]$ and $I_j^{(2)} := [m, \omega]$, where $m = \lfloor(\alpha+\omega)/2\rfloor$ is the middle x-coordinate of the interval I_j. Perform a call on Lemma 19 with A, B and the set of intervals $\mathcal{I}' := \{I_1^{(1)}, I_1^{(2)} \ldots, I_l^{(1)}, I_l^{(2)}\}$, to compute the highest points of the non-empty intervals. In the case of several highest points in an interval, Lemma 19 computes the rightmost highest point. We can now deduce for each of the intervals in $\mathcal{I}'$ if it contains at least one point in the Pareto front of $A+B$. Clearly if the interval is empty, we can disregard it and keep the non-empty half. Assume for I_j, we have that $I_j^{(1)}$ and $I_j^{(2)}$ are both non-empty, with the highest points being s_1 and s_2 respectively. If s_2 dominates the point s_1, we can disregard the interval $I_j^{(1)}$, as it can not be part of the Pareto front of $A+B$, any further subdivision of $I_j^{(1)}$ would create only more dominated points by monotonicity. Otherwise, we know $s_1[0] < s_2[0]$ and $s_1[1] > s_2[1]$. Let d_1 and d_2, be the rightmost highest points of the intervals $[m, M]$ and $[\omega+1, M]$ respectively, which can be computed by a call to Lemma 19 for all non-empty intervals $I_1^{(1)}, I_1^{(2)} \ldots, I_l^{(1)}, I_l^{(2)}$. We will keep the interval $I_j^{(1)}$ (store s_1 in S) iff d_1 does not dominate s_1, and will keep the interval $I_j^{(2)}$ (and store s_2 in S) iff the point d_2 does not dominate s_2. Continue recursively, with the set of non-disregarded intervals $\mathcal{I}'$, and the stored Pareto front points S, where we terminate the recursion and return the set S when all intervals in $\mathcal{I}$ are of length 1.

For the correctness, we remark that the invariant holds that every interval in $\mathcal{I}$ holds at least one point in the Pareto front. For the runtime, notice that the length of the intervals is halved in every recursion step. Finally, we have a recursion depth of at most $\log_2 U$, where $U \in O(poly(n))$ denotes the size of the universe, and in each recursion, we perform a call to Lemma 19 with at most $4k$ intervals (including the computation of d_1 and d_2). Thus, we get a final runtime of $\tilde{O}(T(n,k))$. ◀

5 Conclusion

We conclude with some open questions. It appears difficult to reduce All-ints 3-SUM(n,n,k) to 3-SUM when $k \ll n$, which would lead to a fine-grained equivalence between the problems. A first step might be to investigate if All-ints 3-SUM(n,n,k) admits an efficient self-reduction. Furthermore, it remains to classify further problems into being succinctly-easy or succinctly-equivalent to variants of k-SUM. Finally, it is interesting to study good representations of arbitrary sets X as sumsets $A+B$. In recent work [3], it was shown that determining perfect representability is NP-hard. However, perhaps we can efficiently compute approximate representations of sumsets, which may even lead to efficient (approximation) algorithms for explicit versions of various geometric primitives.

References

1 Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 192–203. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.26.

2 Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Impossibility results for grammar-compressed linear algebra. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/645e6bfdd05d1a69c5e47b20f0a91d46-Abstract.html.

3 Amir Abboud, Nick Fischer, Ron Safier, and Nathan Wallheimer. Recognizing sumsets is np-complete. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 4484–4506. SIAM, 2025. doi:10.1137/1.9781611978322.153.

4 Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2013. doi:10.1007/978-3-642-39206-1_1.

5 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.

6 Gill Barequet and Sariel Har-Peled. Polygon containment and translational min-hausdorff-distance between segment sets are 3sum-hard. *Int. J. Comput. Geom. Appl.*, 11(4):465–474, 2001. doi:10.1142/S0218195901000596.

7 Antonio Hernández Barrera. Finding an o(n2 log n) algorithm is sometimes hard. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 289–294. Carleton University Press, 1996. URL: http://www.cccg.ca/proceedings/1996/cccg1996_0048.pdf.

8 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

9 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Puatracscu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014. doi:10.1007/S00453-012-9734-3.

10 Timothy M. Chan. Orthogonal Range Searching in Moderate Dimensions: k-d Trees and Range Trees Strike Back. In Boris Aronov and Matthew J. Katz, editors, *33rd International Symposium on Computational Geometry (SoCG 2017)*, volume 77 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2017.27.

11 Timothy M. Chan and Qizheng He. Reducing 3sum to convolution-3sum. In Martin Farach-Colton and Inge Li Gørtz, editors, *3rd Symposium on Simplicity in Algorithms, SOSA 2020, Salt Lake City, UT, USA, January 6-7, 2020*, pages 1–7. SIAM, 2020. doi:10.1137/1.9781611976014.1.

12 Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the ram, revisited. In Ferran Hurtado and Marc J. van Kreveld, editors, *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, pages 1–10. ACM, 2011. doi:10.1145/1998196.1998198.

13 Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Hardness for triangle problems under even more believable hypotheses: reductions from real apsp, real 3sum, and OV. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1501–1514. ACM, 2022. doi:10.1145/3519935.3520032.

14 Wei-Mei Chen, Hsien-Kuei Hwang, and Tsung-Hsi Tsai. Maxima-finding algorithms for multidimensional samples: A two-phase approach. *Comput. Geom.*, 45(1-2):33–53, 2012. doi:10.1016/J.COMGEO.2011.08.001.

15 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

16 Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. New techniques for proving fine-grained average-case hardness. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 774–785. IEEE, 2020. doi:10.1109/FOCS46700.2020.00077.

17 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

18 Jeff Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM J. Comput.*, 28(4):1198–1214, 1999. doi:10.1137/S0097539797315410.

19 Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in X+Y and matrices with sorted columns. *J. Comput. Syst. Sci.*, 24(2):197–208, 1982. doi:10.1016/0022-0000(82)90048-4.

20 Michael L. Fredman. How good is the information theory bound in sorting? *Theor. Comput. Sci.*, 1(4):355–361, 1976. doi:10.1016/0304-3975(76)90078-5.

21 Daniel Funke, Demian Hespe, Peter Sanders, Sabine Storandt, and Carina Truschel. Pareto sums of pareto sets: Lower bounds and algorithms. *Algorithmica*, pages 1–34, 2025.

22 Harold N. Gabow, Jon Louis Bentley, and Robert Endre Tarjan. Scaling and related techniques for geometry problems. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 135–143. ACM, 1984. doi:10.1145/800057.808675.

23 Anka Gajentaan and Mark H. Overmars. On a class of o(n2) problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.

24 Geri Gokaj and Marvin Künnemann. Completeness theorems for k-sum and geometric friends: Deciding fragments of linear integer arithmetic. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference, ITCS 2025, January 7-10, 2025, Columbia University, New York, NY, USA*, volume 325 of *LIPIcs*, pages 55:1–55:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ITCS.2025.55.

25 Shreya Gupta, Boyang Huang, Russell Impagliazzo, Stanley Woo, and Christopher Ye. The computational complexity of factored graphs. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference, ITCS 2025, January 7-10, 2025, Columbia University, New York, NY, USA*, volume 325 of *LIPIcs*, pages 58:1–58:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ITCS.2025.58.

26 Udaiprakash I. Gupta, D. T. Lee, and Joseph Y.-T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12(4):459–467, 1982. doi:10.1002/NET.3230120410.

27 Demian Hespe, Peter Sanders, Sabine Storandt, and Carina Truschel. Pareto sums of pareto sets. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 60:1–60:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.60.

28 Donald B. Johnson and Samuel D. Kashdan. Lower bounds for selection in x + y and other multisets. *J. ACM*, 25(4):556–570, October 1978. doi:10.1145/322092.322097.

29 Donald B. Johnson and Tetsuo Mizoguchi. Selecting the kth element in X + Y and x_1 + x_2 + ... + x_m. *SIAM J. Comput.*, 7(2):147–153, 1978. doi:10.1137/0207013.

30 Haim Kaplan, László Kozma, Or Zamir, and Uri Zwick. Selection from heaps, row-sorted matrices, and X+Y using soft heaps. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 5:1–5:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASICS.SOSA.2019.5.

31 Karthik C. S. and Pasin Manurangsi. On closest pair in euclidean metric: Monochromatic is as hard as bichromatic. *Comb.*, 40(4):539–573, 2020. doi:10.1007/S00493-019-4113-1.

32 David G. Kirkpatrick and Raimund Seidel. Output-size sensitive algorithms for finding maximal vectors. In Joseph O'Rourke, editor, *Proceedings of the First Annual Symposium on Computational Geometry, Baltimore, Maryland, USA, June 5-7, 1985*, pages 89–96. ACM, 1985. doi:10.1145/323233.323246.

33 Kathrin Klamroth, Bruno Lang, and Michael Stiglmayr. Efficient dominance filtering for unions and minkowski sums of non-dominated sets. *Computers & Operations Research*, 163:106506, 2024. doi:10.1016/J.COR.2023.106506.

34 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1272–1287. SIAM, 2016. doi:10.1137/1.9781611974331.CH89.

35 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.

36 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985. doi:10.1007/978-1-4612-1098-6.

37 MI Shamos. Computational geometry[ph. d. thesis], 1978.

38 Jack Snoeyink. Maximum independent set for intervals by divide and conquer with pruning. *Networks*, 49(2):158–159, 2007. doi:10.1002/NET.20150.

39 Godfried T Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE Melecon*, volume 83, page A10, 1983.

40 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.67.

41 Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013. doi:10.1137/09076619X.

42 Ryan Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1207–1215. SIAM, 2018. doi:10.1137/1.9781611975031.78.

43 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*, pages 3447–3487. World Scientific, 2018.

Edge Clique Partition and Cover Beyond Independence

Fedor V. Fomin
University of Bergen, Norway

Petr A. Golovach
University of Bergen, Norway

Danil Sagunov
Saint Petersburg State University, Russia

Kirill Simonov
University of Bergen, Norway

Abstract

Covering and partitioning the edges of a graph into cliques are classical problems at the intersection of combinatorial optimization and graph theory, having been studied through a range of algorithmic and complexity-theoretic lenses. Despite the well-known fixed-parameter tractability of these problems when parameterized by the total number of cliques, such a parameterization often fails to be meaningful for sparse graphs. In many real-world instances, on the other hand, the minimum number of cliques in an edge cover or partition can be very close to the size of a maximum independent set $\alpha(G)$.

Motivated by this observation, we investigate *above-α* parameterizations of the edge clique cover and partition problems. Concretely, we introduce and study Edge Clique Cover Above Independent Set (ECC/α) and Edge Clique Partition Above Independent Set (ECP/α), where the goal is to cover or partition all edges of a graph using at most $\alpha(G) + k$ cliques, and k is the parameter. Our main results reveal a distinct complexity landscape for the two variants. We show that ECP/α is fixed-parameter tractable, whereas ECC/α is NP-complete for all $k \geq 2$, yet can be solved in polynomial time for $k \in \{0, 1\}$. These findings highlight intriguing differences between the two problems when viewed through the lens of parameterization above a natural lower bound.

Finally, we demonstrate that ECC/α becomes fixed-parameter tractable when parameterized by $k + \omega(G)$, where $\omega(G)$ is the size of a maximum clique of the graph G. This result is particularly relevant for sparse graphs, in which ω is typically small. For H-minor free graphs, we design a subexponential algorithm of running time $f(H)^{\sqrt{k}} \cdot n^{\mathcal{O}(1)}$.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Theory of computation → Fixed parameter tractability

Keywords and phrases edge clique partition, edge clique cover, independence number, parameterized complexity, above guarantee

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.43

Related Version *Full Version*: `https://arxiv.org/abs/2506.21216` [21]

Funding *Fedor V. Fomin*: Supported by the Research Council of Norway under BWCA project (grant no. 314528).
Petr A. Golovach: Supported by the Research Council of Norway under BWCA (grant no. 314528) and Extreme-Algorithms (grant no 355137) projects.



© Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Covering and partitioning the edges of a graph into cliques are fundamental combinatorial problems that lie at the intersection of combinatorial optimization, graph theory, and complexity theory. In the Edge Clique Cover (ECC) problem one seeks to cover all edges of a graph with as few cliques as possible, while in Edge Clique Partition (ECP) the same objective is pursued under the additional constraint that each edge must lie in exactly one of the chosen cliques. Both problems were extensively studied from various algorithmic perspectives (See Section 1.2 for an overview), in particular from parameterized algorithms.

ECC and ECP are known to be fixed-parameter tractable (FPT) when parameterized by the number of cliques in the cover or partition [24, 40]. While this parameterization is meaningful for dense graphs, it becomes less relevant for sparse graphs (e.g., graphs with bounded maximum degree or planar graphs), where the size of a clique cover or partition usually scales proportionally with the number of vertices. In such cases, a brute-force algorithm that guesses all possible partitions already runs in FPT time under this parameter choice.

These observations naturally lead us to explore a different parameterization – one that captures a lower bound on the size of the cover or partition. In particular, for any graph G without isolated vertices, an edge clique cover or partition must contain at least $\alpha(G)$ cliques, where $\alpha(G)$ denotes the size of a maximum independent set of G. Empirical studies of ECC (see, e.g., [29, Table 3]) indicate that in many real-world instances, after suitable preprocessing, the number of cliques in the cover is *very* close to $\alpha(G)$. This observation motivates us to investigate ECC and ECP in the setting of parameterization *above* α. Concretely, we initiate the study of the following parameterized problems.

Edge Clique Partition Above Independent Set (ECP/α) parameterized by k

Input:	A graph G without isolated vertices, an integer $k \geq 0$.
Task:	Find an edge clique partition of G with at most $\alpha(G) + k$ cliques.

Edge Clique Cover Above Independent Set (ECC/α) parameterized by k

Input:	A graph G without isolated vertices, an integer $k \geq 0$.
Task:	Find an edge clique cover of G with at most $\alpha(G) + k$ cliques.

While ECC and ECP are both FPT when parameterized by the number of cliques, their above-guarantee variants – ECC/α and ECP/α– exhibit notably different and intriguing behaviors.

1.1 Our Results

Our first result establishes parameterized complexity of Edge Clique Partition Above Independent Set (ECP/α). It is worth to note that the running time of our algorithm matches the best known running time of the algorithm for ECP [18]. Moreover, as we show, any improvement in the running time for ECP would improve the running time in our theorem too.

▶ **Theorem 1.** *ECP/α can be solved in $2^{\mathcal{O}(k^{3/2}\log k)} \cdot n^{\mathcal{O}(1)}$ time.*

We underline that the input of ECP/α (or ECC/α) does not contain the value of $\alpha(G)$. Interestingly, while computing the maximum independent set is well-known to be intractable, we show in Lemma 12 that, given an instance (G, k) of ECP/α, in $2^{\mathcal{O}(k^{3/2}\log k)} \cdot n^{\mathcal{O}(1)}$ time

we can either compute $\alpha(G)$ or conclude that (G, k) is a no-instance. This algorithm is used as a subroutine in Theorem 1 and, in fact, our algorithm for ECP/α either outputs an edge clique cover of size at most $\alpha(G) + k$ together with $\alpha(G)$ or correctly reports a no-instance.

While both ECP and ECC are FPT when parameterized by the number of cliques in the solution, their above-guarantee variants exhibit drastically different behaviors. The following theorem shows that for every fixed integer $k \geq 2$, deciding whether a graph can be covered by at most $\alpha(G) + k$ cliques is NP-complete even on prefect graphs for which the independence number can be computed in polynomial time [26]. Consequently, this places ECC/α in the class of Para-NP-complete problems.

▶ **Theorem 2.** *For every $k \geq 2$, ECC/α is NP-complete. Furthermore, the hardness holds even on perfect graphs.*

The condition in Theorem 2 concerning k is tight: for $k < 2$, the problem is solvable in polynomial time. Taken together, Theorems 2 and 3 establish a dichotomy result on the complexity of ECC/α for all values of k.

▶ **Theorem 3.** *ECC/α admits polynomial-time algorithms for $k \in \{0, 1\}$.*

Despite the intractability of ECC/α on general graphs, for certain classes of sparse graphs it is possible to design FPT algorithms. We summarize these findings in the following theorem. Here and further, the statements whose proof are omitted in this extended abstract are labeled ($\star$). The proofs can be found in the full version of this paper [21].

▶ **Theorem 4 ($\star$).** *ECC/α admits parameterized algorithms with the following running times:*

- $4^{\binom{\omega}{2} \cdot k} \cdot n^{\mathcal{O}(1)}$ *on graphs with clique number ω,*
- $2.081^{(d-1)k} \cdot n^{\mathcal{O}(1)}$ *on graphs of degeneracy $d \geq 3$,*
- $1.619^k \cdot n^{\mathcal{O}(1)}$ *on 2-degenerate graphs,*
- $f(H)^{\sqrt{k}} \cdot n^{\mathcal{O}(1)}$ *on H-minor-free graphs.*

Neither of ω, d, H should be given to the corresponding algorithm explicitly.

Complementing this result, we observe that, under the Exponential Time Hypothesis, the dependency on k cannot be improved in neither of the algorithms; additionally, the dependency on d is almost optimal. We refer the reader to the full version of this paper [21] for proper formal discussion.

1.2 Related Work

ECC and ECP are classical combinatorial problems [16, 28, 37, 46], whose origins can be traced back to fundamental questions posed by Boole in 1868 [7]. Over time, these problems have been studied under various names, including Covering by Cliques (Problem GT17) and Intersection Graph Basis (Problem GT59) in Garey and Johnson's compendium [23], as well as Keyword Conflict [34].

From a practical standpoint, ECC and ECP arise in diverse application areas such as computational geometry [2], applied statistics [25, 43], networks [27], compiler optimization [44], or bioinformatics [19]. Due to their broad importance, both ECC and ECP have been investigated from multiple perspectives, including approximation algorithms and inapproximability bounds [3, 38], heuristics [4, 25, 34, 36, 43, 44], and polynomial-time solutions for special graph classes [8, 23, 30, 31, 32, 39, 42].

Moreover, the relation between the smallest edge clique cover and the maximum independent set plays another important role in practical applications. On large instances, it may be infeasible to compute the optimal edge clique cover, and only heuristical methods

could be applied within a reasonable timeframe. Finding a suffiently large independent set serves then as a certificate for the quality of the solution, in terms of the size of the edge clique cover: If, say, there is only a 1% difference between the found independent set and clique cover, then the smallest clique cover is also bound to lie in this small interval. As observed by Hevia et al. [29], this phenomenon occurs frequently on real-world instances. Simultaneously, this certifies that the difference between the smallest edge clique cover and the largest independent set is also small on these instances.

Both ECC and ECP are known to be NP-complete even in restricted graph classes [42, 9, 30]. In particular, ECC remains NP-complete and even when the input graph is planar [9] or has bounded degree [30]. Furthermore, Lund and Yannakakis [38] showed that ECC is not approximable within a factor of n^{ε} (for some $\varepsilon > 0$) unless P=NP.

In particular, a significant amount of work in the algorithms engineering community is devoted to the studies of ECC on sparse graphs, whose independence number is large. For example, Abdullah and Hossain [1], and Conte, Grossi, and Marino [11] studied ECC on d-degenerate graphs. Blanchette, Kim, and Vetta in [6] obtained an PTAS for ECC on planar graphs as well as an FPT algorithm parameterized by the treewidth of a graph.

Within the realm of parameterized complexity, a natural parameterization is by the number of cliques k. Gramm et al. [24] pioneered this direction by presenting simple reduction rules that produce a kernel of size at most 2^k, making it one of the earliest examples of kernelization techniques described in textbooks [12, 22, 41]. Later, Cygan, Pilipczuk, and Pilipczuk [13] showed that there is no algorithm solving ECC in time $2^{2^{o(k)}} \cdot n^{\mathcal{O}(1)}$ unless the Exponential Time Hypothesis (ETH) fails.

Mujuni and Rosamond [40] established that ECP is FPT when parameterized by k, the number of cliques in the solution. Feldmann, Issac, and Rai [18] improved the running time to $2^{k^{3/2}\log k} \cdot n^{\mathcal{O}(1)}$. Fleischer and Wu [20] devised an algorithm for planar graphs running in $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$, and for graphs of degeneracy d they achieved $2^{dk} \cdot k \cdot n^{\mathcal{O}(1)}$ running time.

1.3 Organization of the Paper

In Section 2, we introduce basic notions and provide auxiliary results. Section 3 contains the proof of Theorem 1. In Section 4, we show the dichotomy for ECC/α on general graphs, proving Theorems 2 and 3. Finally, in Section 5, we give algorithmic results and establish computational lower bounds for ECC/α on sparse graph classes, as stated in Theorem 4. Due to the space restrictions, we omit proofs of some of the results. Several parts of Section 5 are also omitted. The omitted proofs and parts can be found in the full version of this paper [21].

2 Definitions and Preliminaries

Graphs. In this paper, we consider simple undirected graphs and refer to the textbook by Diestel [15] for notions that are not defined here. We use $V(G)$ and $E(G)$ to denote the set of vertices and the set of edges of G, respectively. We use n and m to denote the number of vertices and edges in G, respectively, unless this creates confusion. For a vertex subset $X \subseteq V(G)$, we use $G[X]$ to denote the subgraph of G induced by the vertices of X and $G - X$ to denote $G[V(G) \setminus X]$. For a vertex $v \in V(G)$, we write $N_G(v) = \{u \in V(G) \mid uv \in E(G)\}$ to denote the *open neighborhood* of v, and we use $N_G[v] = N_G(v) \cup \{v\}$ for the *closed neighborhood*. For $X \subseteq V(G)$, $N_G(X) = \left(\bigcup_{v \in X} N_G(v)\right) \setminus X$ and $N_G[X] = \bigcup_{v \in X} N_G[v]$. We denote by $d_G(v) = |N_G(v)|$ the *degree* of a vertex v; a vertex of degree zero is *isolated*. The minimum degree is denoted by $\delta(G)$. A graph G is *d-degenerate* for an integer $d \geq 0$ if

$\delta(H) \leq d$ for any subgraph H of G. By $\mathrm{dg}(G)$ we denote the degeneracy of G, i.e. minimum d for which G is d-degenerate. A graph H is a *minor* of G if the graph isomorphic to H can be obtained from G by vertex/edge deletions and edge contractions. A graph G is *H-minor-free* if H is not a minor of G. A set of pairwise adjacent vertices is called a *clique*, and a set of pairwise non-adjacent vertices is *independent*. The maximum size of an independent set in G is denoted by $\alpha(G)$, and the maximum size of a clique in G is denoted by $\omega(G)$. A vertex $v \in V(G)$ is said to be *simplicial* if $N_G[v]$ is a clique, and we say that a clique K is *simplicial* if K equals $N_G[v]$ for a simplicial vertex v. An edge $uv \in E(G)$ is *covered* by a clique K if $u, v \in K$. A set $\mathcal{K}$ of cliques is a *vertex clique cover* if for every $v \in V(G)$, there is $K \in \mathcal{K}$ such that $v \in K$. We say that $\mathcal{K}$ is an *edge clique cover* if for every $uv \in E(G)$, there is $K \in \mathcal{K}$ with $u, v \in K$, and $\mathcal{K}$ is an *edge clique partition* if $\mathcal{K}$ is an edge clique cover such that any two cliques in $\mathcal{K}$ has at most one common vertex. We use $\mathrm{ecc}(G)$ and $\mathrm{ecp}(G)$ to denote the minimum size of an edge clique cover and partition of G, respectively.

Parameterized Complexity. We refer to the textbook by Cygan et al. [12] for an introduction to the area. We remind that a *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$ where Σ^* is a set of strings over a finite alphabet Σ. An input of a parameterized problem is a pair (x, k) where x is a string over Σ and $k \in \mathbb{N}$ is a *parameter*. A parameterized problem is *fixed-parameter tractable* (or FPT) if it can be solved in $f(k) \cdot |x|^{\mathcal{O}(1)}$ time for some computable function f. The complexity class FPT contains all fixed-parameter tractable parameterized problems. We use the *Exponential Time Hypothesis* (ETH) of Impagliazzo, Paturi, and Zane [33] to obtain conditional computational lower bounds. Under ETH, there is a positive real δ such that 3-Satisfiability (3-SAT) with n variables and m clauses cannot be solved in $2^{\delta n} \cdot (n+m)^{\mathcal{O}(1)}$ time; in particular, 3-SAT cannon be solved in time subexponential in n.

We need the following auxiliary results about edge clique covers and independent sets. First, we observe that $\alpha(G)$ gives a lower bound for $\mathrm{ecc}(G)$ and $\mathrm{ecp}(G)$.

▶ **Observation 5** ($\star$). *For a graph G without isolated vertices $\alpha(G) \leq \mathrm{ecc}(G) \leq \mathrm{ecp}(G)$.*

Notice that the absence of isolated vertices is crucial for the lower bound of $\mathrm{ecc}(G)$ and $\mathrm{ecp}(G)$. We also remark that that excluding isolated vertices is essential for our algorithmic results for ECP/α and ECP/α.

▶ **Observation 6** ($\star$). *For any graph class $\mathcal{G}$ closed under adding pendent neighbors and isolated vertices such that ECP (ECC) is* NP*-complete on $\mathcal{G}$, it is* NP*-complete to decide whether a graph $G \in \mathcal{G}$ given together with its maximum independent set has an edge clique partition (cover) of size at most $\alpha(G)$.*

The following lemma plays the most fundamental role for ECC/α and ECP/α.

▶ **Lemma 7.** *Let G be a graph without isolated vertices, and let $\mathcal{C}$ be an edge clique cover (partition) of G of size at most $\alpha(G) + k$ for an integer $k \geq 0$. Then*

(i) *at most k vertices of any maximum independent set are non-simplicial,*

(ii) *at most $2k$ cliques of $\mathcal{C}$ are non-simplicial.*

Proof. As an edge clique partition is an edge clique cover, it is sufficient to show the claim when $\mathcal{C}$ is an edge clique cover of size at most $\alpha(G)+k$. Let X be a maximum independent set of G. Clearly, any clique of G contains at most one vertex of X by independence. Because G has no isolated vertices, each vertex $v \in X$ is included in at least one clique of $\mathcal{C}$. Furthermore, v is simplicial if v is included in a single clique of $\mathcal{C}$. Since $|\mathcal{C}| \leq \alpha(G) + k$, we obtain that at least $\alpha(G) - k$ vertices of X are contained in exactly one clique of $\mathcal{C}$. Therefore, at least

$\alpha(G) - k$ vertices of X are simplicial, and at least $\alpha(G) - k$ cliques of $\mathcal{C}$ are simplicial. This means that at most k vertices of X are non-simplicial and proves (i). To show (ii), note that because at least $\alpha(G) - k$ cliques of $\mathcal{C}$ are simplicial, at most $2k$ cliques could be non-simplicial. This concludes the proof. ◀

In particular, this lemma implies that ECC/α and ECP/α are trivial for $k = 0$.

▶ **Observation 8** ($\star$). *ECC/α and ECP/α can be solved in polynomial time for $k = 0$.*

We use Lemma 7 to argue that for graphs admitting an edge clique cover of bounded size, we can compute $\alpha(G)$.

▶ **Lemma 9** ($\star$). *Let G be a graph and let $\mathcal{C}$ be an edge clique cover (partition) of G of size at most k. Then $\alpha(G)$ can be computed in $\mathcal{O}(2^k \cdot k^2 n)$ time.*

We remark that our algorithm from Lemma 9 can be easily modified to output an independent set of maximum size.

3 Clique Partition above Independent Set

In this section, we prove Theorem 1. For this, we need several auxiliary results. We start with the classical result of de Bruijn and Erdős from 1948 [14].

▶ **Proposition 10** (de Brujin and Erdős [14]). *For any $n \geq 3$, an edge clique partition of K_n into cliques of size at most $n - 1$ contains at least n cliques.*

To solve ECP/α, we use as a black box an algorithm for ECP parameterized by the number of cliques in a solution. The state-of-the-art FPT algorithm was given by Feldmann, Issac, and Rai [18].

▶ **Proposition 11** (Feldmann, Issac, Rai [18]). *ECP can be solved in $2^{\mathcal{O}(k^{3/2}\log k)} \cdot n^{\mathcal{O}(1)}$ time where k is the maximum number of cliques in a solution. Furthermore, the algorithm outputs an edge clique partition of size at most k if it exists.*

Next, we show that combining Lemma 7, Lemma 9, and Proposition 11, we can compute $\alpha(G)$ for graphs admitting an edge clique partition of size at most $\alpha(G) + k$.

▶ **Lemma 12.** *There is an algorithm with running time $2^{\mathcal{O}(k^{3/2}\log k)} \cdot n^{\mathcal{O}(1)}$ that, given an instance (G, k) of ECP/α, either computes $\alpha(G)$ or correctly reports that (G, k) is a no-instance of ECP/α.*

Proof. Let G be a graph without isolated vertices. We compute the set $\mathcal{S}$ of all simplicial cliques and select a simplicial vertex in each simplicial clique. Denote by X the obtained set of simplicial vertices. It is well-known that G has a maximum independent set U such that $X \subseteq U$. This implies that $\alpha(G) = \alpha(H) + |\mathcal{S}|$ where $H = G[W]$ for $W = V(G) \setminus \bigcup_{S\in\mathcal{S}} S$. If G admits an edge clique partition of size at most $\alpha(G) + k$ then, by Lemma 7, at most $2k$ cliques of the partition are non-simplicial. Notice that the edges of H can be covered only by non-simplicial cliques. Therefore, H should have an edge clique partition of size at most $2k$. We apply the algorithm from Proposition 11 to H and check in $2^{\mathcal{O}(k^{3/2}\log k)} \cdot n^{\mathcal{O}(1)}$ time whether H has an edge clique partition of size at most $k' = 2k$. If the algorithm reports that such a partition does not exist then we conclude that (G, k) is a no-instance of ECP/α. Otherwise, the algorithm outputs an edge clique partition $\mathcal{C}$ of size at most $2k$. Given this partition, we use the algorithm from Lemma 9 to compute $\alpha(H)$ in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. Then

we output $\alpha(G) = \alpha(H) + |\mathcal{S}|$. Because simplicial cliques can be found in polynomial time, for example, by the algorithm of Kloks, Kratsch, and Müller [35], the overall running time is $2^{\mathcal{O}(k^{3/2}\log k)} \cdot n^{\mathcal{O}(1)}$. This concludes the proof. ◀

By the following claim, which is proved by making use of Proposition 10, we show that all cliques of size at least $6k+1$ that belong to any solution to an instance of ECP/α can be found in polynomial time.

▶ **Lemma 13** ($\star$). *There is a polynomial algorithm that, given an instance (G,k) of ECP/α for $k \geq 1$, either outputs a set $\mathcal{K}$ of cliques of G such that*

- **(i)** *each clique $C \in \mathcal{K}$ is included in any edge clique partition of G of size at most $\alpha(G)+k$,*
- **(ii)** *for every edge clique partition $\mathcal{C}$ of G of size at most $\alpha(G)+k$ and any clique $C \in \mathcal{C}$ of size at least $6k+1$, $C \in \mathcal{K}$,*
- **(iii)** *for any two distinct cliques $C_1, C_2 \in \mathcal{K}$, $|C_1 \cap C_2| \leq 1$,*

or correctly concludes that (G,k) is a no-instance.

Now we are ready to show Theorem 1 which we restate.

▶ **Theorem 1.** *ECP/α can be solved in $2^{\mathcal{O}(k^{3/2}\log k)} \cdot n^{\mathcal{O}(1)}$ time.*

Proof. Let (G,k) be an instance of ECP/α. As pointed out in Observation 8, the problem is trivial if $k=0$. Hence, we assume that $k \geq 1$. First, we call the algorithm from Lemma 12 for (G,k). The algorithm either computes $\alpha(G)$ or correctly reposts that (G,k) is a no-instance of ECP/α. If we obtain a no-instance, we report it and stop. We assume from now on that this is not the case, and the value $\alpha(G)$ is known to us.

In the following step, we construct a partial solution $\mathcal{K}$ using the algorithm from Lemma 13. This algorithm works in polynomial time and either outputs a set $\mathcal{K}$ of cliques of G such that

- **(i)** each clique $C \in \mathcal{K}$ is included in any edge clique partition of G of size at most $\alpha(G)+k$,
- **(ii)** for every edge clique partition $\mathcal{C}$ of G of size at most $\alpha(G)+k$ and any clique $C \in \mathcal{C}$ of size at least $6k+1$, $C \in \mathcal{K}$,
- **(iii)** for any two distinct cliques $C_1, C_2 \in \mathcal{K}$, $|C_1 \cap C_2| \leq 1$,

or correctly concludes that (G,k) is a no-instance. If the algorithm returns that (G,k) is a no-instance then we return the answer and stop. We also conclude that (G,k) is a no-instance if $|\mathcal{K}| > \alpha(G)+k$. From now on, we assume that $\mathcal{K}$ is a set of cliques that are included in any edge clique partition of size at most $\alpha(G)+k$ such that any two distinct cliques of $\mathcal{K}$ do not cover the same edge.

Next, we list all distinct simplicial cliques of G and denote by $\mathcal{S}$ the set of simplicial cliques. We say that a clique $S \in \mathcal{S}$ is *broken* in a (potential) solution to (G,k), if S is not included in the edge clique partition. Notice that if S is a broken simplicial clique, then at least two distinct non-simplicial cliques should be used to cover the edges incident to the corresponding simplicial vertex. By Lemma 7, any edge clique partition of size at most $\alpha(G)+k$ can include at most $2k$ non-simplicial cliques. Thus, for any solution, the number of broken simplicial cliques should not exceed k. Our algorithm identifies the set $\mathcal{B}$ of broken cliques in a solution.

We initialize $\mathcal{B}$ using the following branching algorithm. Consider the auxiliary graph H whose nodes are simplicial cliques. Two distinct nodes $S_1, S_2 \in \mathcal{S}$ are adjacent in H if and only if $|S_1 \cap S_2| \geq 2$. Observe that if $|S_1 \cap S_2| \geq 2$ then either S_1 or S_2 should be broken in a solution. Thus, the set of broken cliques in any solution should be a vertex cover of H of size at most k. We can decide in $\mathcal{O}(2^k \cdot n)$ time whether H has a vertex cover of size at most k and, moreover, we can list all inclusion minimal vertex covers using the standard recursive branching algorithm for Vertex Cover (see [12]). If H has no vertex cover of

size at most k, we report that (G,k) is a no-instance of ECP/α and stop. Assume that this is not the case. Then we obtain the list $\mathcal{L}$ of all inclusion minimal vertex covers of H of size at most k. Notice that $|\mathcal{L}| \leq 2^k$. Then if (G,k) admits an edge clique partition of size at most $\alpha(G)+k$, there is $L \in \mathcal{L}$ such that $L \subseteq \mathcal{B}$ – the set of broken cliques in $\mathcal{C}$. To initialize $\mathcal{B}$, we branch on all possible choices of $L \in \mathcal{L}$ and set $\mathcal{B} := L$. If we find a solution for some choice of L, we output it and stop. Otherwise, if we fail to find an edge clique partition of size at most $\alpha(G)+k$ for any initial selection of $\mathcal{B}$, we report that (G,k) is a no-instance of ECP/α.

From now on, we assume that the initial choice of $\mathcal{B}$ is fixed from one of at most 2^k choices given by $\mathcal{L}$. We update $\mathcal{B}$ by adding cliques that get broken because of the cliques in the partial solution. Notice that if $S \notin \mathcal{K}$ is a simplicial clique covering the same edge as one of the cliques in the partial solution, then S has to be broken. This implies the correctness of the following step:

- For every simplicial clique $S \notin \mathcal{B} \cup \mathcal{K}$ such that $|S \cap K| \geq 2$ for some $K \in \mathcal{K}$, set $\mathcal{B} := \mathcal{B} \cup \{S\}$.
- If $|\mathcal{B}| > k+1$ then stop and discard the current choice of L.

Assume that the algorithm does not stop in this step. We call the set $\mathcal{B}$ obtained in this step a *base set of broken cliques*.

We say that a simplicial clique S is *free* if $S \notin \mathcal{B} \cup \mathcal{K}$, and we use $\mathcal{F}$ to denote the set of free cliques. Lemma 13 and the construction of $\mathcal{B}$ imply the following property.

▷ Claim 14. For any two distinct cliques $S_1, S_2 \in \mathcal{F} \cup \mathcal{K}$, $|S_1 \cap S_2| \leq 1$.

This means that no edge of G is covered by two cliques of $\mathcal{F} \cup \mathcal{K}$. In the final step of our algorithm, we try to extend $\mathcal{F} \cup \mathcal{K}$ to an edge clique partition of G of size at most $\alpha(G)+k$. If we fail, then we recurse by including one of the free cliques to the set of broken cliques. Formally, we use the subroutine Extend$(\mathcal{F},\mathcal{B})$ in Algorithm 1 which either finds a solution extending $\mathcal{K}$, such that all cliques in $\mathcal{B}$ are broken, or discards the current choice of $\mathcal{F}$ and $\mathcal{B}$.

The description of the algorithm is finished. To argue correctness, observe that in each recursive call of the algorithm, we increase the size of $\mathcal{B}$ and stop if $|\mathcal{B}| > k$ in line 2. This means that Extend$(\mathcal{F},\mathcal{B})$ is finite. If this subroutine outputs a set of cliques $\mathcal{C}^*$ then it is a valid edge partition of G. First, because of Claim 14, no two cliques intersect. Second, the cover $\mathcal{C}$, constructed by the algorithm from Proposition 11 in line 5, outputs an edge clique partition for the edges which are not covered by the cliques of $\mathcal{F} \cup \mathcal{K}$. Since we verify that $|\mathcal{C}^*| \leq \alpha(G)+k$ in line 7, if the algorithm outputs some $\mathcal{C}^*$ then it is a valid solution to (G,k). Thus, we have to show that if (G,k) is a yes-instance of ECP/α, then the algorithm necessarily finds a solution. For this, we show the following claim.

▷ Claim 15 ($\star$). Suppose that G has an edge clique partition of size at most $\alpha(G)+k$ such that all simplicial cliques in $\mathcal{B}$ are broken. Suppose also that $\mathcal{B}$ contains the base set of broken cliques, and let $\mathcal{F}$ be the set of simplicial cliques S such that $S \notin \mathcal{B} \cup \mathcal{K}$. Then Extend$(\mathcal{F},\mathcal{B})$ outputs a solution to (G,k).

To complete the correctness proof, observe that if (G,k) is a yes-instance of ECP/α then there is $L \in \mathcal{L}$ such that every clique of L is broken in a solution. Then for the base set of broken cliques $\mathcal{B}$ constructed for L, each clique of $\mathcal{B}$ should be broken. By Claim 15, Extend$(\mathcal{F},\mathcal{B})$ called for this $\mathcal{B}$ and the corresponding set of free cliques $\mathcal{F}$ should output an edge clique partition of size at most $\alpha(G)+k$. This completes the correctness proof.

We conclude the proof with a claim explaining the running time bound.

Algorithm 1 EXTEND$(\mathcal{F}, \mathcal{B})$.

Input: $\mathcal{F}$, $\mathcal{B}$
Result: An edge clique cover of G or No
begin
  if $|\mathcal{B}| > k$ **then return** No and quit;
  construct the graph Q with the edge set $R = \{uv \in E(G) \mid$ there is no clique $K \in \mathcal{F} \cup \mathcal{K}$ s.t. $u, v \in K\}$, whose vertex set is the set of the endpoints of the edges of R; note that R is the set of edges which are not covered by $\mathcal{F} \cup \mathcal{K}$;
  if $|V(Q)| > 12k^2$ **then return** No and quit;
  use the algorithm from Proposition 11 to find an edge clique partition $\mathcal{C}$ of Q of minimum size $k' \leq 2k$ if such a partition exists;
  if *the algorithm returns* $\mathcal{C}$ **then**
    if $|\mathcal{C}^*| \leq \alpha(G) + k$ *for* $\mathcal{C}^* = \mathcal{C} \cup \mathcal{F} \cup \mathcal{K}$ **then return** $\mathcal{C}^*$ and quit;
  end
  foreach $uv \in E(G)$ *such that* $u, v \in V(Q)$ *and there is* $F \in \mathcal{F}$ *with* $u, v \in F$ **do**
    if *a call to* EXTEND$(\mathcal{F} \setminus \{F\}, \mathcal{B} \cup \{F\})$ *returns a solution* $\mathcal{C}^*$ **then**
      return $\mathcal{C}^*$ and quit;
    end
  end
  return No
end

▷ Claim 16 (⋆). The algorithm runs in $2^{\mathcal{O}(k^{3/2} \log k)} \cdot n^{\mathcal{O}(1)}$ time.

The proof is complete. ◀

In the conclusion of this section, we remark that the algorithm of Proposition 11 is a bottleneck for the running time of our algorithm for ECP/α. More precisely, given an algorithm solving ECP in time $f(k) \cdot n^{\mathcal{O}(1)}$, the running time of our algorithm is $k^{\mathcal{O}(k)} f(2k) \cdot n^{\mathcal{O}(1)}$. Furthermore, consider the graph G' obtained from a graph G by adding a pendent neighbor to each vertex of G. Since $\mathrm{ecp}(G') = \alpha(G') + \mathrm{ecp}(G)$, a faster algorithm for ECP/α would improve the algorithm of Proposition 11.

4 Clique Cover above Independent Set

In this section, we establish the dichotomy result for ECC/α by showing Theorem 2 and Theorem 3. For this, we need some auxiliary results about relations of EDGE CLIQUE COVER ABOVE INDEPENDENT SET with other problems which will be useful also in Section 5. Therefore, we put them in a separate subsection.

4.1 Reductions between ECC/α, Annotated ECC, and k-Coloring

Following Orlin [42], we consider the generalization of ECC where the aim is to cover a subset of edges.

Annotated Edge Clique Cover (Annotated ECC) parameterized by k

Input:	A graph G, an edge set $B \subseteq E(G)$, an integer k.
Task:	Find k cliques in G, such that each edge in B belongs to at least one clique.

Recall that $\mathrm{ecc}(G)$ denotes the size of the smallest edge clique cover of G. Similarly, in the context of Annotated ECC, for a graph G and a subset of its edges $B \subseteq E(G)$, we denote by $\mathrm{aecc}_B(G)$ the size of the smallest collection of cliques in G that cover all edges in B.

▶ **Lemma 17** (⋆). *ECC/α admits an* FPT*-Turing-reduction to Annotated ECC, where an instance (G, k) is reduced to at most $2k$ instances (G', R', k') such that G' is an induced subgraph of G and $k' \leq 2k$. The reduction works in time $4^k \cdot n^{\mathcal{O}(1)}$.*

The running time of Lemma 17 is dominated by computing the maximum independent set in an induced subgraph, and the reduction can be performed more efficiently on certain graph classes, where instead of Lemma 9 one can compute the independent set directly with a faster algorithm. To accomodate for this, we state the following corollary of Lemma 17.

▶ **Corollary 18** (⋆). *Let $\mathcal{G}$ be a hereditary graph class. ECC/α admits an* FPT*-reduction to Annotated ECC, where an instance (G, k) with $G \in \mathcal{G}$ is reduced to at most $2k$ instances (G', R', k') such that G' is an induced subgraph of G and $k' \leq 2k$. If it can be determined in time $f(k) \cdot n^{\mathcal{O}(1)}$ whether a graph in $\mathcal{G}$ has an independent set of size k, then the reduction admits the same running time bound.*

Next, we show that Annotated ECC can be, in turn, reduced to the k-Coloring problem. We note that the reduction is almost the same as the standard conversion from ECC to Vertex Clique Cover, as introduced by Kou, Stockmeyer and Wong [36].

▶ **Lemma 19** (⋆). *Annotated ECC is polynomial-time reducible to k-Coloring on a graph with $|B|$ vertices.*

As k-Coloring is polynomial-time solvable when $k \leq 2$, we immediately get the following corollary.

▶ **Corollary 20.** *Annotated ECC is polynomial-time solvable for $k \leq 2$.*

We note that a polynomial-time reduction from k-Coloring or Vertex Clique Cover to Annotated ECC follows immediately from a reduction to ECC.

4.2 Dichotomy for ECC/α

In this subsection, we prove that ECC/α is Para-NP-complete when parameterized by k. More precisely, we show that the problem is NP-complete for every $k \geq 2$ on instances where the graph is perfect without isolated vertices. We complement this result by proving that for $k = 0$ or 1, the problem is in P. We start with obtaining the following result for Annotated ECC. The arguments are similar to the arguments of Orlin [42].

▶ **Lemma 21** (⋆). *Annotated ECC is* NP*-complete on co-bipartite graphs for every $k \geq 3$. Furthermore, the hardness holds for the case when B is a perfect matching between the bipartition sets.*

We use Lemma 21 to prove Theorem 2. We restate it here.

▶ **Theorem 2.** *For every $k \geq 2$, ECC/α is* NP*-complete. Furthermore, the hardness holds even on perfect graphs.*

Proof. We use Lemma 21 and reduce from Annotated ECC.

Let (G, B, k) be an instance of Annotated ECC where G is an n-vertex co-bipartite graph for $n \geq 2$. We also assume that B is a perfect matching in G, that is, each vertex of G is incident to an edge of B. We set $R = E(G) \setminus B$ and construct the graph G' as follows.

- Construct a copy of G.
- Construct a vertex v and make it adjacent to each vertex of G.
- For each vertex $x \in V(G)$, construct a vertex u_x and make it adjacent to x.
- For every edge $e = xy \in R$, construct a vertex w_e and make it adjacent to x and y.

Because G is a co-bipartite graph with at least two vertices that has a perfect matching, G has no isolated vertices. Then the construction implies that the same holds for G'. Notice that G' is a perfect graph. To see this, we use the perfect graph theorem [10]. Trivially, a vertex of degree one does not belong to any odd hole or antihole. For any vertex w_e for $e \in R$, w_e does not belong to any odd hole because the neighbors of w_e are adjacent, and w_e does not belong to any antihole with at least 7 vertices because the degree of w_e is two. Since the graph obtained from the co-bipartite graph G by adding the universal vertex v is co-bipartite, this graph is perfect. This immediately implies that G' is perfect.

Consider the set $I = \{v\} \cup \{u_x \mid x \in V(G)\} \cup \{w_e \mid e \in R\}$. By the construction of G', I is an independent set of size $|R| + n + 1$. We claim that I is a maximum independent set. To see this, note that the vertices of $I' = \{u_x \mid x \in V(G)\} \cup \{w_e \mid e \in R\}$ are simplicial. Therefore, there is a maximum independent set of G' containing I'. Since v is the unique vertex nonadjacent to the vertices of I', we obtain that the size of I is maximum. Thus, $\alpha(G') = |R| + n + 1$.

We set $k' = k - 1$ and claim the following.

▷ Claim 22 (⋆). (G, B, k) is a yes-instance of Annotated ECC if and only if (G', k') is a yes-instance of ECC/α.

This concludes the proof of the theorem. ◀

As a corollary of Lemma 19 and Corollary 20, we obtain that when $k = 0$ or $k = 1$, ECC/α can be solved in polynomial time. Thus, combining Theorem 2 and Theorem 3, we get the computational complexity dichotomy for the problem with respect to k.

▶ **Theorem 3.** *ECC/α admits polynomial-time algorithms for $k \in \{0, 1\}$.*

Proof. Given an instance (G, k) of ECC/α for $k \leq 1$, the algorithm from Lemma 17 in polynomial time reduces the problem to solving at most two instances (G', B, k') of Annotated ECC where $k' \leq 2$. Then we apply the algorithm from Corollary 20 to the obtained instances and solve the problem in polynomial time. ◀

Using the result of Orlin [42] about the NP-completeness of the Edge Biclique Cover problem, it is easy to see that ECC is NP-complete on co-bipartite graphs. This result immediately implies that ECC/α is NP-complete on co-bipartite graphs. We provide the proof for completeness in [21].

▶ **Observation 23** (⋆). *ECC/α is* NP*-complete on co-bipartite graphs.*

Because ECC/α is NP-complete on co-bipartite graphs, we have that for every integer $p \geq 2$, NP-complete on graphs G with $\alpha(G) \leq p$. It is trivial to see that if $\alpha(G) = 1$ then G is a complete graph and, therefore, the edges of G can be covered by a single clique.

Theorem 2 and Observation 23 imply that ECC/α is Para-NP-hard when parameterized by either k or the independence number. Because ECC is FPT when parameterized by the number of cliques in a solution [24], it is straightforward to make the following observation.

▶ **Observation 24.** *ECC/α is* FPT *when parameterized by both k and the independence number of the input graph.*

5 Covering Sparse Graphs

In this section, we design FPT algorithms for ECC/α for several well-studied graph classes, where ECC or Independent Set remain NP-complete. We build on Lemma 17 (and Corollary 18), that allow us to transform parameterized algorithms for Annotated ECC (and Independent Set) on a graph class $\mathcal{G}$, into parameterized algorithms for ECC/α on $\mathcal{G}$. The only requirement for $\mathcal{G}$ is being closed under deletion of simplicial vertices.

In the first two subsections of this section, we give parameterized algorithms for Annotated ECC on K_r-free, d-degenerate and H-minor-free graphs. In the third subsection, we combine these algorithms into corresponding algorithms for ECC/α, proving Theorem 4. Finally, we complement these results with lower bounds based on the Exponential Time Hypothesis.

5.1 Bounded Clique Number

Annotated ECC on K_r-free graphs can alternatively be seen as Annotated ECC parameterized by $k+\omega$, where ω is the maximum clique size in G. Naturally, we cannot have more than $\binom{\omega}{2} \cdot k$ edges in B in a yes-instance of Annotated ECC, which hints that Annotated ECC should be FPT with respect to $k + \omega$. There is, however, a crucial obstacle: computing ω is NP-hard and W[1]-hard when parameterized by ω.

Fortunately, independent sets of size more than k in G will guarantee that (G, B, k) is a no-instance. This allows us to use the classical Ramsey's theorem [45] to either conclude that the instance is negative or to find ω. Formally, we use the following algorithmic result that is derived from the Ramsey number upper bound given by Erdős and Szekeres [17].

▶ **Proposition 25** (Adaptation of a proof by Erdős and Szekeres [17])**.** *There is a polynomial-time algorithm that, given n-vertex graph G and two integers p, q such that $n \geq \binom{p+q-2}{p-1}$, finds either a clique of size p or an independent set of size q in G.*

We are ready to prove that Annotated ECC is FPT with respect to $k + \omega$.

▶ **Lemma 26.** *Annotated ECC is fixed-parameter tractable with respect to the combined parameter $k + \omega$. The running time of the corresponding algorithm is $2^{\binom{\omega}{2} \cdot k} \cdot n^{\mathcal{O}(1)}$. The value of ω needs not to be given in the input.*

Proof. We present an algorithm that finds a solution to the given instance (G, B, k) of Annotated ECC. We assume that G, B are not empty, as otherwise the instance is trivial. If $k \leq 2$, we use the algorithm of Corollary 20 as a subroutine to solve the instance in polynomial time. We explain how the algorithm deals with the general case $k \geq 3$ in several steps below.

Triangle-free check. The algorithm first checks whether G is triangle-free, that is, $\omega > 2$. This is performed in polynomial time. If G has $\omega = 2$, then no two edges can belong to the same clique in G, so optimal cover of edges in B consists of $|B|$ cliques. Hence, if G turns out to be triangle-free, then the algorithm reports that (G, B, k) is a yes-instance if $|B| \leq k$ and no-instance otherwise.

Clique or independent set. If there is a vertex $v \in V(G)$ without incident edges in B, then there exists an optimal solution where none of the cliques contain v. The algorithm removes all such vertices from G. Now any solution to (G, B, k) should contain each vertex of G in at least one of the cliques. The algorithm then finds the largest r such that $n \geq \binom{k+r-1}{k}$ and $n < \binom{k+r}{k}$. Substituting r and $k+1$ in Proposition 25, the algorithm finds either an independent set of size $k+1$ in G or a clique of size r in G in polynomial time. An independent set of size $k+1$ guarantees that (G, B, k) is a no-instance, as it is impossible to cover each vertex of G by using at most k cliques. With this outcome, our algorithm reports a no-instance and stops. In the complementary case, we have that $\omega \geq r$ and $n < \binom{k+\omega}{k}$.

Computing ω. For an integer $p \geq 3$, $\omega < p$ can be verified in time $\mathcal{O}(p^2 \cdot n^p)$. As the next step, the algorithm computes the exact value of ω by performing these checks iteratively for consecutive integer values starting from $\max\{4, r+1\}$ up to $\omega + 1$. This routine takes $n^{\omega+\mathcal{O}(1)}$ running time. To give an upper bound on the running time in terms of k and ω, we need the following claim.

▷ Claim 27 (⋆). If a, b are positive integers with $a, b \geq 3$, then $\binom{a+b}{b} \leq 2^{\frac{ab}{2}}$.

By substituting $k, \omega \geq 3$ in Claim 27, we obtain

$$n^{\omega-1} < \binom{k+\omega}{\omega}^{\omega-1} < \left(2^{\frac{k\omega}{2}}\right)^{\omega-1} = 2^{\binom{\omega}{2}\cdot k}.$$

Hence, it takes time $2^{\binom{\omega}{2}\cdot k} \cdot n^{\mathcal{O}(1)}$ for the algorithm to compute the exact value of ω.

Solution via set cover. No clique in G can cover more than $\binom{\omega}{2}$ edges in B. Thus, if $|B| > \binom{\omega}{2} \cdot k$, the algorithm reports correctly that (G, B, k) is a no-instance. Otherwise, the algorithm reduces (G, B, k) to an instance $(B, \mathcal{F}, k)$ of Set Cover. In this instance, B is treated as the universe set. Subset family $\mathcal{F}$ consists of subsets of B, and a subset $B' \subset B$ belongs to $\mathcal{F}$ if and only if all edges of B' belong simultaneously to the same clique in G. Clearly, (G, B, k) is a yes-instance if and only if B can be covered by at most k sets from $\mathcal{F}$, i.e. $(B, \mathcal{F}, k)$ is a yes-instance of Set Cover.

The Set Cover instance is constructed in time $2^{|B|} \cdot n^{\mathcal{O}(1)}$ by the algorithm. Using the celebrated subset convolution algorithm for Set Cover due to Björklund, Husfeldt and Koivisto [5] as a subroutine, our algorithm finally decides $(B, \mathcal{F}, k)$ within the same $2^{|B|} \cdot n^{\mathcal{O}(1)}$ running time bound. The overall running time is therefore dominated by $2^{\binom{\omega}{2}\cdot k} \cdot n^{\mathcal{O}(1)}$. ◀

Consequent parts of Section 5 are omitted and can be found in the full version of the paper [21].

References

1 Wali Mohammad Abdullah and Shahadat Hossain. A sparse matrix approach for covering large complex networks by cliques. In *International Conference on Computational Science*, pages 505–517. Springer, 2022. doi:10.1007/978-3-031-08757-8_43.

2 Pankaj K. Agarwal, Noga Alon, Boris Aronov, and Subhash Suri. Can visibility graphs be represented compactly? *Discrete & Computational Geometry*, 12:347–365, 1994. doi:10.1007/BF02574385.

3 Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999. doi:10.1007/978-3-642-58412-1.

4 Michael Behrisch and Anusch Taraz. Efficiently covering complex networks with cliques of similar vertices. *Theoretical Computer Science*, 355(1):37–47, 2006. doi:10.1016/j.tcs.2005.12.005.

5 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, January 2009. doi:10.1137/070683933.

6 Mathieu Blanchette, Ethan Kim, and Adrian Vetta. Clique cover on sparse networks. In *Proceedings of the 14th Meeting on Algorithm Engineering & Experiments (ALENEX)*, pages 93–102. SIAM / Omnipress, 2012. doi:10.1137/1.9781611972924.10.

7 George Boole. *Of Propositions Numerically Definite*. Watts, London, 1952. Originally published in 1868, Chapter IV.

8 Márcia R. Cerioli, Luérbio Faria, Talita O. Ferreira, Carlos A. J. Martinhon, Fábio Protti, and Bruce A. Reed. Partition into cliques for cubic graphs: Planar case, complexity and approximation. *Discret. Appl. Math.*, 156(12):2270–2278, 2008. doi:10.1016/J.DAM.2007.10.015.

9 Maw-Shang Chang and Haiko Müller. On the tree-degree of graphs. In *In Proceedings of Graph-Theoretic Concepts in Computer Science, 27th International Workshop (WG 2001)*, volume 2204 of *Lecture Notes in Computer Science*, pages 44–54. Springer, 2001. doi:10.1007/3-540-45477-2_6.

10 Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Ann. of Math. (2)*, 164(1):51–229, 2006. doi:10.4007/annals.2006.164.51.

11 Alessio Conte, Roberto Grossi, and Andrea Marino. Large-scale clique cover of real-world networks. *Information and Computation*, 270:104464, 2020. doi:10.1016/J.IC.2019.104464.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM J. Comput.*, 45(1):67–83, 2016. doi:10.1137/130947076.

14 Nicolaas Govert de Bruijn and Pál Erdős. On a combinatorial problem. *Nederl. Akad. Wetensch., Proc.*, 51:1277–1279 = Indagationes Math. 10, 421–423, 1948.

15 Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 5th edition, 2017.

16 Pál Erdős, Adolph W. Goodman, and Lajos Pósa. The representation of a graph by set intersections. *Canadian Journal of Mathematics*, 18:106–112, 1966.

17 Pál Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935. URL: https://www.numdam.org/item/CM_1935__2__463_0/.

18 Andreas Emil Feldmann, Davis Issac, and Ashutosh Rai. Fixed-parameter tractability of the weighted edge clique partition problem. In *15th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 180 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 17, 16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

19 Andres Figueroa, James Bornemann, and Tao Jiang. Clustering binary fingerprint vectors with missing values for dna array data analysis. *Journal of Computational Biology*, 11:887–901, 2004. doi:10.1089/CMB.2004.11.887.

20 Rudolf Fleischer and Xiaotian Wu. Edge clique partition of k 4-free and planar graphs. In *International Conference on Computational Geometry, Graphs and Applications*, pages 84–95. Springer, 2010. doi:10.1007/978-3-642-24983-9_9.

21 Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. Edge clique partition and cover beyond independence, 2025. doi:10.48550/arXiv.2506.21216.

22 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization. Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

23 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.

24 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *ACM J. Exp. Algorithmics*, 13, 2008. doi:10.1145/1412228.1412236.

25 Jens Gramm, Jiong Guo, Falk Hüffner, Rolf Niedermeier, Hans-Peter Piepho, and Ramona Schmid. Algorithms for compact letter displays: Comparison and evaluation. *Computational Statistics & Data Analysis*, 52(2):725–736, 2007. doi:10.1016/j.csda.2006.09.035.

26 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. doi:10.1007/978-3-642-97881-4.

27 Jean-Loup Guillaume and Matthieu Latapy. Bipartite structure of all complex networks. *Inf. Process. Lett.*, 90(5):215–221, 2004. doi:10.1016/j.ipl.2004.03.007.

28 Marshall Hall Jr. A problem in partitions. *Bulletin of the American Mathematical Society*, 47:804–807, 1941.

29 Anthony Hevia, Benjamin Kallus, Summer McClintic, Samantha Reisner, Darren Strash, and Johnathan Wilson. Solving Edge Clique Cover Exactly via Synergistic Data Reduction. In *31st Annual European Symposium on Algorithms (ESA)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 61:1–61:19, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.61.

30 Douglas N Hoover. Complexity of graph covering problems for graphs of low degree. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 11:187–208, 1992.

31 Wen-Lian Hsu and Kuo-Hui Tsai. Linear time algorithms on circular-arc graphs. *Inf. Process. Lett.*, 40(3):123–129, 1991. doi:10.1016/0020-0190(91)90165-E.

32 Harry B. Hunt III, Madhav V. Marathe, Venkatesh Radhakrishnan, and Richard Edwin Stearns. The complexity of planar counting problems. *SIAM J. Comput.*, 27(4):1142–1167, 1998. doi:10.1137/S0097539793304601.

33 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity. *J. Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

34 Eduardo Kellerman. Determination of keyword conflict. *IBM Technical Disclosure Bulletin*, 16(2):544–546, 1973.

35 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000. doi:10.1016/S0020-0190(00)00047-8.

36 Lawrence T. Kou, Larry J. Stockmeyer, and C. K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Communications of the ACM*, 21(2):135–139, 1978. doi:10.1145/359340.359346.

37 László Lovász. On covering of graphs. In *Theory of Graphs, Proceedings of the Colloquium Tihany, Hungary, 1966*, pages 231–236. Academic Press, Budapest, 1968.

38 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994. doi:10.1145/185675.306789.

39 Shaohan Ma, Walter D. Wallis, and Julin Wu. Clique covering of chordal graphs. *Utilitas Mathematica*, 36:151–152, 1989.

40 Egbert Mujuni and Frances A Rosamond. Parameterized complexity of the clique partition problem. In *CATS*, pages 75–78. Citeseer, 2008. URL: `http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV77Mujuni.html`.

41 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

42 James B. Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406–424, 1977.

43 Hans-Peter Piepho. An algorithm for a letter-based representation of all-pairwise comparisons. *Journal of Computational and Graphical Statistics*, 13(2):456–466, 2004.

44 Subramanian Rajagopalan, Manish Vachharajani, and Sharad Malik. Handling irregular ILP within conventional VLIW schedulers using artificial resource constraints. In *CASES 2000*, pages 157–164, 2000. `doi:10.1145/354880.354902`.

45 Frank Plumpton Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, s2-30(1):264–286, 1930. `doi:10.1112/plms/s2-30.1.264`.

46 Herbert John Ryser. Intersection properties of finite sets. *Combinatorial Theory*, 14:79–92, 1973. `doi:10.1016/0097-3165(73)90065-4`.

Near-Optimal Vertex Fault-Tolerant Labels for Steiner Connectivity

Koustav Bhanja
Weizmann Institute of Science, Rehovot, Israel

Asaf Petruschka
Weizmann Institute of Science, Rehovot, Israel

Abstract

We present a compact labeling scheme for determining whether a designated set of terminals in a graph remains connected after any f (or less) vertex failures occur. An *f-FT Steiner connectivity labeling scheme* for an n-vertex graph $G = (V, E)$ with terminal set $U \subseteq V$ provides labels to the vertices of G, such that given only the labels of any subset $F \subseteq V$ with $|F| \leq f$, one can determine if U remains connected in $G - F$. The main complexity measure is the maximum label length.

The special case $U = V$ of global connectivity has been recently studied by Jiang, Parter, and Petruschka [33], who provided labels of $n^{1-1/f} \cdot \text{poly}(f, \log n)$ bits. This is near-optimal (up to $\text{poly}(f, \log n)$ factors) by a lower bound of Long, Pettie and Saranurak [37]. Our scheme achieves labels of $|U|^{1-1/f} \cdot \text{poly}(f, \log n)$ for general $U \subseteq V$, which is near-optimal for any given size $|U|$ of the terminal set. To handle terminal sets, our approach differs from [33]. We use a well-structured Steiner tree for U produced by a decomposition theorem of Duan and Pettie [24], and bypass the need for Nagamochi-Ibaraki sparsification [40].

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Data structures design and analysis; Theory of computation → Distributed algorithms

Keywords and phrases Fault Tolerance, Labeling Schemes, Steiner Connectivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.44

Related Version *Full Version*: `https://arxiv.org/abs/2506.23215`

Funding *Koustav Bhanja*: Supported by Merav Parter's European Research Council (ERC) grant under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 949083.
Asaf Petruschka: Supported by an Azrieli Foundation fellowship, and by Merav Parter's European Research Council (ERC) grant under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 949083.

Acknowledgements We are thankful to Merav Parter for encouraging this collaboration and for valuable discussions.

1 Introduction

Labeling schemes are distributed graph data structures with many applications in graph algorithms and distributed computing. The goal is to assign the vertices (and/or edges) of the graph with succinct yet informative labels, so that queries of interest can be answered from only the labels of the query arguments, without any access to the graph or to any other centralized storage. The main complexity measure is the *label length*, which is the maximum number of bits stored in a single label; construction and query time are secondary measures. Diverse types of labeling have been studied, with notable examples being adjacency [14, 15, 34, 4, 7], ancestry and least common ancestors in trees [1, 6], exact or approximate distances [46, 26, 5, 12, 27], and pairwise edge- or vertex-connectivity [35, 30, 31, 44].


© Koustav Bhanja and Asaf Petruschka;
licensed under Creative Commons License CC-BY 4.0



The error-prone nature of many real-life networks has further motivated researchers to consider labeling schemes designed to support events of failures in the graph. Courcelle and Twig [19] were the first to explicitly define and study *fault-tolerant (FT)* (a.k.a *forbidden set*) labeling schemes, where the goal is to answer queries subject to the failure (i.e., deletion) of a set F of vertices or edges in the graph. Here, the set F is part of the query arguments, so the labels assigned to F are available to the query algorithm. Usually, $|F|$ is assumed to be bounded by some integer parameter $f \geq 1$ (this is abbreviated to the f-FT model). A lot of the works on FT labeling studied special graph families, such as planar or bounded doubling-dimension graphs [19, 18, 2, 3, 8, 13]. In this work, we consider a general n-vertex (undirected) graph $G = (V, E)$ and focus on connectivity-related f-FT labeling schemes.

s-t connectivity. The "s-t variant" of this problem has been extensively studied in recent years. Here, a query consists of a triplet (s, t, F), and it is required to determine if s, t are connected in $G - F$. The study of f-FT s-t connectivity labeling was initiated by Dory and Parter [23] with subsequent works in [32, 41, 42, 37], rendering it reasonably well-understood by now: it is known to admit very succinct labels of only $\text{poly}(f, \log n)$ bits, either with edge failures or with vertex failures; we refer to [37] for a detailed presentation of the state-of-the-art bounds.

Global connectivity. Parter, Petruschka and Pettie [42] have proposed the "global connectivity" variant, where the query consists only of F, and it is required to determine if $G-F$ is a connected graph. Unlike with s-t connectivity, the global variant turns out to behave very differently with edge failures ($F \subseteq E$) or with vertex failures ($F \subseteq V$). Essentially all known labeling schemes for the s-t connectivity variant under edge failures [23, 32, 37] also solve the global variant en route, hence the latter also admits succinct $\text{poly}(f, \log n)$-bit labels. However, in case of vertex failures, it was shown in [42] that polynomial dependency in f is impossible, and their lower bound was improved to $\Omega(n^{1-1/f}/f)$ by Long, Pettie and Saranurak [37]. Very recently, Jiang, Parter and Petruschka [33] gave a nearly matching upper bound by providing a construction of f-FT global connectivity labels of $n^{1-1/f} \cdot \text{poly}(f, \log n)$ bits, essentially settling the problem.

Steiner connectivity. In this work, we study the natural generalization of the problem to *Steiner connectivity*. In this variant, the graph $G = (V, E)$ comes with a (fixed) subset of designated vertices $U \subseteq V$ called *terminals*. Given a query $F \subseteq V \cup E$ of up to f faulty vertices/edges, it is required to report if F is a *U-Steiner cut* in G, i.e., if there is a pair of disconnected terminals in $G - F$. Note that U is fixed and not a part of the query, so we only get to see the labels of F.[1] We call this variant *f-FT Steiner connectivity labeling scheme*.

We first note that the lower bound for the global connectivity variant of [37] immediately yields a lower bound for the Steiner setting as well:[2]

▶ **Theorem 1** ([37]). *Any f-FT Steiner connectivity labeling schemes for n-vertex graphs with k terminals must have label length of $\Omega(k^{1-1/f}/f)$ bits (in the worst case).*

Our technical contribution is providing a nearly matching upper bound which is optimal up to $\text{poly}(f, \log n)$ factors, hence generalizing the result of [33] (recovered when $U = V$):

[1] If we also get the labels of U, we can just use the $\text{poly}(f, \log n)$-bit labels of the s-t variant, and apply $|U| - 1$ pairwise connectivity queries (between one terminal in U to all the others) in $G - F$.

[2] Construct the lower bound instance of [37] on k vertices set as terminals, and add $n - k$ isolated vertices.

▶ **Theorem 2.** *For every n-vertex graph $G = (V, E)$ with terminal set $U \subseteq V$ and $f \geq 1$, there is an f-FT Steiner connectivity labeling scheme for (G, U) with labels of $|U|^{1-1/f} \cdot \text{poly}(f, \log n)$ bits. The labels are computed deterministically in polynomial time.*

The f-FT Steiner connectivity labeling scheme of Theorem 2 supports any query $F \subseteq V \cup E$ of vertices and edges with $|F| \leq f$. However, the challenge lies entirely in the case of vertex failures, i.e., $F \subseteq V$: proving Theorem 2 for this case implies also the general case. Indeed, we can construct a new instance (G', U) where G' is obtained from G by splitting each edge e of G by a new middle v_e, so the failure of e in G is equivalent to the failure of v_e in G' (in terms of affect on connectivity between original vertices).

As for the case of only edge failures $F \subseteq E$, much like in the global connectivity variant, it behaves very differently and admits labels of $\text{poly}(f, \log n)$ bits, implicit by known constructions of f-FT connectivity labels for pairwise connectivity queries (the standard s-t variant in the literature). Specifically, both the labels of [23] based on linear graph sketches and its derandomized version in [32] are straightforward to augment to answer Steiner connectivity queries,[3] and other constructions in [23, 37] seem plausibly amenable to such augmentations as well. In light of the discussions above, in the rest of the paper we will only consider the case of vertex failures, i.e., queries are of the form $F \subseteq V$ with $|F| \leq f$.

Related work: Steiner variants. We mention that providing "Steiner generalizations" that interpolate between the two extreme scenarios of the terminal set ($|U| = n$ and $|U| = 2$) is a common objective in various algorithmic and graph-theoretic problems, such as computing Steiner mincut [22, 17, 29, 20], data structures for Steiner mincuts [21, 22, 9] and minimum+1 Steiner cuts [11], sensitivity oracles for Steiner mincuts [10, 11], Steiner connectivity augmentation and splitting-off [16] and construction of cactus graphs for compactly storing and characterizing Steiner mincuts [22, 29].

Related work: connectivity oracles. FT connectivity labeling schemes can be seen as distributed versions of *connectivity oracles under failures*, which are centralized data structures designed to answer the same kind of queries. These have been extensively studied especially in the s-t variant, with both edge faults [43, 24, 28] and vertex faults [24, 47, 38, 45, 36, 39]. Much like with labeling schemes, in the global variant of connectivity oracles, vertex failures require much larger complexity than edge failures, as was first observed by Long and Saranurak [38]. Very recently, Jiang, Parter and Petruschka [33] provided almost-optimal oracles for global connectivity under vertex failures, and their approach seems plausible to extend also to the Steiner variant.

1.1 Our Approach

While our scheme of Theorem 2 and its formal analysis are not complicated, the intuitions behind them might seem illusive. In this section we aim to clarify these. We first introduce a warm-up suboptimal labeling scheme for the Steiner case, then survey the solution of [33] for the global variant, and finally explain how we "glue together" ideas from both to get our solution. For notational convenience, we use here $\tilde{O}(\cdot)$ to hide $\text{poly}(f, \log n)$ factors.

[3] These are based on fixing some spanning tree T for G, and reconnecting the connected components of $T - F$ to obtain those of $G - F$. Augment the label of each tree edge with the number of terminals in the subtree below it. Then we can invoke a similar strategy to the sketch computation in [23, 32] to determine the number of terminals in each component of $T - F$, and hence also of $G - F$.

A basic building block are the previously-introduced f-FT labels for the s-t connectivity variant, with length $\tilde{O}(1)$ bits [42, 37]. Their construction is highly complicated, but we use them as a black-box. Let $\ell(\cdot)$ denote these labels, and $L(\cdot)$ the labels that we construct; each $L(\cdot)$-label will store many $\ell(\cdot)$-labels.

Warm-up: using degree classification. To gain intuition, we first sketch a *suboptimal* construction based on a strategy introduced in [41]. Assume first $U = V$. Choose an arbitrary spanning tree T in G. Let r be some degree threshold. Let B be the set of vertices with T-degree $\geq r$ (high-deg), and $V - B$ be the rest (low-deg). For a low-deg $x \in V - B$, let $L(x)$ store $\{\ell(x)\} \cup \{\ell(y) \mid y \text{ is } T\text{-neighbor of } x\}$. This handles the following queries:

- **(all-low-deg)** If $F \subseteq V - B$, use the $\ell(\cdot)$-labels to check pairwise connectivity between all T-neighbors of F; one observes that F is a cut in G iff we find some disconnected pair.

To handle the remaining queries where $F \cap B \neq \emptyset$, we use recursion. For each $x \in V$ let $L(x)$ store, for every $y \in B$, x's label *to handle* $f - 1$ *faults in* $G - \{y\}$ (recursively). Then,

- **(one-high-deg)** If there is some $y \in F \cap B$, we answer the query $F - \{y\}$ in $G - y$.

Let b_f denote the resulting label length. As $|B| = O(n/r)$, we get $b_f = \tilde{O}(r + (n/r) \cdot b_{f-1})$. Optimizing r and solving the recurrence (with base case $b_1 = 1$) yields $b_f = \tilde{O}(n^{1-1/2^{f-1}})$.

Generalizing to the Steiner variant with arbitrary terminals $U \subseteq V$ is rather smooth. Essentially, one takes T as a *minimal Steiner tree* that connects all terminals U. Then high-deg vertices in T are only $|B| \leq O(|U|/r)$, which eventually results in $\tilde{O}(|U|^{1-1/2^{f-1}})$-bit labels. The details are provided in the full version of the paper.

Existing solution: the global case. As previously mentioned, Jiang, Parter and Petruschka [33] gave near-optimal labels for the f-FT global connectivity variant, i.e., the case $U = V$. Roughly speaking, they also use a high/low-deg partition B and $V - B$ (with threshold r), but divide queries F in a dual manner to the warm-up above: **(all-high-deg)** $F \subseteq B$, or **(one-low-deg)** $F \cap (V - B) \neq \emptyset$. The high/low-deg partition is w.r.t. to degrees in G, not in a spanning tree. To enable this, they use *Nagamochi-Ibaraki sparsification* [40], yielding a subgraph H of G with $\tilde{O}(n)$ edges such that $G - F$ and $H - F$ have the same connected components, for every $F \subseteq V$, $|F| \leq f$. So w.l.o.g., G has only $m = \tilde{O}(n)$ edges.

To see the broad strokes, we explain their solution in the special case that G is f-vertex-connected. As [33] observe, in this case, if the query F is indeed a cut, then every $x \in F$ is adjacent to every connected component of $G - F$. Thus, for a low-deg $x \in V - B$, one lets $L(x)$ store $\{\ell(y) \mid (x, y) \in E\}$, and answers a query with $x \in F$ by looking for a pair of neighbors disconnected in $G - F$. Note that this argument crucially hinges on the fact that these are all of x's neighbors in the (sparsified) graph G, and not in a spanning tree. To handle queries $F \subseteq B$, one observes that a high-deg $x \in B$ can participate in at most $O(|B|^{f-1})$ such queries, and simply writes the answers to these inside $L(x)$. As $|B| \leq O(m/r)$, the label length is $\tilde{O}(r + (m/r)^{f-1})$. Setting $r = m^{1-1/f}$ gives $\tilde{O}(m^{1-1/f}) = \tilde{O}(n^{1-1/f})$.

Our new solution: the Steiner case. The main issue that prevents extending the solution of [33] to the Steiner case with arbitrary terminals $U \subseteq V$ lies in the sparsification step. Intuitively, to get $\tilde{O}(|U|^{1-1/f})$ label length, we would like to have $\tilde{O}(|U|)$ edges in the sparsified subgraph H, but then preserving connected components under faults is clearly impossible: $|U|$ can be much smaller than $|V|$, so such H cannot even be connected. To tackle this, one may wish to construct other meaningful sparsifiers for Steiner connectivity under faults. This seems to be a challenging task, as even a plausible definition is non-trivial.

Instead, we take a different route and combine the "best-of-both-worlds": we use a Steiner tree for high/low-deg partition as in the warm-up, but (roughly speaking) divide queries F to *all-high-deg* or *one-low-deg* as in [33]. Thus, even in the global case $U = V$, our approach yields a near-optimal scheme which is different than that of [33].

On a high level, the key is not using just any (minimal) Steiner tree T, but one where high-degree vertices B admit a lot of structure, by using a powerful graph decomposition of Duan and Pettie [24]. While it still holds that $|B| \leq O(|U|/r)$, the additional structure is that after deleting B from G, the remaining forest $T - B$ is still a Steiner forest for $U - B$ (i.e., connected terminals in $G - B$ are also connected in $T - B$). The main technical part of our solution lies in a structural analysis that studies the interplay between the (known-in-advance) $G - B$ and $T - B$, and the terminals in the graph $G - F$ defined by a given query F (this is mostly reflected in the correctness proof, specifically Claim 10). Essentially, this extra structure lets us design labels where it is enough for low-deg vertices to be defined w.r.t. the tree, and store the $\ell(\cdot)$-labels of only their tree neighbors.

2 Preliminaries

Throughout the paper, $G = (V, E)$ is an undirected graph with n vertices, $U \subseteq V$ is a designated terminal set in G, and $f \geq 1$ is an integer parameter.

For $K, W \subseteq V$, we say that K *separates* W if there exist two vertices $w, w' \in W - K$ that are disconnected in $G - K$. A vertex set $F \subseteq V$ is called *a cut* in G if it separates V, and a *Steiner cut* if it separates the terminals U.

To shorten statements of theorems, lemmas, etc., we always use the term "algorithm" to mean a deterministic polynomial-time algorithm.

3 Basic Tools

As mentioned in Section 1.1, the most basic tool, which we use as a black box, is the succinct f-FT labels for s-t connectivity:

▶ **Theorem 3** ([37]). *There is an algorithm that outputs an $O(f^4 \log^{7.5}(n))$-bit label $\ell(v)$ to every $v \in V$ with the following property:*

For every $F \subseteq V$ with $|F| \leq f$ and $s, t \in V$, one can determine if s, t are connected in $G - F$, by only inspecting the $\ell(\cdot)$-labels of $F \cup \{s, t\}$.

As explained in Section 1.1, our labels $L(\cdot)$ of Theorem 2 store many $\ell(\cdot)$ labels, and to answer a query F, we make many s-t connectivity queries in $G - F$ using these $\ell(\cdot)$-labels. Suppose that during the query algorithm, we found in this way a pair s, t which is disconnected in $G - F$. Then clearly F is a cut in G, but we still cannot determine that is a *Steiner cut*, as one of s, t might lie in a connected component of $G - F$ without any terminals. We now present a simple tool that augments Theorem 3 to help us detect this situation.

▶ **Lemma 4.** *There is an algorithm that outputs an $O(f^4 \log^{7.5})$-bit label $\bar{\ell}(v)$ to every $v \in V$ with the following property:*

For every $F \subseteq V$ with $|F| \leq f$ and $x \in V$, one can determine there exists some $u \in U$ such that x, u are connected in $G - F$, by only inspecting the $\bar{\ell}(\cdot)$-labels of $F \cup \{x\}$.

Proof. Construct an auxiliary graph G' from G by adding a new vertex z and adding an edge (z, u) for every terminal $u \in U$. Let $\ell'(\cdot)$ be the labels computed by the algorithm of Theorem 3 on G', and define $\bar{\ell}(v) := (\ell'(v), \ell'(z))$. Now, let $F \subseteq V$ with $|F| \leq f$ and

$x \in V$. Note that there exists some $u \in U$ which is connected to x in $G - F$ iff x, z are connected in $G' - F$. The $\bar{\ell}(\cdot)$-labels of $F \cup \{x\}$ contain all $\ell'(\cdot)$-labels of $F \cup \{x, z\}$, so we can use them to determine if the latter condition holds. ◀

4 The Labeling Scheme: Proof of Theorem 2

In this section, we prove our main Theorem 2, giving f-FT Steiner connectivity labels of $|U|^{1-1/f} \cdot \text{poly}(f, \log n)$ bits. Recall that, as discussed in the introduction, we only need to consider up to f vertex failures, i.e., queries $F \subseteq V$ with $|F| \leq f$.

Subset labels. We first introduce a somewhat technical lemma. Roughly speaking, in terms of the high-level intuition of Section 1.1, it serves to treat the *all-high-degree* case. The idea is to store explicit information for some relatively small collection of vertex-subsets (of "high-deg vertices", see Section 1.1), such we have the information on $K \subseteq F$, we can determine if K itself already separates $U - F$, which implies that F is a Steiner cut.

▶ **Lemma 5.** *There is an algorithm that given any $K \subseteq V$, outputs a* subset label $\hat{\ell}(K)$ *of $O(f \log n)$ bits with the following property:*

For every $F \subseteq V$ with $|F| \leq f$, one can determine if K separates $U - F$, by only inspecting $\hat{\ell}(K)$ and the names/identifiers of the vertices in F.

Proof. Let $C_1, \ldots, C_p$ be the connected components of $G - K$ that contain some terminal, and denote $U_i = C_i \cap U$, where w.l.o.g. $|U_1| \leq \cdots \leq |U_p|$. Our goal is that given $\hat{\ell}(K)$ and any $F \subseteq V$, $|F| \leq f$, we can decide if at least two sets in $\{U_i - F \mid 1 \leq i \leq p\}$ are non-empty.

Suppose there exists some q such that $\sum_{i=1}^{q} |U_i| > f$ and $\sum_{i=q+1}^{p} |U_i| > f$. Then for every $F \subseteq V$ with $|F| \leq f$ there is one non-empty set in $\{U_i - F \mid i \leq q\}$ and another in $\{U_i - F \mid i \geq q+1\}$. Thus, we just indicate this situation in $\hat{\ell}(K)$ and always answer YES.

Assume now that no such q exists.

- If $|U_p| > f$: Then $\sum_{i=1}^{p-1} |U_i| < f$, and we let $\hat{\ell}(K)$ store $U_1, \ldots, U_{p-1}$ (and indicate that $|U_p| > f$). This suffices as for every $F \subseteq V$ with $|F| \leq f$ we know that $U_p - F \neq \emptyset$, so we only need to decide if $\{U_i - F \mid 1 \leq i \leq p-1\}$ has a non-empty set.
- If $|U_p| \leq f$: We assert that the sum $\sum_{i=1}^{p} |U_i|$ is at most $3f$. Indeed, if it is more than f, let q be maximal such that $\sum_{i=q+1}^{p} |U_i| > f$. Then $\sum_{i=1}^{q} |U_i| \leq f$, $\sum_{i=q+2}^{p} |U_i| \leq f$ (by maximality), and $|U_{q+1}| \leq |U_p| \leq f$. Thus, we just let $\hat{\ell}(K)$ store all of $U_1, \ldots, U_p$. ◀

The Duan-Pettie Decomposition. Our label construction crucially relies on a graph decomposition algorithm of Duan-Pettie [24] (as explained in Section 1.1). This algorithm is a recursive version of the Fürer-Raghavachari [25] algorithm for finding a Steiner tree with smallest possible maximum degree, up to $+1$ error. The Duan-Pettie decomposition utilizes the structural properties of the Fürer-Raghavachari tree (rather than its strong approximation guarantee). In the following, a *Steiner forest* means a forest in the graph where each pair of connected terminals (in the graph) is also connected in the forest.

▶ **Theorem 6** ([24]). *Let $r \geq 3$ be a* degree threshold. *There is an algorithm* $\mathsf{Decomp}(G, U, r)$ *that outputs a pair (T, B) such that the following hold:*

1. *T is a Steiner forest for U in G, and $T - B$ is a Steiner forest for $U - B$ in $G - B$.*
2. *The maximum degree in the forest $T - B$ is at most r.*
3. *$|B| < |U|/(r-2)$ and $|B \cap U| < |U|/(r-1)$.*

The running time of $\mathsf{Decomp}(G, U, r)$ *is $O(|U| m \log |U|)$.*

In the following we assume that $f \geq 2$, as Theorem 2 is trivial when $f = 1$: the label of vertex x just stores a single bit indicating whether $\{x\}$ is a Steiner cut.

Labeling. Our first step is to invoke $\mathsf{Decomp}(G, U, r := |U|^{1-1/f} + 2)$ of Theorem 6; let (T, B) be the output. For every $x \in V$, we choose an arbitrary terminal $u_x \in U$ such that x, u_x are connected in $G - B$ (where $u_x := \perp$, i.e., a null value, if there is no such terminal). Next, we construct the $\ell(\cdot)$-labels and $\bar{\ell}(\cdot)$-labels of Theorem 3 and Lemma 4; for every $v \in V$, define $\ell^*(v) := (v, \ell(v), \bar{\ell}(v))$. Finally, the $L(\cdot)$ labels are constructed as follows:

Algorithm 1 Creating the label $L(x)$ of a $x \in V$.

1: **store** $\ell^*(x)$
2: **store** $\ell^*(u_x)$ (or $\perp$ if $u_x = \perp$)
3: **store** $\{\ell^*(b) \mid b \in B\}$
4: **store** $\hat{\ell}(K = \emptyset)$ of Lemma 5 (i.e., applying this lemma with $K = \emptyset$)
5: **if** $x \notin B$ **then**
6: **store** $\{\ell^*(y) \mid (x, y)$ is an edge of $T - B\}$
7: **else** ▷ $x \in B$
8: **for** every $K \subseteq B$ s.t. $x \in K$ and $|K| \leq f$ **do**
9: **store** K along with $\hat{\ell}(K)$ of Lemma 5

The label length analysis is rather straightforward, given in detail in the following claim.

▷ Claim 7. Each label $L(x)$ generated by Algorithm 1 has $|U|^{1-1/f} \cdot \mathrm{poly}(f, \log n)$ bits, and takes polynomial time to compute.

Proof. By Theorem 3, Lemma 4 and Lemma 5 each $\ell^*(\cdot)$-label or $\hat{\ell}(\cdot)$-label consists of $\mathrm{poly}(f, \log n)$ bits, so we show that only $O(|U|^{1-1/f})$ such labels are stored in $L(x)$. The first four lines only store $O(|B|)$ such labels, and $|B| \leq |U|/(r-2) = |U|^{1/f}$ by the properties of $\mathsf{Decomp}(G, U, r)$ in Theorem 6). As we assume $f \geq 2$, $|U|^{1/f} \leq |U|^{1-1/f}$, so this is within budget. If $x \notin B$, then, line 6 stores $\ell^*(y)$ for each neighbor y of x in $T - B$, which are at most $r = O(|U|^{1-1/f})$ again by Theorem 6. Next suppose $x \in B$. The number of sets $K \subseteq B$ s.t. $x \in K$ and $|K| \leq f$ can be bounded by

$$\sum_{k=0}^{f-1} \binom{|B|-1}{k} \leq 1 + \sum_{k=1}^{f-1} \left(\frac{|B| \cdot e}{k}\right)^k \leq 1 + |B|^{f-1} \cdot \sum_{k=1}^{f-1} \left(\frac{e}{k}\right)^k \leq O(|U|^{1-1/f})$$

(we used $\binom{n}{k} \leq (n/k)^k$ for $1 \leq k \leq n$, $\sum_{k=1}^{\infty} (e/k)^k$ is convergent, and $|B| \leq |U|^{1/f}$). Thus, lines 8 and 9 store $O(|U|^{1-1/f})$ $\hat{\ell}(\cdot)$-labels.

Note that constructing the $\ell^*(\cdot)$ labels for all vertices takes polynomial time by Theorem 3 and Lemma 4. Also, we only need to construct $\hat{\ell}(K)$ labels for sets $K \subseteq B$ such that $x \in B$ and $|K| \leq f$ (or such that $K = \emptyset$), and there are only $O(|U|^{1-1/f})$ such sets K by the arguments above; constructing each $\hat{\ell}(K)$ takes polynomial time by Lemma 5. ◁

▶ Remark 8. The subset labels of Lemma 5 are trivial to construct with $O(f^2 \log n)$ bits: with notations from the proof, choose arbitrary $f+1$ sets among $U_1, \ldots, U_p$ (or all if $p \leq f$), take $f+1$ terminals from each chosen U_i (or all if $|U_i| \leq f$). We've included the more efficient version to demonstrate that the $\mathrm{poly}(f, \log n)$ factors in Claim 7 come from the f-FT labels of the s-t variant (Theorem 3). These have an $\Omega(f + \log n)$-bit lower bound bits [42], so even if the latter is matched, the contribution of the subset labels is lightweight.

Query algorithm. The pseudo-code for the query algorithm is given in the following Algorithm 2, designed to return *YES* if the query F is a Steiner cut, and *NO* otherwise.

Algorithm 2 Answering a query $F \subseteq V$, $|F| \leq f$ from the labels $\{L(x) \mid x \in F\}$.

1: let $K := F \cap B$
2: **if** K separates $U - F$ in G **then return** *YES*
3: let $S := \{v \in V - F \mid \ell^*(v)$ is stored in $L(x)$ of some $x \in F\}$
4: let $W := \{w \in S \mid \exists u \in U$ s.t. u, w are connected in $G - F\}$
5: **if** $W \neq \emptyset$ **then**
6: choose an arbitrary $w^* \in W$
7: **for** every $w \in W - \{w^*\}$ **do**
8: **if** w^*, w are disconnected in $G - F$ **then return** *YES*
9: **return** *NO*

We now provide the implementation details:

- Line 2 is executed using $\hat{\ell}(K)$ of Lemma 5 (and the names of the vertices in F, given in their labels). If $K = F \cap B$ is non-empty, we can find $\hat{\ell}(K)$ in the label $L(x)$ of some $x \in F \cap B$. Otherwise, $\hat{\ell}(K)$ is stored in $L(x)$ for every $x \in F$.
- Line 4 is executed by using the $\bar{\ell}(\cdot)$-labels of the vertices in $S \cup F$ (which appear in their $\ell^*(\cdot)$-labels). For every $w \in S$, we use $\bar{\ell}(w)$ and $\{\bar{\ell}(x) \mid x \in F\}$ to decide if w is connected to some $u \in U$ in $G - F$, which is possible by Lemma 4.
- Line 8 is executed using the $\ell(\cdot)$-labels of w^*, w and F (which appear in their $\ell^*(\cdot)$-labels), which enable to determine if w^*, w are connected in $G - F$ by Theorem 3.

Correctness. We need to show that *YES* is returned if and only if F is a Steiner cut, namely F separates U in G. The soundness direction ('only if') is trivial:

▷ Claim 9. If the query algorithm (Algorithm 2) returns *YES* then F is a Steiner cut in G.

Proof. If *YES* was returned in line 2, then F has a subset K which separates two terminals in $U - F$, so these are also separated by F. Thus, F is a Steiner cut.

Otherwise, *YES* was returned in line 8, so there are $w^*, w \in W$ disconnected in $G - F$. By definition of W (line 4 of Algorithm 1), there exists terminals $u^*, u \in U$ such that u^*, w^* and u, w are connected pairs in $G - F$. Thus, u^*, u are two terminals disconnected in $G - F$, so F is a Steiner cut. ◁

Next, we address the completeness ('if') direction:

▷ Claim 10. If F is a Steiner cut in G, then the query algorithm (Algorithm 2) returns *YES*.

Proof. Let us first consider the trivial case when $K = F \cap B$ separates $U - F$ in G. Then Algorithm 2 returns *YES* in line 2 as needed. Henceforth, we assume that F satisfies the following property:

(P1) $F \cap B$ does not separate $U - F$ in G.

Note that given Property (P1), the only possible place where Algorithm 2 can return *YES* is in line 8, so we aim to show that this is indeed the case. We will crucially use Property (P1) later on in the proof.

Let $u, u' \in U$ be two terminals disconnected in $G - F$, which exists as F is a Steiner cut. We will show there exists $w, w' \in W$ such that (i) u, w are connected in $G - F$ and (ii) u', w' are connected in $G - F$. Let us argue why this suffices to complete the proof. First, this

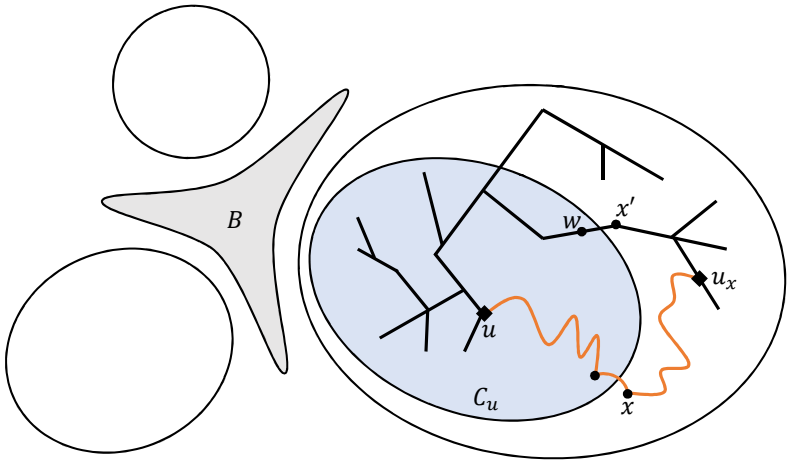


Figure 1 Illustrating the proof of Claim 10. Given (P1) and (P2), the connected component C_u of u in $G - F$ is fully contained in some connected component of $G - B$, and C_u has some neighbor $x \in F$ within this bigger component. Thus, both u and the chosen terminal u_x for x are connected to x in this component. The forest $T - B$ contains a tree that connects *all terminals* in this component, so it must connect u and u_x. If $u_x \notin C_u$ (as shown in the figure), then the tree must contain a vertex $w \in C_u$ which is a neighbor of some $x' \in F$ in $T - B$, and so $w \in S$. Otherwise, $u_x \in C_u \cap S$. In either case, we find some vertex in $C_u \cap S$ as required.

clearly implies that $W \neq \emptyset$. Second, one of w, w' must be disconnected from w^* in $G - F$, as otherwise w, w' would be connected in $G - F$, so by (i) and (ii) we get that u, u' are also connected in $G - F$, but this is a contradiction. Therefore, line 8 of Algorithm 2, when executed with either w or w' must return *YES* as required.

We thus focus on showing the existence of $w \in W$ which is connected to u in $G - F$ (and the proof for w' and u' is symmetric). Let C_u be the connected component of u in $G - F$. Recall the definitions of S and its subset W from lines 3 and 4 of Algorithm 2. Note that every vertex $w \in C_u \cap S$ is automatically in W, since it belongs to S and is connected to the terminal u in $G - F$. So, our goal is simply to show that $C_u \cap S \neq \emptyset$, or in other words, that the $\ell^*(\cdot)$-label of some vertex in C_u is stored in some $L(x)$ of $x \in F$.

Again, there is a trivial case: if $C_u \cap B \neq \emptyset$ then we are done, because $\{\ell^*(b) \mid b \in B\}$ is stored in every $L(x)$ of $x \in F$ (line 3 of Algorithm 1), so $B \subseteq S$. Thus, we assume that:

(P2) The connected component C_u of u in $G - F$ does not intersect B.

In fact, the inclusion of the $\ell^*(\cdot)$-labels of B in every label $L(x)$ is precisely aimed at enforcing Property (P2).

Our next step is to "capitalize on" properties (P1), (P2), and the important property that $T - B$ is a Steiner forest for $U - B$ in $G - B$ (guaranteed by Theorem 6). Consider the set $N_G(C_u)$ of all vertices outside C_u that are adjacent to C_u in G. Note that, as C_u is a connected component in $G - F$, we have $N_G(C_u) \subseteq F$. Observe that $N_G(C_u)$ separates $U - F$ in G, as $u, u' \in U - F$ with $u \in C_u$ and $u' \notin C_u$. Thus, by Property (P1), it must be that $N_G(C_u) \not\subseteq F \cap B$. Hence, there exists some $x \in N_G(C_u) \cap (F - B)$. Now, using Property (P2), we see that all vertices in $C_u \cup \{x\}$ are connected in $G - B$, so u is connected to x in $G - B$. Recall that the terminal $u_x \in U$ is also (by definition) connected to x in $G - B$. (Note that u_x exists since x is connected to the terminal u in $G - B$.) Hence, u, u_x are connected in $G - B$, and thus also in $T - B$. This is the crux for finishing the proof in the following argument. See Figure 1 for an illustration.

Observe that $u_x \in S$, since $\ell^*(u_x)$ is stored in $L(x)$ (line 2 of Algorithm 1). So, if u, u_x are connected in $G - F$, then $u_x \in C_u \cap S$ and we are done. Otherwise, the path in $T - B$ between u and u_x must go through F. Let $x' \in F$ be the vertex closest to u on this path, and let w be its neighbor in the direction of u on the path. (It might be that $x' = x$.) Then the subpath between u and w avoids F, so $w \in C_u$. Also, $x' \notin B$ and (x', w) is an edge of $T - B$, hence $\ell^*(w)$ is stored in $L(x')$ (line 6 of Algorithm 1), so $w \in S$. Namely, we have shown that $C_u \cap S \neq \emptyset$, and the proof is concluded. ◁

Theorem 2 follows immediately from the combination of Claim 7, Claim 9 and Claim 10.

References

1 Serge Abiteboul, Stephen Alstrup, Haim Kaplan, Tova Milo, and Theis Rauhe. Compact labeling scheme for ancestor queries. *SIAM J. Comput.*, 35(6):1295–1309, 2006. doi:10.1137/S0097539703437211.

2 Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1199–1218. ACM, 2012. doi:10.1145/2213977.2214084.

3 Ittai Abraham, Shiri Chechik, Cyril Gavoille, and David Peleg. Forbidden-set distance labels for graphs of bounded doubling dimension. *ACM Trans. Algorithms*, 12(2):22:1–22:17, 2016. doi:10.1145/2818694.

4 Stephen Alstrup, Søren Dahlgaard, and Mathias Bæk Tejs Knudsen. Optimal induced universal graphs and adjacency labeling for trees. *J. ACM*, 64(4):27:1–27:22, 2017. doi:10.1145/3088513.

5 Stephen Alstrup, Cyril Gavoille, Esben Bistrup Halvorsen, and Holger Petersen. Simpler, faster and shorter labels for distances in graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 338–350. SIAM, 2016. doi:10.1137/1.9781611974331.CH25.

6 Stephen Alstrup, Esben Bistrup Halvorsen, and Kasper Green Larsen. Near-optimal labeling schemes for nearest common ancestors. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 972–982. SIAM, 2014. doi:10.1137/1.9781611973402.72.

7 Stephen Alstrup, Haim Kaplan, Mikkel Thorup, and Uri Zwick. Adjacency labeling schemes and induced-universal graphs. *SIAM J. Discret. Math.*, 33(1):116–137, 2019. doi:10.1137/16M1105967.

8 Aviv Bar-Natan, Panagiotis Charalampopoulos, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Fault-tolerant distance labeling for planar graphs. *Theor. Comput. Sci.*, 918:48–59, 2022. doi:10.1016/J.TCS.2022.03.020.

9 Surender Baswana and Abhyuday Pandey. The connectivity carcass of a vertex subset in a graph: both odd and even case. In *2025 Symposium on Simplicity in Algorithms, SOSA 2025, New Orleans, LA, USA, January 13-15, 2025*, pages 385–422. SIAM, 2025. doi:10.1137/1.9781611978315.30.

10 Koustav Bhanja. Optimal sensitivity oracle for steiner mincut. In *35th International Symposium on Algorithms and Computation, ISAAC 2024, December 8-11, 2024, Sydney, Australia*, volume 322 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ISAAC.2024.10.

11 Koustav Bhanja. Minimum+1 steiner cuts and dual edge sensitivity oracle: Bridging gap between global and (s,t)-cuts. In *The 52nd EATCS International Colloquium on Automata, Languages, and Programming (ICALP)*, 2025.

12 Marthe Bonamy, Cyril Gavoille, and Michal Pilipczuk. Shorter labeling schemes for planar graphs. *SIAM J. Discret. Math.*, 36(3):2082–2099, 2022. doi:10.1137/20M1330464.

13 Itai Boneh, Shiri Chechik, Shay Golan, Shay Mozes, and Oren Weimann. Õptimal fault-tolerant labeling for reachability and approximate distances in directed planar graphs.

14 Melvin A. Breuer. Coding the vertexes of a graph. *IEEE Trans. Inf. Theory*, 12(2):148–153, 1966. doi:10.1109/TIT.1966.1053860.

15 Melvin A Breuer and Jon Folkman. An unexpected result in coding the vertices of a graph. *Journal of Mathematical Analysis and Applications*, 20(3):583–600, 1967. doi:10.1016/0022-247X(67)90082-0.

16 Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. Steiner connectivity augmentation and splitting-off in poly-logarithmic maximum flows. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2449–2488. SIAM, 2023. doi:10.1137/1.9781611977554.CH95.

17 Richard Cole and Ramesh Hariharan. A fast algorithm for computing steiner edge connectivity. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 167–176, 2003. doi:10.1145/780542.780568.

18 Bruno Courcelle, Cyril Gavoille, Mamadou Moustapha Kanté, and Andrew Twigg. Connectivity check in 3-connected planar graphs with obstacles. *Electron. Notes Discret. Math.*, 31:151–155, 2008. doi:10.1016/J.ENDM.2008.06.030.

19 Bruno Courcelle and Andrew Twigg. Compact forbidden-set routing. In *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2007. doi:10.1007/978-3-540-70918-3_4.

20 Matthew Ding and Jason Li. Deterministic minimum steiner cut in maximum flow time. In *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 46:1–46:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.46.

21 Yefim Dinitz and Alek Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 716–725, 1994. doi:10.1145/195058.195442.

22 Yefim Dinitz and Alek Vainshtein. The general structure of edge-connectivity of a vertex subset in a graph and its incremental maintenance. odd case. *SIAM J. Comput.*, 30(3):753–808, 2000. doi:10.1137/S0097539797330045.

23 Michal Dory and Merav Parter. Fault-tolerant labeling and compact routing schemes. In *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 445–455. ACM, 2021. doi:10.1145/3465084.3467929.

24 Ran Duan and Seth Pettie. Connectivity oracles for graphs subject to vertex failures. *SIAM J. Comput.*, 49(6):1363–1396, 2020. doi:10.1137/17M1146610.

25 M. Furer and B. Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *Journal of Algorithms*, 17(3):409–423, 1994. doi:10.1006/jagm.1994.1042.

26 Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *J. Algorithms*, 53(1):85–112, 2004. doi:10.1016/J.JALGOR.2004.05.002.

27 Pawel Gawrychowski and Przemyslaw Uznanski. Better distance labeling for unweighted planar graphs. *Algorithmica*, 85(6):1805–1823, 2023. doi:10.1007/S00453-023-01133-Z.

28 David Gibb, Bruce M. Kapron, Valerie King, and Nolan Thorn. Dynamic graph connectivity with improved worst case update time and sublinear space. *CoRR*, abs/1509.06464, 2015. arXiv:1509.06464.

29 Zhongtian He, Shang-En Huang, and Thatchaphol Saranurak. Cactus representations in polylogarithmic max-flow via maximal isolating mincuts. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1465–1502. SIAM, 2024. doi:10.1137/1.9781611977912.60.

30 Tai-Hsin Hsu and Hsueh-I Lu. An optimal labeling for node connectivity. In *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, volume 5878 of *Lecture Notes in Computer Science*, pages 303–310. Springer, 2009. doi:10.1007/978-3-642-10631-6_32.

31 Rani Izsak and Zeev Nutov. A note on labeling schemes for graph connectivity. *Inf. Process. Lett.*, 112(1-2):39–43, 2012. doi:10.1016/J.IPL.2011.10.001.

32 Taisuke Izumi, Yuval Emek, Tadashi Wadayama, and Toshimitsu Masuzawa. Deterministic fault-tolerant connectivity labeling scheme. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 190–199. ACM, 2023. doi:10.1145/3583668.3594584.

33 Yonggang Jiang, Merav Parter, and Asaf Petruschka. New oracles and labeling schemes for vertex cut queries. *CoRR*, abs/2501.13596, 2025. doi:10.48550/arXiv.2501.13596.

34 Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. *SIAM J. Discret. Math.*, 5(4):596–603, 1992. doi:10.1137/0405049.

35 Michal Katz, Nir A. Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004. doi:10.1137/S0097539703433912.

36 Evangelos Kosinas. Connectivity Queries Under Vertex Failures: Not Optimal, but Practical. In *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 75:1–75:13, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.75.

37 Yaowei Long, Seth Pettie, and Thatchaphol Saranurak. Connectivity labeling schemes for edge and vertex faults via expander hierarchies. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–47, 2025. doi:10.1137/1.9781611978322.1.

38 Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1002–1010. IEEE, 2022. doi:10.1109/FOCS54457.2022.00098.

39 Yaowei Long and Yunfan Wang. Better decremental and fully dynamic sensitivity oracles for subgraph connectivity. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 109:1–109:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.109.

40 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. doi:10.1007/BF01758778.

41 Merav Parter and Asaf Petruschka. Õptimal dual vertex failure connectivity labels. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPIcs*, pages 32:1–32:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.DISC.2022.32.

42 Merav Parter, Asaf Petruschka, and Seth Pettie. Connectivity labeling and routing with multiple vertex failures. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 823–834. ACM, 2024. doi:10.1145/3618260.3649729.

43 Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 263–271. IEEE Computer Society, 2007. doi:10.1109/FOCS.2007.59.

44 Seth Pettie, Thatchaphol Saranurak, and Longhui Yin. Optimal vertex connectivity oracles. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 151–161. ACM, 2022. doi:10.1145/3519935.3519945.

45 Michal Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Torunczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 102:1–102:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.102.

46 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

47 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 424–435. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00034.

Bounded Weighted Edit Distance

Dynamic Algorithms and Matching Lower Bounds

Itai Boneh
Reichman University, Herzliya, Israel
University of Haifa, Israel

Egor Gorbachev
Saarland University and Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Tomasz Kociumaka
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

The edit distance $\mathsf{ed}(X, Y)$ of two strings $X, Y \in \Sigma^*$ is the minimum number of character edits (insertions, deletions, and substitutions) needed to transform X into Y. Its weighted counterpart $\mathsf{ed}^w(X, Y)$ minimizes the total cost of edits, where the costs of individual edits, depending on the edit type and the characters involved, are specified using a function w, normalized so that each edit costs at least one. The textbook dynamic-programming procedure, given strings $X, Y \in \Sigma^{\le n}$ and oracle access to w, computes $\mathsf{ed}^w(X, Y)$ in $\mathcal{O}(n^2)$ time. Nevertheless, one can achieve better running times if the computed distance, denoted k, is small: $\mathcal{O}(n + k^2)$ for unit weights [Landau and Vishkin; JCSS'88] and $\widetilde{\mathcal{O}}(n + \sqrt{nk^3})$[1] for arbitrary weights [Cassis, Kociumaka, Wellnitz; FOCS'23].

In this paper, we study the dynamic version of the weighted edit distance problem, where the goal is to maintain $\mathsf{ed}^w(X, Y)$ for strings $X, Y \in \Sigma^{\le n}$ that change over time, with each update specified as an edit in X or Y. Very recently, Gorbachev and Kociumaka [STOC'25] showed that the *unweighted* distance $\mathsf{ed}(X, Y)$ can be maintained in $\widetilde{\mathcal{O}}(k)$ time per update after $\widetilde{\mathcal{O}}(n + k^2)$-time preprocessing; here, k denotes the *current* value of $\mathsf{ed}(X, Y)$. Their algorithm generalizes to small integer weights, but the underlying approach is incompatible with large weights.

Our main result is a dynamic algorithm that maintains $\mathsf{ed}^w(X, Y)$ in $\widetilde{\mathcal{O}}(k^{3-\gamma})$ time per update after $\widetilde{\mathcal{O}}(nk^\gamma)$-time preprocessing. Here, $\gamma \in [0, 1]$ is a real trade-off parameter and $k \ge 1$ is an integer threshold *fixed* at preprocessing time, with ∞ returned whenever $\mathsf{ed}^w(X, Y) > k$. We complement our algorithm with conditional lower bounds showing fine-grained optimality of our trade-off for $\gamma \in [0.5, 1)$ and justifying our choice to fix k.

We also generalize our solution to a much more robust setting while preserving the fine-grained optimal trade-off. Our full algorithm maintains $X \in \Sigma^{\le n}$ subject not only to character edits but also substring deletions and copy-pastes, each supported in $\widetilde{\mathcal{O}}(k^2)$ time. Instead of dynamically maintaining Y, it answers queries that, given any string Y specified through a sequence of $\mathcal{O}(k)$ arbitrary edits transforming X into Y, in $\widetilde{\mathcal{O}}(k^{3-\gamma})$ time compute $\mathsf{ed}^w(X, Y)$ or report that $\mathsf{ed}^w(X, Y) > k$.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Edit distance, dynamic algorithms, conditional lower bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.45

Related Version *Full Version*: `https://arxiv.org/abs/2507.02548v1` [6]

Funding *Itai Boneh*: supported by Israel Science Foundation grant 810/21.
Egor Gorbachev: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

[1] Henceforth, the $\widetilde{\mathcal{O}}(\cdot)$ notation hides factors poly-logarithmic in n.


© Itai Boneh, Egor Gorbachev, and Tomasz Kociumaka;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Among the most fundamental string processing problems is the task of deciding whether two strings are similar to each other. A classic measure of string (dis)similarity is the *edit distance*, also known as the *Levenshtein distance* [22]. The (unit-cost) edit distance $\mathsf{ed}(X, Y)$ of two strings X and Y is defined as the minimum number of character edits (insertions, deletions, and substitutions) needed to transform X into Y. In typical use cases, edits model naturally occurring modifications, such as typographical errors in natural-language texts or mutations in biological sequences. Some of these changes occur more frequently than others, which motivated introducing *weighted edit distance* already in the 1970s [33]. In this setting, each edit has a cost that may depend on its type and the characters involved (but nothing else), and the goal is to minimize the total cost of edits rather than their quantity. The costs of individual edits can be specified through a weight function $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, where $\overline{\Sigma} = \Sigma \cup \{\varepsilon\}$ denotes the alphabet extended with a symbol ε representing the lack of a character. For $a, b \in \Sigma$, the cost of inserting b is $w(\varepsilon, b)$, the cost of deleting a is $w(a, \varepsilon)$, and the cost of substituting a for b is $w(a, b)$. (Note that the cost of an edit is independent of the position of the edited character in the string.) Consistently with previous works, we assume that w is normalized: $w(a, b) \geq 1$ and $w(a, a) = 0$ hold for every distinct $a, b \in \overline{\Sigma}$. The unweighted edit distance constitutes the special case when $w(a, b) = 1$ holds for $a \neq b$.

The textbook dynamic-programming algorithm [32, 25, 26, 33] takes $\mathcal{O}(n^2)$ time to compute the edit distance of two strings of length at most n, and some of its original formulations incorporate weights [26, 33, 27]. Unfortunately, there is little hope for much faster solutions: for any constant $\delta > 0$, an $\mathcal{O}(n^{2-\delta})$-time algorithm would violate the Orthogonal Vectors Hypothesis [1, 7, 2, 5], and hence the Strong Exponential Time Hypothesis [16, 17]. The lower bound holds already for unit weights and many other fixed weight functions [7].

Bounded Edit Distance. On the positive side, the quadratic running time can be improved when the computed distance is small. For unweighted edit distance, the algorithm by Landau and Vishkin [21], building upon the ideas of [30, 24], computes $k := \mathsf{ed}(X, Y)$ in $\mathcal{O}(n + k^2)$ time for any two strings $X, Y \in \Sigma^{\leq n}$. This running time is fine-grained optimal: for any $\delta > 0$, a hypothetical $\mathcal{O}(n + k^{2-\delta})$-time algorithm, even restricted to instances satisfying $k = \Theta(n^\kappa)$ for some constant $\kappa \in (0.5, 1]$, would violate the Orthogonal Vectors Hypothesis.

As far as the bounded *weighted* edit distance problem is concerned, a simple optimization by Ukkonen [30] improves the quadratic time complexity of the classic dynamic-programming algorithm to $\mathcal{O}(nk)$, where $k := \mathsf{ed}^w(X, Y)$. Recently, Das, Gilbert, Hajiaghayi, Kociumaka, and Saha [12] developed an $\mathcal{O}(n + k^5)$-time solution and, soon afterwards, Cassis, Kociumaka, and Wellnitz [8] presented an $\widetilde{\mathcal{O}}(n + \sqrt{nk^3})$-time algorithm. Due to the necessity to read the entire input, the latter running time is optimal for $k \leq \sqrt[3]{n}$. More surprisingly, there is a tight conditional lower bound for $\sqrt{n} \leq k \leq n$ [8], valid already for edit costs in the real interval $[1, 2]$. For any constants $\kappa \in [0.5, 1]$ and $\delta > 0$, an $\mathcal{O}(\sqrt{nk^{3-\delta}})$-time algorithm restricted to instances satisfying $k = \Theta(n^\kappa)$ would violate the All-Pairs Shortest Paths Hypothesis [31]. The optimal complexity remains unknown for $\sqrt[3]{n} < k < \sqrt{n}$, where the upper bound is $\widetilde{\mathcal{O}}(\sqrt{nk^3})$, yet the lower-bound construction allows for an $\widetilde{\mathcal{O}}(n + k^{2.5})$-time algorithm.

Dynamic Edit Distance. Over the last decades, the edit distance problem has been studied in numerous variants, including the dynamic ones, where the input strings change over time. The most popular dynamic (weighed) edit distance formulation is the following one [9, 15].

▶ **Problem 1.1** (Dynamic Weighted Edit Distance). *Given an integer n and oracle access to a normalized weight function $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, maintain strings $X, Y \in \Sigma^{\leq n}$ subject to updates (character edits in X and Y) and report $\mathsf{ed}^w(X,Y)$ after each update.*[2]

The time complexity of Problem 1.1 is already well understood if it is measured solely in terms of the string length n. In case of unit weights, the algorithm of Charalampopoulos, Kociumaka, and Mozes [9] (building upon the techniques by Tiskin [28]) is characterized by $\widetilde{\mathcal{O}}(n)$ update time and $\mathcal{O}(n^2)$ preprocessing time; any polynomial-factor improvement would violate the lower bound for the static edit distance problem. As far as arbitrary weights are concerned, the approach presented in [9, Section 4] achieves $\widetilde{\mathcal{O}}(n\sqrt{n})$ update time after $\widetilde{\mathcal{O}}(n^2)$-time preprocessing, and there is a matching fine-grained lower bound [8, Section 6] conditioned on the All-Pairs Shortest Paths Hypothesis.

In this work, we aim to understand Problem 1.1 for small distances:

How fast can one dynamically maintain $\mathsf{ed}^w(X,Y)$ when this value is small?

Very recently, Gorbachev and Kociumaka [15] settled the parameterized complexity of Problem 1.1 in the *unweighted* case. Their solution achieves $\widetilde{\mathcal{O}}(n+k^2)$ preprocessing time and $\widetilde{\mathcal{O}}(k)$ update time; improving either complexity by a polynomial factor, even for instances satisfying $k = \Theta(n^\kappa)$ for some $\kappa \in (0,1]$, would violate the Orthogonal Vectors Hypothesis.

Interestingly, the approach of [15] builds upon the static procedure for bounded *weighted* edit distance [8] and, as a result, the dynamic algorithm seamlessly supports *small integer* weights, achieving $\mathcal{O}(W^2k)$ update time for edit costs in $[1..W]$. Unfortunately, [15] does not provide any improvements for arbitrary weights; in that case, the results of [8] yield a solution to Problem 1.1 with $\widetilde{\mathcal{O}}(n)$ preprocessing time and $\widetilde{\mathcal{O}}(\min(k^3, \sqrt{nk^3}))$ update time. This is far from satisfactory: for $k \geq \sqrt[3]{n}$, the algorithm simply recomputes $\mathsf{ed}^w(X,Y)$ from scratch after every update, and for $k < \sqrt[3]{n}$, it does not store (and reuse) anything beyond a dynamic data structure for checking the equality between fragments of X and Y.

Our Results: Lower Bounds

The reason why the approach of [15] is incompatible with large weights, e.g., $W = \Omega(k)$, is that their presence allows a single update to drastically change $\mathsf{ed}^w(X,Y)$. Our first result strengthens the lower bound of [8] and shows that the $\widetilde{\mathcal{O}}(n+\sqrt{nk^3})$-time static algorithm is fine-grained optimal already for instances that can be transformed from a pair of equal strings using four edits. The ideas behind the following theorem are presented in Section 5.

▶ **Theorem 1.2.** *Let $\kappa \in [0.5, 1]$ and $\delta > 0$ be real parameters. Assuming the APSP Hypothesis, there is no algorithm that, given two strings $X, Y \in \Sigma^{\leq n}$ satisfying $\mathsf{ed}(X,Y) \leq 4$, a threshold $k \leq n^\kappa$, and oracle access to a normalized weight function $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, in time $\mathcal{O}(n^{0.5+1.5\kappa-\delta})$ decides whether $\mathsf{ed}^w(X,Y) \leq k$.*

As a simple corollary, we conclude that significantly improving upon the naïve update time of $\widetilde{\mathcal{O}}(\sqrt{nk^3})$ for $\sqrt{n} \leq k \leq n$ requires large preprocessing time.

▶ **Corollary 1.3.** *Suppose that Problem 1.1 admits a solution with preprocessing time $T_P(n, \mathsf{ed}^w(X,Y))$ and update time $T_U(n, \mathsf{ed}^w(X,Y))$ for some non-decreasing functions T_P and T_U. If $T_P(n,0) = \mathcal{O}(n^{0.5+1.5\kappa-\delta})$ and $T_U(n, n^\kappa) = \mathcal{O}(n^{0.5+1.5\kappa-\delta})$ hold for some real parameters $\kappa \in [0.5, 1]$ and $\delta > 0$, then the APSP Hypothesis fails.*

[2] For clarity, the introduction focuses on the version of the problem where the value $\mathsf{ed}^w(X,Y)$ is reported after every update. In Theorem 1.7, we address a more general setting where queries may occur less frequently, allowing us to distinguish between update time and query time.

Proof idea. For an instance (X, Y) of the static problem of Theorem 1.2, we initialize the dynamic algorithm with (X, X) and transform one copy of X into Y using four updates. ◀

As discussed in the full version of the paper [6], instead of initializing the dynamic algorithm with (X, X), we can initialize it with a pair of empty strings and gradually transform this instance to (X, X) using $\mathcal{O}(n)$ updates while keeping the weighted edit distance between 0 and 2 at all times. Hence, as long as the preprocessing of $(\varepsilon, \varepsilon)$ takes $n^{\mathcal{O}(1)}$ time and is not allowed to access the entire weight function (e.g., weights are revealed online when an update introduces a fresh character for the first time) , we can replace $T_P(n, 0) = \mathcal{O}(n^{0.5+1.5\kappa-\delta})$ with $T_U(n, 2) = \mathcal{O}(n^{1.5\kappa-0.5-\delta})$ in the statement of Corollary 1.3.

Overall, we conclude that, in order to support updates with relatively large distances faster than naively (i.e., by computing $\mathsf{ed}^w(X, Y)$ from scratch using a static algorithm), one needs to pay a lot for both preprocessing and updates while $\mathsf{ed}^w(X, Y)$ is still very small. In particular, we should decide in advance how large distances need to be supported efficiently. This justifies a simplified variant of Problem 1.1, where the parameter k is fixed at preprocessing time. Here, instead of $\mathsf{ed}^w(X, Y)$, the algorithm reports the following quantity:

$$\mathsf{ed}^w_{\leq k}(X, Y) = \begin{cases} \mathsf{ed}^w(X, Y) & \text{if } \mathsf{ed}^w(X, Y) \leq k, \\ \infty & \text{otherwise.} \end{cases}$$

▶ **Problem 1.4** (Dynamic Weighted Edit Distance with Fixed Threshold)**.** *Given integers $1 \leq k \leq n$ and oracle access to a normalized weight function $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, maintain strings $X, Y \in \Sigma^{\leq n}$ subject to updates (character edits in X and Y) and report $\mathsf{ed}^w_{\leq k}(X, Y)$ upon each update.*

Corollary 1.3 can be rephrased in terms of Problem 1.4, but its major limitation is that it only applies to $k \geq \sqrt{n}$. To overcome this issue, we compose multiple hard instances from Theorem 1.2. This results in the following theorem, discussed in Section 5.

▶ **Theorem 1.5.** *Suppose that Problem 1.4 admits a solution with preprocessing time $T_P(n, k)$ and update time $T_U(n, k)$ for some non-decreasing functions T_P and T_U. If $T_P(n, n^\kappa) = \widetilde{\mathcal{O}}(n^{1+\kappa\cdot\gamma})$ and $T_U(n, n^\kappa) = \mathcal{O}(n^{\kappa\cdot(3-\gamma)-\delta})$ hold for some parameters $\gamma \in [0.5, 1)$, $\kappa \in (0, 1/(3-2\gamma)]$, and $\delta > 0$, then the APSP Hypothesis fails.*

Theorem 1.5 indicates that (conditioned on the APSP Hypothesis and up to subpolynomial-factor improvements), for a dynamic algorithm with preprocessing time $\widetilde{\mathcal{O}}(nk^\gamma)$ for $\gamma \in [0.5, 1)$, the best possible update time is $\widetilde{\mathcal{O}}(k^{3-\gamma})$. The lower bound does not extend to $\kappa > 1/(3-2\gamma)$; precisely then, the $\widetilde{\mathcal{O}}(\sqrt{nk^3})$-time static algorithm improves upon the $\widetilde{\mathcal{O}}(k^{3-\gamma})$ update time.

Our Results: Algorithms

Our main positive result is an algorithm that achieves the fine-grained-optimal trade-off.

▶ **Theorem 1.6.** *There exists an algorithm that, initialized with an extra real parameter $\gamma \in [0, 1]$, solves Problem 1.4 with preprocessing time $\widetilde{\mathcal{O}}(nk^\gamma)$ and update time $\widetilde{\mathcal{O}}(k^{3-\gamma})$.*

As a warm-up, in Section 3, we prove Theorem 1.6 for $\gamma = 1$. Similarly to [15], the main challenge is that updates can make X and Y slide with respect to each other. If we interleave insertions at the end of Y and deletions at the beginning of Y, then, after $\Omega(k)$ updates, for every character $X[i]$, the fragment $Y[i-k..i+k]$ containing the plausible matches of $X[i]$ changes completely. The approach of [15] was to restart the algorithm every $\Theta(k)$ updates.

Our preprocessing time is too large for this, so we resort to a trick originating from dynamic edit distance approximation [20]: we maintain a data structure for aligning X with itself. Instead of storing Y, we show how to compute $\mathsf{ed}^w_{\le k}(X,Y)$ for any string Y specified using $\mathcal{O}(k)$ edits (of arbitrary costs) transforming X into Y. To find such edits in the original setting of Problem 1.4, one can use a dynamic *unweighted* edit distance algorithm. Already the folklore approach, based on [21, 23], with $\widetilde{\mathcal{O}}(k^2)$ time per update is sufficiently fast.

In Section 4, we discuss how to generalize our dynamic algorithm to arbitrary $\gamma \in [0,1]$. This is our most difficult result – it requires combining the techniques of [8] and their subsequent adaptations in [15] with several new ideas, including a novel *shifted* variant of self-edit distance. We derive Theorem 1.6 as a straightforward corollary of the following result, which demonstrates that the same fine-grained optimal trade-off can be achieved in a much more robust setting compared to the one presented in Problem 1.4.

▶ **Theorem 1.7.** *There is a dynamic algorithm that, initialized with a real parameter $\gamma \in [0,1]$, integers $1 \le k \le n$, and $\mathcal{O}(1)$-time oracle access to a normalized weight function $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\ge 0}$, maintains a string $X \in \Sigma^{\le n}$ and, after $\widetilde{\mathcal{O}}(nk^\gamma)$-time preprocessing, supports the following operations (updates and queries):*

- *Apply a character edit, substring deletion, or copy-paste to X in $\widetilde{\mathcal{O}}(k^2)$ time.*
- *Given u edits transforming X into a string Y, compute $\mathsf{ed}^w_{\le k}(X,Y)$ in $\widetilde{\mathcal{O}}(u + k^{3-\gamma})$ time.*

We remark that our query algorithm for $\gamma = 1$ extends a subroutine of [11], where the string X is static. In [11], this result is used for approximate pattern matching with respect to weighted edit distance. If the pattern is far from periodic, which is arguably true in most natural instances, the trade-off of Theorem 1.7 yields a faster pattern matching algorithm.

Open Problems

Recall that Theorem 1.5 proves fine-grained optimality of Theorem 1.6 only for $\gamma \in [0.5, 1)$. The omission of $\gamma \in [0, 0.5)$ stems from a gap between algorithms and lower bounds for the underlying static problem. Hence, we reiterate the open question of [8] asking for the complexity of computing $\mathsf{ed}^w(X,Y)$ when $\sqrt[3]{n} < \mathsf{ed}^w(X,Y) < \sqrt{n}$. Since our lower bounds arise from Theorem 1.2, it might be instructive to first try designing faster algorithms for the case of $\mathsf{ed}(X,Y) \le 4$. Before this succeeds, one cannot hope for faster dynamic algorithms.

Wider gaps in our understanding of Problem 1.4 remain in the regime of large preprocessing time. It is open to determine the optimal update time for $\gamma = 1$ and to decide whether $\gamma > 1$ can be beneficial. As far as $\gamma = 1$ is concerned, we believe that [9, Section 4] can be used to obtain $\widetilde{\mathcal{O}}(k\sqrt{n})$ update time, which improves upon Theorem 1.6 for $k \ge \sqrt{n}$. Moreover, by monotonicity, the lower bound of [8, Section 6] excludes update times of the form $\mathcal{O}(k^{1.5-\delta})$.

Finally, we remark that all our lower bounds exploit the fact that $\mathsf{ed}^w(X,Y)$ can change a lot as a result of a single update. It would be interesting to see what can be achieved if the weights are small (but fractional) or when the update time is parameterized by the change in $\mathsf{ed}^w(X,Y)$. In these cases, we hope for non-trivial applications of the approach of [15].

2 Preliminaries[3]

A string $X \in \Sigma^n$ is a sequence of $|X| := n$ characters over an alphabet Σ. We denote the set of all strings over Σ by Σ^*, we use ε to represent the empty string, and we write $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$. For a *position* $i \in [0..n)$, we say that $X[i]$ is the i-th character of X. We

[3] In this section, we partially follow the narration of [8, 15].

say that a string X occurs as a *substring* of a string Y, if there exist indices $i, j \in [0..|Y|]$ satisfying $i \leq j$ such that $X = Y[i] \cdots Y[j-1]$. The occurrence of X in Y specified by i and j is a *fragment* of Y denoted by $Y[i..j)$, $Y[i..j-1]$, $Y(i-1..j-1]$, or $Y(i-1..j)$.

Weighted Edit Distances and Alignment Graphs

Given an alphabet Σ, we set $\overline{\Sigma} \coloneqq \Sigma \cup \{\varepsilon\}$. We call w a *normalized weight function* if $w\colon \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_{\geq 1} \cup \{0\}$ and, for $a, b \in \overline{\Sigma}$, we have $w(a,b) = 0$ if and only if $a = b$. Note that w does not need to satisfy the triangle inequality nor does w need to be symmetric.

We interpret weighted edit distances of strings as distances in alignment graphs.

▶ **Definition 2.1** (Alignment Graph, [8]). *For strings $X, Y \in \Sigma^*$ and a normalized weight function $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, we define the* alignment graph $\mathsf{AG}^w(X,Y)$ *to be a weighted directed graph with vertices $[0..|X|] \times [0..|Y|]$ and the following edges:*

- *a horizontal edge $(x,y) \to (x+1,y)$ of cost $w(X[x],\varepsilon)$ representing a deletion of $X[x]$, for every $(x,y) \in [0..|X|) \times [0..|Y|]$,*
- *a vertical edge $(x,y) \to (x,y+1)$ of cost $w(\varepsilon,Y[y])$ representing an insertion of $Y[y]$, for every $(x,y) \in [0..|X|] \times [0..|Y|)$, and*
- *a diagonal edge $(x,y) \to (x+1,y+1)$ of cost $w(X[x],Y[y])$ representing a substitution of $X[x]$ into $Y[y]$ or, if $X[x] = Y[y]$, a match of $X[x]$ with $Y[y]$, for every $(x,y) \in [0..|X|) \times [0..|Y|)$.*

We visualize the graph $\mathsf{AG}^w(X,Y)$ as a grid graph with $|X|+1$ columns and $|Y|+1$ rows, where $(0,0)$ and $(|X|,|Y|)$ are the top-left and bottom-right vertices, respectively.

Alignments between fragments of X and Y can be interpreted as paths in $\mathsf{AG}^w(X,Y)$.

▶ **Definition 2.2** (Alignment). *Consider strings $X, Y \in \Sigma^*$ and a normalized weight function $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\geq 0}$. An* alignment $\mathcal{A}$ *of $X[x..x')$ onto $Y[y..y')$, denoted $\mathcal{A}\colon X[x..x') \rightsquigarrow Y[y..y')$, is a path from (x,y) to (x',y') in $\mathsf{AG}^w(X,Y)$, often interpreted as a sequence of vertices. The* cost *of $\mathcal{A}$, denoted by $\mathsf{ed}^w_{\mathcal{A}}(X[x..x'),Y[y..y'))$, is the total cost of edges belonging to $\mathcal{A}$.*

The *weighted edit distance* of strings $X, Y \in \Sigma^*$ with respect to a weight function w is $\mathsf{ed}^w(X,Y) \coloneqq \min \mathsf{ed}^w_{\mathcal{A}}(X,Y)$ where the minimum is taken over all alignments $\mathcal{A}$ of X onto Y. An alignment $\mathcal{A}$ of X onto Y is *w-optimal* if $\mathsf{ed}^w_{\mathcal{A}}(X,Y) = \mathsf{ed}^w(X,Y)$.

We say that an alignment $\mathcal{A}$ of X onto Y aligns fragments $X[x..x')$ and $Y[y..y')$ if $(x,y),(x',y') \in \mathcal{A}$. In this case, we also denote the cost of the *induced alignment* (the subpath of $\mathcal{A}$ from (x,y) to (x',y')) by $\mathsf{ed}^w_{\mathcal{A}}(X[x..x'),Y[y..y'))$.

A normalized weight function satisfying $w(a,b) = 1$ for distinct $a, b \in \overline{\Sigma}$ gives the *unweighted* edit distance (Levenshtein distance [22]). In this case, we drop the superscript w.

▶ **Definition 2.3** (Augmented Alignment Graph, [15]). *For strings $X, Y \in \Sigma^*$ and a weight function $w\colon \overline{\Sigma}^2 \to [0, W]$, the* augmented alignment graph $\overline{\mathsf{AG}}^w(X,Y)$ *is obtained from $\mathsf{AG}^w(X,Y)$ by adding, for every edge of $\mathsf{AG}^w(X,Y)$, a back edge of weight $W+1$.*[4]

The following fact shows several properties of $\overline{\mathsf{AG}}^w(X,Y)$. Importantly, $\mathsf{ed}^w(X[x..x'),Y[y..y'))$ is the distance from (x,y) to (x',y') in $\overline{\mathsf{AG}}^w(X,Y)$.

[4] These edges ensure that $\overline{\mathsf{AG}}^w(X,Y)$ is strongly connected, and thus the distance matrices used throughout this work are Monge matrices (see Definition 2.11) rather than so-called partial Monge matrices.

▶ **Fact 2.4** ([15, Lemma 5.2]). *Consider strings $X, Y \in \Sigma^*$ and a weight function $w\colon \overline{\Sigma}^2 \to [0, W]$. Every two vertices (x, y) and (x', y') of the graph $\overline{\mathsf{AG}}^w(X, Y)$ satisfy the following properties:*

Monotonicity. *Every shortest path from (x, y) to (x', y') in $\overline{\mathsf{AG}}^w(X, Y)$ is (weakly) monotone in both coordinates.*

Distance preservation. *If $x \le x'$ and $y \le y'$, then*

$$\mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}((x, y), (x', y')) = \mathsf{dist}_{\mathsf{AG}^w(X,Y)}((x, y), (x', y')).$$

Path irrelevance. *If ($x \le x'$ and $y \ge y'$) or ($x \ge x'$ and $y \le y'$), then every path from (x, y) to (x', y') in $\overline{\mathsf{AG}}^w(X, Y)$ monotone in both coordinates is a shortest path between these two vertices.*

Consistently with many edit-distance algorithms, we will often compute not only the distance from $(0, 0)$ to $(|X|, |Y|)$ in $\overline{\mathsf{AG}}^w(X, Y)$ but the entire *boundary distance matrix* $\mathsf{BM}^w(X, Y)$ that stores the distances from every *input* vertex on the top-left boundary to every *output* vertex on the bottom-right boundary of the graph $\overline{\mathsf{AG}}^w(X, Y)$.

▶ **Definition 2.5** (Input and Output Vertices of an Alignment Graph, [15, Definition 5.5]). *Let $X, Y \in \Sigma^*$ be two strings and $w\colon \overline{\Sigma}^2 \to [0, W]$ be a weight function. Furthermore, for fragments $X[x..x')$ and $Y[y..y')$, let $R = [x..x'] \times [y..y']$ be the corresponding* rectangle *in $\overline{\mathsf{AG}}^w(X, Y)$. We define the* input vertices *of R as the sequence of vertices of $\overline{\mathsf{AG}}^w(X, Y)$ on the left and top boundary of R in the clockwise order. Analogously, we define the* output vertices *of R as a sequence of vertices of $\overline{\mathsf{AG}}^w(X, Y)$ on the bottom and right boundary of R in the counterclockwise order. Furthermore, the input and output vertices of $X \times Y$ are the corresponding boundary vertices of the whole graph $\overline{\mathsf{AG}}^w(X, Y)$.*

▶ **Definition 2.6** (Boundary Distance Matrix, [15, Definition 5.6]). *Let $X, Y \in \Sigma^*$ be two strings and $w\colon \overline{\Sigma}^2 \to [0, W]$ be a weight function. The* **b**oundary distance **m**atrix *$\mathsf{BM}^w(X, Y)$ of X and Y with respect to w is a matrix M of size $(|X|+|Y|+1) \times (|X|+|Y|+1)$, where $M_{i,j}$ is the distance from the i-th input vertex to the j-th output vertex of $X \times Y$ in $\overline{\mathsf{AG}}^w(X, Y)$.*

The following fact captures the simple dynamic programming algorithm for bounded weighted edit distance.

▶ **Fact 2.7** ([15, Fact 3.3]). *Given strings $X, Y \in \Sigma^*$, an integer $k \ge 1$, and $\mathcal{O}(1)$-time oracle access to a normalized weight function $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\ge 0}$, the value $\mathsf{ed}^w_{\le k}(X, Y)$ can be computed in $\mathcal{O}(\min(|X| + 1, |Y| + 1) \cdot \min(k, |X| + |Y| + 1))$ time. Furthermore, if $\mathsf{ed}^w(X, Y) \le k$, the algorithm returns a w-optimal alignment of X onto Y.*

The proof of Fact 2.7 relies on the fact that a path of length k can visit only a limited area of the alignment graph $\mathsf{AG}^w(X, Y)$. This property can be formalized as follows:

▶ **Fact 2.8** ([15, Lemma 5.3]). *Let $X, Y \in \Sigma^*$ be two strings, k be an integer, and $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\ge 0}$ be a normalized weight function. Consider a path P of cost at most k connecting (x, y) to (x', y') in $\overline{\mathsf{AG}}^w(X, Y)$. All vertices $(x^*, y^*) \in P$ satisfy $|(x - y) - (x^* - y^*)| \le k$ and $|(x' - y') - (x^* - y^*)| \le k$.*

The *breakpoint representation* of an alignment $\mathcal{A} = (x_t, y_t)_{t=0}^m$ of X onto Y is the subsequence of $\mathcal{A}$ consisting of pairs (x_t, y_t) such that $t \in \{0, m\}$ or $\mathcal{A}$ does not match $X[x_t]$ with $Y[y_t]$. Note that the size of the breakpoint representation is at most $2 + \mathsf{ed}_{\mathcal{A}}(X, Y)$ and that it can be used to retrieve the entire alignment and its cost: for any two consecutive

elements $(x', y'), (x, y)$ of the breakpoint representation, it suffices to add $(x - \delta, y - \delta)$ for $\delta \in (0..\max(x - x', y - y'))$. We will also talk of *breakpoint representations* of paths in $\overline{\mathsf{AG}}^w(X, Y)$ that are weakly monotone in both coordinates. If such a path starts in (x, y) and ends in (x', y') with $x \leq x'$ and $y \leq y'$, then the breakpoint representation of such a path is identical to the breakpoint representation of the corresponding alignment. Otherwise, if $x > x'$ or $y > y'$, we call the breakpoint representation of such a path a sequence of all its vertices; recall that all edges in such a path have strictly positive weights.

Planar Graphs, the Monge Property, and Min-Plus Matrix Multiplication

As alignment graphs are planar, we can use the following tool of Klein.

▶ **Fact 2.9** (Klein [19]). *Given a directed planar graph G of size n with non-negative edge weights, we can construct in $\mathcal{O}(n \log n)$ time a data structure that, given two vertices $u, v \in V(G)$ at least one of which lies on the outer face of G, computes the distance $\mathsf{dist}_G(u, v)$ in $\mathcal{O}(\log n)$ time. Moreover, the data structure can report the shortest $u \rightsquigarrow v$ path P in $\mathcal{O}(|P| \log\log \Delta(G))$ time, where $\Delta(G)$ is the maximum degree of G.*

The following lemma specifies a straightforward variation of Klein's algorithm tailored for path reconstruction in alignment graphs.

▶ **Lemma 2.10.** *Given strings $X, Y \in \Sigma^+$ and $\mathcal{O}(1)$-time access to a normalized weight function $w\colon \overline{\Sigma}^2 \to [0, W]$, we can construct in $\mathcal{O}(|X| \cdot |Y| \log^2(|X| + |Y|))$ time a data structure that, given two vertices $u, v \in \overline{\mathsf{AG}}^w(X, Y)$ at least one of which lies on the outer face of $\overline{\mathsf{AG}}^w(X, Y)$, computes the distance $\mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}(u, v)$ in $\mathcal{O}(\log(|X| + |Y|))$ time. Moreover, the breakpoint representation of a shortest $u \rightsquigarrow v$ path can be constructed in $\mathcal{O}((1 + \mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}(u, v)) \log^2(|X| + |Y|))$ time.*

Proof. The data structure we build is recursive. We first build the data structure of Klein's algorithm (Fact 2.9) for $\overline{\mathsf{AG}}^w(X, Y)$, and then if $|X| > 1$, recurse onto $(X[0..\lfloor |X|/2 \rfloor), Y)$ and $(X[\lfloor |X|/2 \rfloor..|X|), Y)$. Preprocessing costs $\mathcal{O}(|X| \cdot |Y| \log^2(|X| + |Y|))$ time. For distance queries, the top-level instance of Klein's algorithm is sufficient. It remains to answer path reconstruction queries. We reconstruct the shortest $u \rightsquigarrow v$ path recursively. First, in $\mathcal{O}(\log(|X| + |Y|))$ time we query $\mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}(u, v)$. If the distance is 0, we return the breakpoint representation of the trivial shortest $u \rightsquigarrow v$ path. Otherwise, we make further recursive calls. If u and v both lie in $X[0..\lfloor |X|/2 \rfloor) \times Y$ or both lie in $X[\lfloor |X|/2 \rfloor..|X|) \times Y$, by the monotonicity property of Fact 2.4 we can recurse onto the corresponding small recursive data structure instance. Otherwise, every $u \rightsquigarrow v$ path contains a vertex (x, y) with $x = \lfloor |X|/2 \rfloor$. Furthermore, due to Fact 2.8, there are $\mathcal{O}(\mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}(u, v))$ possible values for y. In $\mathcal{O}(\mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}(u, v) \log(|X| + |Y|))$ time we query $\mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}(u, (x, y))$ and $\mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}((x, y), v)$ using the smaller recursive data structure instances and find some y satisfying $\mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}(u, v) = \mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}(u, (x, y)) + \mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}((x, y), v)$. We then recursively find the breakpoint representations of the optimal $u \rightsquigarrow (x, y)$ path and the optimal $(x, y) \rightsquigarrow v$ path in the smaller recursive data structure instances. There are $\mathcal{O}(\log(|X| + 1))$ recursive levels, on each one of them our total work is $\mathcal{O}((1+\mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}(u, v)) \log(|X|+|Y|))$; thus, the total time complexity is $\mathcal{O}((1 + \mathsf{dist}_{\overline{\mathsf{AG}}^w(X,Y)}(u, v)) \log^2(|X| + |Y|))$. ◀

As discussed in [13, Section 2.3] and [15, Fact 3.7 and Section 5.1], the planarity of alignment graphs implies that the $\mathsf{BM}^w(X, Y)$ matrices satisfy the following *Monge property*:

▶ **Definition 2.11** (Monge Matrix). *A matrix A of size $p \times q$ is a* Monge *matrix if, for all $i \in [0..p - 1)$ and $j \in [0..q - 1)$, we have $A_{i,j} + A_{i+1,j+1} \leq A_{i,j+1} + A_{i+1,j}$.*

In the context of distance matrices, concatenating paths corresponds to the min-plus product of matrices; this operation preserves Monge property and can be evaluated fast.

▶ **Fact 2.12** (Min-Plus Matrix Product, SMAWK Algorithm [3], [29, Theorem 2]). *Let A and B be matrices of size $p \times q$ and $q \times r$, respectively. Their* min-plus product *is a matrix $A \otimes B \coloneqq C$ of size $p \times r$ such that $C_{i,k} = \min_j A_{i,j} + B_{j,k}$ for all $i \in [0..p)$ and $k \in [0..r)$.*

If A and B are Monge matrices, then C is also a Monge matrix. Moreover, C can be constructed in $\mathcal{O}(pr + \min(pq, qr))$ time assuming $\mathcal{O}(1)$-time random access to A and B.

3 $\widetilde{\mathcal{O}}(k^2)$-Time Updates after $\widetilde{\mathcal{O}}(nk)$-Time Preprocessing

In this section, we present a dynamic algorithm that maintains $\mathsf{ed}^w_{\le k}(X, Z)$ for two dynamic strings $X, Z \in \Sigma^{\le n}$ with $\widetilde{\mathcal{O}}(nk)$ preprocessing time and $\widetilde{\mathcal{O}}(k^2)$ update time. As all vertical and horizontal edges cost at least 1, the relevant part of the graph $\overline{\mathsf{AG}}^w(X, Z)$ has size $\mathcal{O}(nk)$ (i.e., a band of width $\Theta(k)$ around the main diagonal), and thus we can afford to preprocess this part of $\overline{\mathsf{AG}}^w(X, Z)$. Nevertheless, the curse of dynamic algorithms for bounded weighted edit distance is that, after $\Theta(k)$ updates (e.g., $\Theta(k)$ character deletions from the end of Z and $\Theta(k)$ character insertions at the beginning of Z), the preprocessed part of the original graph $\overline{\mathsf{AG}}^w(X, Z)$ can be completely disjoint from the relevant part of the updated graph $\overline{\mathsf{AG}}^w(X, Z)$, which renders the original preprocessing useless. To counteract this challenge, we use the idea from [20]: we dynamically maintain information about $\overline{\mathsf{AG}}^w(X, X)$ instead of $\overline{\mathsf{AG}}^w(X, Z)$. Every update to X now affects both dimensions of $\overline{\mathsf{AG}}^w(X, X)$, so the relevant part of the graph does not shift anymore, and hence our preprocessing does not expire. Tasked with computing $\mathsf{ed}^w_{\le k}(X, Z)$ upon a query, we rely on the fact that, unless $\mathsf{ed}^w_{\le k}(X, Z) = \infty$, the graph $\overline{\mathsf{AG}}^w(X, Z)$ is similar to $\overline{\mathsf{AG}}^w(X, X)$.

Consistently with the previous works [8, 15], we cover the relevant part of $\overline{\mathsf{AG}}^w(X, X)$ with $m = \mathcal{O}(n/k)$ rectangular subgraphs G_i of size $\Theta(k) \times \Theta(k)$ each; see Figure 1. We preprocess each subgraph in $\widetilde{\mathcal{O}}(k^2)$ time. A single update to X alters a constant number of subgraphs G_i, so we can repeat the preprocessing for such subgraphs in $\widetilde{\mathcal{O}}(k^2)$ time. The string Z does not affect any subgraph G_i; we only store it in a dynamic strings data structure supporting efficient substring equality queries for the concatenation $X \cdot Z$.

Let V_i for $i \in [1..m)$ be the set of vertices in the intersection of G_i and G_{i+1}; see Figure 1. Furthermore, let $V_0 \coloneqq \{(0, 0)\}$ and $V_m \coloneqq \{(|X|, |X|)\}$. Let G be the union of all graphs G_i. Let $D_{i,j}$ for $i, j \in [0..m]$ with $i < j$ denote the matrix of pairwise distances from V_i to V_j in G. Note that $D_{a,c} = D_{a,b} \otimes D_{b,c}$ holds for all $a, b, c \in [0..m]$ with $a < b < c$.

Upon a query for $\mathsf{ed}^w(X, Z)$, we note that only $\mathcal{O}(\mathsf{ed}(X, Z))$ subgraphs G_i in $\overline{\mathsf{AG}}^w(X, X)$ differ from the corresponding subgraphs in $\overline{\mathsf{AG}}^w(X, Z)$. (If $\mathsf{ed}(X, Z) > k$, then $\mathsf{ed}^w_{\le k}(X, Z) = \infty$, so we can assume that $\mathsf{ed}(X, Z) \le k$.) We call such subgraphs *affected*. Let G'_i, V'_i, and $D'_{i,j}$ be the analogs of G_i, V_i, and $D_{i,j}$ in $\overline{\mathsf{AG}}^w(X, Z)$. Note that if $\mathsf{ed}^w(X, Z) \le k$, then $\mathsf{ed}^w(X, Z)$ is the only entry of the matrix $D'_{0,m}$. Furthermore, we have $D'_{0,m} = D'_{0,1} \otimes \cdots \otimes D'_{m-1,m}$. We compute this product from left to right. For unaffected subgraphs G_i, we have $D'_{i-1,i} = D_{i-1,i}$, and we can use precomputed information about $\overline{\mathsf{AG}}^w(X, X)$. To multiply by $D'_{i-1,i}$ for affected subgraphs G_i, we use the following lemma, the proof of which can be found in the full version of the paper [6].

▶ **Lemma 3.1** (Generalization of [11, Claim 4.2]). *Let $X, Y \in \Sigma^+$ be nonempty strings and $w\colon \overline{\Sigma}^2 \to [0, W]$ be a weight function. Given $\mathcal{O}(1)$-time oracle access to w, the strings X and Y can be preprocessed in $\mathcal{O}(|X| \cdot |Y| \log^2(|X| + |Y|))$ time so that, given a string $Y' \in \Sigma^*$ and a vector v of size $|X| + |Y'| + 1$, the row $v^T \otimes \mathsf{BM}^w(X, Y')$ can be computed in*

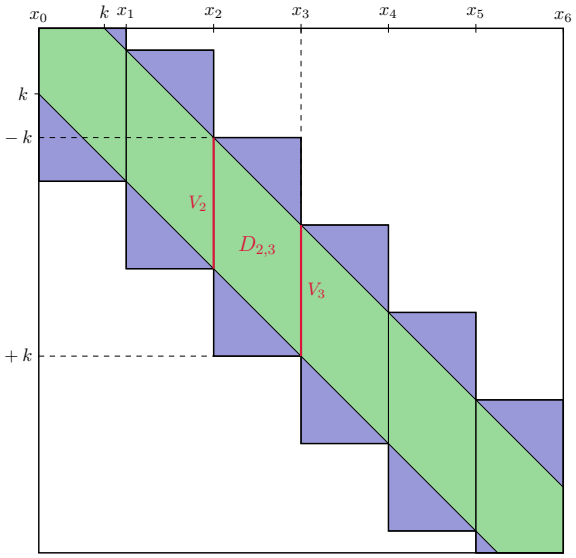


Figure 1 The decomposition of the alignment graph of X onto X into subgraphs G_i of size $\Theta(k) \times \Theta(k)$ that cover the whole stripe of width $\Theta(k)$ around the main diagonal. Each matrix $D_{i,i+1}$ represents the pairwise distances from V_i to V_{i+1}.

$\mathcal{O}((\mathsf{ed}(Y, Y') + 1) \cdot (|X| + |Y|) \log(|X| + |Y|))$ *time. Furthermore, given an input vertex* a *and an output vertex* b *of* $X \times Y'$*, the shortest path from* a *to* b *in* $\overline{\mathsf{AG}}^w(X, Y')$ *can be computed in the same time complexity.*

There are $\mathsf{ed}(X, Z)$ edits that transform $\overline{\mathsf{AG}}^w(X, X)$ into $\overline{\mathsf{AG}}^w(X, Z)$, and each edit affects a constant number of subgraphs G_i. Therefore, applying Lemma 3.1 for all affected subgraphs takes time $\widetilde{\mathcal{O}}(k^2)$ in total. Between two consecutive affected subgraphs G_i and G_j, we have $D'_{i,j-1} = D_{i,j-1}$, and thus it is sufficient to store some range composition data structure over $(D_{i,i+1})_{i=0}^{m-1}$ to quickly propagate between subsequent affected subgraphs.

As the information we maintain dynamically regards $\overline{\mathsf{AG}}^w(X, X)$ and does not involve the string Z, we may generalize the problem we are solving. All we need upon a query is an alignment of X onto Z of *unweighted* cost at most k. Such an alignment can be obtained, for example, from a dynamic bounded unweighted edit distance algorithm [15] in $\widetilde{\mathcal{O}}(k)$ time per update. Alternatively, it is sufficient to have $\widetilde{\mathcal{O}}(1)$-time equality tests between fragments of X and Z. In this case, we can run the classical Landau–Vishkin [21] algorithm from scratch after every update in $\widetilde{\mathcal{O}}(k^2)$ time to either get an optimal unweighted alignment or learn that $\mathsf{ed}^w(X, Z) \geq \mathsf{ed}(X, Z) > k$. Efficient fragment equality tests are possible in a large variety of settings [10] including the dynamic setting [23, 4, 14, 18], so we will not focus on any specific implementation. Instead, we assume that the string Z is already given as a sequence of $u \leq k$ edits of an unweighted alignment transforming X into Z, and our task is to find a w-optimal alignment. We are now ready to formulate the main theorem of this section.

▶ **Theorem 3.2.** *There is a dynamic data structure that, given a string* $X \in \Sigma^*$*, a threshold* $k \geq 1$*, and* $\mathcal{O}(1)$*-time oracle access to a normalized weight function* $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\geq 0}$*, can be initialized in* $\mathcal{O}((|X| + 1)k \log^2(|X| + 2))$ *time and allows for the following operations:*

- *Apply a character edit to* X *in* $\mathcal{O}(k^2 \log^2(|X| + 2))$ *time.*
- *Given* $u \leq k$ *edits transforming* X *into a string* $Z \in \Sigma^*$*, compute* $\mathsf{ed}^w_{\leq k}(X, Z)$ *in* $\mathcal{O}(k \cdot (u + \log(|X| + 2)) \cdot \log(|X| + 2))$ *time. Furthermore, if* $\mathsf{ed}^w(X, Z) \leq k$*, the algorithm returns the breakpoint representation of a* w*-optimal alignment of* X *onto* Z*.*

Proof Sketch. At the initialization phase, we decompose $\overline{\mathsf{AG}}^w(X,X)$ into subgraphs G_i and, for each subgraph G_i, initialize the algorithm of Lemma 3.1 and compute $D_{i-1,i}$ using Lemma 2.10. We also build a dynamic range composition data structure over $(D_{i,i+1})_{i=0}^{m-1}$, implemented as a self-balancing binary tree. Given a range $[a..b)$, in $\mathcal{O}(\log n)$ time, this data structure returns (pointers to) $\ell = \mathcal{O}(\log n)$ matrices $S_1, \ldots, S_\ell$ such that $D_{a,b} = S_1 \otimes \cdots \otimes S_\ell$.

When an update to X comes, it affects a constant number of subgraphs G_i, which we locally adjust and recompute the initialization for in $\widetilde{\mathcal{O}}(k^2)$ time.

Upon a query, we are tasked with computing $D'_{0,1} \otimes \cdots \otimes D'_{m-1,m}$. We locate the $\mathcal{O}(u)$ affected subgraphs G_i and process them one-by-one. We use Lemma 3.1 to multiply the currently computed row $v^T = D'_{0,1} \otimes \cdots \otimes D'_{i-2,i-1}$ by $D'_{i-1,i}$. The algorithm then uses the range composition data structure over $(D_{i,i+1})_{i=0}^{m-1}$ to obtain $\ell = \mathcal{O}(\log n)$ matrices $S_1, \ldots, S_\ell$ that correspond to the segment of matrices between the current affected subgraph G_i and the next affected subgraph G_j. We use SMAWK algorithm (Fact 2.12) to multiply v^T by matrices $S_1, \ldots, S_\ell$. After processing all the matrices, we obtain $D'_{0,1} \otimes \cdots \otimes D'_{m-1,m}$, the only entry of which is equal to $\mathsf{ed}^w(X,Z)$ if $\mathsf{ed}^w_{\le k}(X,Z) \neq \infty$. The applications of Lemma 3.1 work in $\mathcal{O}(ku \log n)$ time in total. Each of the $\mathcal{O}(u)$ blocks of unaffected subgraphs is treated by invoking SMAWK algorithm (Fact 2.12) $\mathcal{O}(\log n)$ times for a total of $\mathcal{O}(ku \log n)$ time.

See the full version of the paper [6] for a formal proof including the alignment reconstruction procedure. ◀

We complete this section with a simple corollary improving the query time if $\mathsf{ed}^w(X,Z) \ll k$.

▶ **Corollary 3.3.** *There is a dynamic data structure that, given a string $X \in \Sigma^*$, a threshold $k \ge 1$, and $\mathcal{O}(1)$-time oracle access to a normalized weight function $w\colon \Sigma^2 \to \mathbb{R}_{\ge 0}$ can be initialized in $\mathcal{O}((|X|+1)k\log^2(|X|+2))$ time and allows for the following operations:*

- *Apply a character edit to X in $\mathcal{O}(k^2\log^2(|X|+2))$ time.*
- *Given $u \le k$ edits transforming X into a string $Z \in \Sigma^*$, compute $\mathsf{ed}^w_{\le k}(X,Z)$ in $\mathcal{O}(1 + (u+d)\cdot(u+\log(|X|+2))\cdot\log(|X|+2))$ time, where $d := \min(\mathsf{ed}^w(X,Z), k)$. Furthermore, if $\mathsf{ed}^w(X,Z) \le k$, the algorithm returns the breakpoint representation of a w-optimal alignment of X onto Z.*

Proof. We maintain the data structures of Theorem 3.2 for thresholds $1, 2, 4, \ldots, 2^{\lceil \log k \rceil}$. Upon a query, we query the data structures for subsequent thresholds starting from $2^{\lceil \log u \rceil}$ until we get a positive result or reach $2^{\lceil \log k \rceil}$. ◀

As discussed earlier, in a variety of settings, upon a query, in $\widetilde{\mathcal{O}}(k^2)$ time we can compute an optimal unweighted alignment of X onto Z or learn that $\mathsf{ed}^w(X,Z) \ge \mathsf{ed}(X,Z) > k$. Therefore, we may assume that whenever Corollary 3.3 is queried, we have $u = \mathsf{ed}(X,Z) \le k$. In this case, the query time complexity can be rewritten as $\mathcal{O}(1 + \min(\mathsf{ed}^w(X,Z), k) \cdot (\mathsf{ed}(X,Z) + \log(|X|+2)) \cdot \log(|X|+2))$.

Due to the robustness of the setting of Corollary 3.3, there are many problems beyond dynamic weighted edit distance that can be solved using Corollary 3.3. For example, given strings $X, Y_0, \ldots, Y_{m-1} \in \Sigma^{\le n}$, we can find $\mathsf{ed}^w_{\le k}(X, Y_i)$ for all $i \in [0..m)$ in $\widetilde{\mathcal{O}}(nm+nk+mk^2)$ time, compared to $\widetilde{\mathcal{O}}(m \cdot (n + \sqrt{nk^3}))$ time arising from m applications of the optimal static algorithm [8].

4 Trade-Off Algorithm: Technical Overview

In the full version of the paper [6], we generalize the result of Theorem 3.2 in two ways. First, we extend the set of updates the string X supports from character edits to *block edits*, which include substring deletions and copy-pastes. More importantly, rather than having $\widetilde{\mathcal{O}}(nk)$

preprocessing time and $\widetilde{\mathcal{O}}(k^2)$ query time, we allow for a complete trade-off matching the lower bound of Theorem 1.5, i.e., $\widetilde{\mathcal{O}}(nk^\gamma)$ preprocessing and $\widetilde{\mathcal{O}}(k^{3-\gamma})$ update time for any $\gamma \in [0,1]$.

A major challenge for $\gamma < 1$ is that we cannot preprocess the whole width-$\Theta(k)$ band around the main diagonal of $\overline{\mathsf{AG}}^w(X,X)$. In the proof of Theorem 3.2, we cover the band with $\Theta(k)\times\Theta(k)$-sized subgraphs G_i and preprocess each of them in $\widetilde{\mathcal{O}}(k^2)$ time for a total of $\widetilde{\mathcal{O}}(nk)$. To achieve $\widetilde{\mathcal{O}}(nk^\gamma)$-time preprocessing, we are forced to instead use $\Theta(k^{2-\gamma})\times\Theta(k^{2-\gamma})$-sized subgraphs G_i and still spend just $\widetilde{\mathcal{O}}(k^2)$ preprocessing time per subgraph.

To remedy the shortage of preprocessing time, we use the ideas behind the recent static algorithms [8, 15]. At the expense of a logarithmic-factor slowdown, they allow us to answer queries only for "repetitive" fragments of X rather than the whole X. More precisely, we are tasked to answer queries involving fragments $\hat{X}$ of X satisfying $\mathsf{self\text{-}ed}(\hat{X}) \le k$, where the *self-edit distance* $\mathsf{self\text{-}ed}(\hat{X})$ is the minimum unweighted cost of an alignment of $\hat{X}$ onto itself that does not match any character of $\hat{X}$ with itself. We can use this to our advantage: the subgraphs G_i corresponding to more "repetitive" fragments X_i allow for more thorough preprocessing in $\widetilde{\mathcal{O}}(k^2)$ time since we can reuse some computations. On the other hand, upon a query, we can afford to spend more time on subgraphs G_i with less "repetitive" fragments X_i as their number is limited due to $\mathsf{self\text{-}ed}(\hat{X}) \le k$.

This approach requires the repetitiveness measure to be super-additive: $\sum_i \mathsf{self\text{-}ed}(X_i) \le \mathsf{self\text{-}ed}(\hat{X})$. Unfortunately, self-edit distance is sub-additive: $\sum_i \mathsf{self\text{-}ed}(X_i) \ge \mathsf{self\text{-}ed}(\hat{X})$. To circumvent this issue, we introduce a better-suited notion of repetitiveness we call *k-shifted self-edit distance* $\mathsf{self\text{-}ed}^k(\hat{X})$: a relaxed version of $\mathsf{self\text{-}ed}(\hat{X})$ allowed to delete up to k first characters of $\hat{X}$ and insert up to k last characters of $\hat{X}$ for free. In contrast to self-edit distance, the k-shifted variant $\mathsf{self\text{-}ed}^k$ satisfies $\sum_i \mathsf{self\text{-}ed}^k(X_i) \le \mathsf{self\text{-}ed}(\hat{X})$ if $\mathsf{self\text{-}ed}(\hat{X}) \le k$.

By using the value of $\mathsf{self\text{-}ed}^k(X_i)$ to determine the level of preprocessing performed on G_i, upon a query, similarly to Theorem 3.2, we have $\mathcal{O}(k)$ "interesting" subgraphs G_i (subgraphs that are either affected by the input edits or satisfy $\mathsf{self\text{-}ed}^k(X_i) > 0$) that we process one-by-one in time $\widetilde{\mathcal{O}}((u_i + \mathsf{self\text{-}ed}^k(X_i))\cdot k^{2-\gamma})$ each using a generalization of Lemma 3.1. The super-additivity property of $\mathsf{self\text{-}ed}^k$ guarantees that this sums up to a total of $\widetilde{\mathcal{O}}(k^{3-\gamma})$. The remaining subgraphs G_i are not affected by the input edits and satisfy $\mathsf{self\text{-}ed}^k(X_i) = 0$, which means that X_i has a period of at most k. For such subgraphs G_i, we can afford the complete preprocessing and, upon a query, processing such subgraphs in chunks in $\widetilde{\mathcal{O}}(k^2)$ time in total precisely as in the proof of Theorem 3.2. Furthermore, similarly to Theorem 3.2, all the information the data structure stores is either local or cumulative, and thus block edits can be supported straightforwardly. This rather simple high-level approach runs into several low-level technical complications that are discussed in detail in the full version of the paper [6].

5 Lower Bounds: Technical Overview

As discussed in Section 1, our conditional lower bounds stem from the following result:

▶ **Theorem 1.2.** *Let $\kappa \in [0.5, 1]$ and $\delta > 0$ be real parameters. Assuming the APSP Hypothesis, there is no algorithm that, given two strings $X, Y \in \Sigma^{\le n}$ satisfying $\mathsf{ed}(X,Y) \le 4$, a threshold $k \le n^\kappa$, and oracle access to a normalized weight function $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\ge 0}$, in time $\mathcal{O}(n^{0.5+1.5\kappa-\delta})$ decides whether $\mathsf{ed}^w(X,Y) \le k$.*

The reduction provided in [8] is insufficient because the instances it produces satisfy $\mathsf{ed}(X,Y) = \Theta(k)$ rather than $\mathsf{ed}(X,Y) \le 4$. Still, we reuse hardness of the following problem:

▶ **Problem 5.1** (Batched Weighted Edit Distance [8]). *Given a* batch *of strings* $X_1, \ldots, X_m \in \Sigma^x$*, a string* $Y \in \Sigma^y$*, a threshold* $k \in \mathbb{R}_{\geq 0}$*, and oracle access to a weight function* $w : \overline{\Sigma}^2 \to \mathbb{R}_{\geq 0}$*, decide if* $\min_{i=1}^{m} \mathsf{ed}^w(X_i, Y) \leq k$.

The hard instances of Problem 5.1, which cannot be solved in $\mathcal{O}(x^{2-\delta}\sqrt{m})$ time for any $\delta > 0$ assuming the APSP Hypothesis, satisfy many technical properties, including $h := \max_{i=1}^{m-1} \mathsf{hd}(X_i, X_{i+1}) = \mathcal{O}(x/m)$, where $\mathsf{hd}(X_i, X_{i+1})$ is the Hamming distance, i.e., the number of positions in which X_i differs from X_{i+1}.

The approach of [8, Section 7] is to construct strings $\bar{X}$ and $\bar{Y}$ that take the following form if we ignore some extra gadgets: $\bar{X} = X_1 Y X_2 \cdots X_{m-1} Y X_m$ and $\bar{Y} = X_0^{\perp} Y X_1^{\perp} \cdots X_{m-1}^{\perp} Y X_m^{\perp}$, where $X_i^{\perp} \in \Sigma^x$ is chosen so that $\mathsf{ed}^w(X_i^{\perp}, X_{i-1}) = \mathsf{ed}^w(X_i^{\perp}, X_i) = h$. The ignored gadgets let us focus on alignments $\mathcal{A}_i : \bar{X} \rightsquigarrow \bar{Y}$ that satisfy the following properties for $i \in [1..m]$:

- the fragment X_i is aligned with the copy of Y in $\bar{Y}$ located between $X_{i-1}^{\perp}$ and $X_i^{\perp}$;
- the $m-1$ copies of Y in $\bar{X}$ are matched with the remaining $m-1$ copies of Y in $\bar{Y}$;
- all the characters within the fragments $X_{i-1}^{\perp}$ and $X_i^{\perp}$ of $\bar{Y}$ are inserted;
- for $j \neq i$, the fragment X_j is aligned with $X_{j-1}^{\perp}$ if $j < i$ and with $X_j^{\perp}$ if $j > i$.

This ensures that the cost of $\mathcal{A}_i$ is equal to a baseline cost (independent of i) plus $\mathsf{ed}^w(X_i, Y)$.

Our reduction uses three types of "X gadgets": X_i, $X_i^{\perp}$, and $X_i^{\top}$, with the latter two playing similar roles. We also introduce a new symbol \$ that is extremely expensive to delete, insert, or substitute with any other symbol. Modulo some extra gadgets, our strings are of the following form:

$$\tilde{X} = X_0^{\top} Y X_0^{\perp} Y \$ \bigodot_{i=1}^{m-1} \Big(X_i Y X_i^{\top} Y X_i^{\perp} Y \Big) X_m Y X_m^{\top} Y X_m^{\perp} \$,$$

$$\tilde{Y} = \$ X_0^{\top} Y X_0^{\perp} Y \bigodot_{i=1}^{m-1} \Big(X_i Y X_i^{\top} Y X_i^{\perp} Y \Big) X_m Y X_m^{\top} \$ Y X_m^{\perp}.$$

Notice that removing the two \$ symbols from $\tilde{X}$ or from $\tilde{Y}$ results in the same string, so we have $\mathsf{ed}(\tilde{X}, \tilde{Y}) \leq 4$. Due to the expensive cost for editing a \$, any optimal alignment must match the two \$ symbols in $\tilde{X}$ with the two \$ symbols in $\tilde{Y}$, deleting a prefix of $\tilde{X}$ and a suffix of $\tilde{Y}$ for some predictable cost. Thus, we can focus our analysis on $\mathsf{ed}^w(\hat{X}, \hat{Y})$, where

$$\hat{X} = X_1 Y X_1^{\top} Y X_1^{\perp} Y \cdots X_m Y X_m^{\top} Y X_m^{\perp} \quad \text{and} \quad \hat{Y} = X_0^{\top} Y X_0^{\perp} Y X_1 Y \cdots X_{m-1}^{\perp} Y X_m Y X_m^{\top}.$$

Observe that both $\hat{X}$ and $\hat{Y}$ consists of "X gadgets" interleaved with Y, and that $\hat{Y}$ contains one more copy of Y and one more "X gadget". Analogously to [8], the ignored extra gadgets let us focus on the alignments $\mathcal{A}_i : \hat{X} \rightsquigarrow \hat{Y}$ satisfying the following properties for $i \in [1..m]$:

- the fragment X_i in $\hat{X}$ is aligned with the copy of Y in $\hat{Y}$ located between $X_{i-1}^{\perp}$ and $X_{i-1}^{\top}$;
- the $3m-1$ copies of Y in $\hat{X}$ are matched with the remaining $3m-1$ copies of Y in $\hat{Y}$;
- all characters within the fragments $X_{i-1}^{\perp}$ and $X_{i-1}^{\top}$ of $\hat{Y}$ are inserted;
- the remaining "X gadgets" in $\hat{X}$ are aligned with the remaining "X gadgets" in $\hat{Y}$.

To perfectly mimic [8], we should ensure that all the "X gadgets" that we can possibly align are at weighted edit distance h. We only managed to construct $X_i^{\perp}$ and $X_i^{\top}$ so that:

- $\mathsf{ed}^w(X_i, X_{i-1}^{\top}) = \mathsf{ed}^w(X_i, X_{i-1}^{\perp}) = \mathsf{ed}^w(X_i^{\top}, X_i) = \mathsf{ed}^w(X_i^{\perp}, X_i) = 2h$, and
- $\mathsf{ed}^w(X_i^{\top}, X_{i-1}^{\perp}) = \mathsf{ed}^w(X_i^{\perp}, X_i^{\top}) = 4h$.

Although we have two different distances between the relevant pairs of "X gadgets", it is easy to see the number of pairs of either type is constant across all the alignments $\mathcal{A}_i$. This is sufficient to guarantee that the cost of $\mathcal{A}_i$ is equal to some baseline cost plus $\mathsf{ed}^w(X_i, Y)$. Moreover, as in [8], the costs of alignments $\mathcal{A}_i$ is $\mathcal{O}(x + mh) = \mathcal{O}(x)$ and, if we denote $n := |\tilde{X}| = |\tilde{Y}|$, then the lower bound derived from the lower bound for Problem 5.1 excludes running times of the form $\mathcal{O}(x^{2-\delta}\sqrt{m}) = \mathcal{O}(x^{2-\delta}\sqrt{n/x}) = \mathcal{O}(\sqrt{nx^{3-2\delta}})$.

Lower Bounds for Dynamic Weighed Edit Distance

From Theorem 1.2 we derive two dynamic lower bounds, each justifying a different limitation of our algorithm given in Theorem 1.6. Our first lower bound, formalized in the following lemma and proved in the full version of the paper [6], concerns our choice to fix a threshold k at the preprocessing phase. If, instead, we aimed for time complexities depending on the current value of $\mathsf{ed}^w(X,Y)$, then, for sufficiently large values of $\mathsf{ed}^w(X,Y)$, we could not achieve update times improving upon the static algorithm.

▶ **Lemma 5.2.** *Consider the following dynamic problem: Given an integer $n \geq 1$ and $\mathcal{O}(1)$-time oracle access to a normalized weight function $w\colon \overline{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, maintain two initially empty strings $X, Y \in \Sigma^{\leq n}$ that throughout the lifetime of the algorithm satisfy $\mathsf{ed}(X,Y) \leq 4$ subject to updates (character edits) and output $\mathsf{ed}^w(X,Y)$ after every update. Suppose that there is an algorithm that solves this problem with $T_P(n)$ preprocessing and $T_U(n, \mathsf{ed}^w(X,Y))$ update time, where T_U is non-decreasing. If $T_P(n) = \mathcal{O}(n^{0.5+1.5\kappa-\delta})$, $T_U(n,2) = \mathcal{O}(n^{1.5\kappa-0.5-\delta})$, and $T_U(n,n^\kappa) = \mathcal{O}(n^{0.5+1.5\kappa-\delta})$ hold for some real parameters $\kappa \in [0.5,1]$ and $\delta > 0$, then the APSP Hypothesis fails.*

Our second dynamic lower bound justifies the specific trade-off between the preprocessing and update time in Theorem 1.6. In simple words, we prove that, with preprocessing time $\widetilde{\mathcal{O}}(nk^\gamma)$ for $\gamma \in [0.5,1)$, the best possible update time is $\mathcal{O}(k^{3-\gamma-o(1)})$. For that, we note that Theorem 1.2 is based on a reduction from the Negative Triangle problem, which is *self-reducible*: solving m instances of bounded weighted edit distance from Theorem 1.2 is hence, in general, m times harder than solving a single instance. Given such m instances $(X_0,Y_0),\ldots,(X_{m-1},Y_{m-1})$, we initialize a dynamic algorithm with a pair of equal strings $\hat{X} = \hat{Y} = \bigodot_{i=0}^{m-1}(X_i \cdot \dagger)$, where $\dagger$ is an auxiliary symbol that is very expensive to edit. For $i \in [0..m)$, in $\mathsf{ed}(X_i,Y_i) = \mathcal{O}(1)$ updates, we can transform the fragment X_i of $\hat{Y}$ into Y_i and retrieve $\mathsf{ed}^w_{\leq k}(\hat{X},\hat{Y}) = \mathsf{ed}^w_{\leq k}(X_i,Y_i)$. By applying and reverting these updates for every $i \in [0..m)$, we can thus find $\mathsf{ed}^w_{\leq k}(X_i,Y_i)$ for all i. If we pick $m \coloneqq n/k^{3-2\gamma-2\delta}$ for an arbitrary small constant $\delta > 0$, then the static lower bound requires $m \cdot (\sqrt{(n/m)k^3})^{1-o(1)} \geq (nk^{\gamma+\delta})^{1-o(1)}$ total time. Our preprocessing time is asymptotically smaller, so, among $\mathcal{O}(m)$ updates, some must take $\Omega((\sqrt{(n/m)k^3})^{1-o(1)}) = \Omega(k^{3-\gamma-\delta-o(1)})$ time. See the full version of the paper [6] for a formal proof.

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *FOCS 2015*, pages 59–78. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.14.
2. Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *STOC 2016*, pages 375–388. ACM, 2016. doi:10.1145/2897518.2897653.
3. Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.
4. Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In *SODA 2000*, pages 819–828. ACM/SIAM, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338645.
5. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

6 Itai Boneh, Egor Gorbachev, and Tomasz Kociumaka. Bounded weighted edit distance: Dynamic algorithms and matching lower bounds, 2025. arXiv:2507.02548v1.

7 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *FOCS 2015*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.

8 Alejandro Cassis, Tomasz Kociumaka, and Philip Wellnitz. Optimal algorithms for bounded weighted edit distance. In *FOCS 2023*, pages 2177–2187. IEEE, 2023. doi:10.1109/FOCS57990.2023.00135.

9 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Shay Mozes. Dynamic string alignment. In *CPM 2020*, volume 161 of *LIPIcs*, pages 9:1–9:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CPM.2020.9.

10 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *FOCS 2020*, pages 978–989. IEEE, 2020. doi:10.1109/FOCS46700.2020.00095.

11 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Pattern matching under weighted edit distance. In *FOCS 2025*, 2025.

12 Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, and Barna Saha. Weighted edit distance computation: Strings, trees, and Dyck. In *STOC 2023*, STOC 2023, pages 377–390, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585178.

13 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006. doi:10.1016/j.jcss.2005.05.007.

14 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal dynamic strings. In *SODA 2018*, pages 1509–1528. SIAM, 2018. doi:10.1137/1.9781611975031.99.

15 Egor Gorbachev and Tomasz Kociumaka. Bounded edit distance: Optimal static and dynamic algorithms for small integer weights. In *57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 2157–2166, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3717823.3718168.

16 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

18 Dominik Kempa and Tomasz Kociumaka. Dynamic suffix array with polylogarithmic queries and updates. In *STOC 2022*, pages 1657–1670. ACM, 2022. doi:10.1145/3519935.3520061.

19 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *SODA*, pages 146–155. SIAM, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070454.

20 Tomasz Kociumaka, Anish Mukherjee, and Barna Saha. Approximating edit distance in the fully dynamic model. In *FOCS 2023*, pages 1628–1638. IEEE, 2023. doi:10.1109/FOCS57990.2023.00098.

21 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *J. Comput. System Sci.*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

22 Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Dokl. Akad. Nauk SSSR*, 163:845–848, 1965. URL: http://mi.mathnet.ru/eng/dan31411.

23 Kurt Mehlhorn, Rajamani Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17:183–198, 1997. doi:10.1007/bf02522825.

24 Eugene W. Myers. An $\mathcal{O}(ND)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986. doi:10.1007/BF01840446.

25 Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48(3):443–453, 1970. doi:10.1016/b978-0-12-131200-8.50031-9.

26 Peter H. Sellers. On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.*, 26(4):787–793, 1974. doi:10.1137/0126070.

27 Peter H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *J. Algorithms*, 1(4):359–373, 1980. doi:10.1016/0196-6774(80)90016-4.

28 Alexander Tiskin. Semi-local string comparison: Algorithmic techniques and applications. *Math. Comput. Sci.*, 1(4):571–603, 2008. doi:10.1007/s11786-007-0033-3.

29 Alexander Tiskin. Fast distance multiplication of unit-Monge matrices. *Algorithmica*, 71(4):859–888, 2015. doi:10.1007/S00453-013-9830-Z.

30 Esko Ukkonen. Finding approximate patterns in strings. *J. Algorithms*, 6(1):132–137, 1985. doi:10.1016/0196-6774(85)90023-9.

31 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

32 Taras K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968. doi:10.1007/BF01074755.

33 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974. doi:10.1145/321796.321811.

Max-Distance Sparsification for Diversification and Clustering

Soh Kumabe
CyberAgent, Tokyo, Japan

Abstract

Let $\mathcal{D}$ be a set family that is the solution domain of some combinatorial problem. The *max-min diversification problem on* $\mathcal{D}$ is the problem to select k sets from $\mathcal{D}$ such that the Hamming distance between any two selected sets is at least d. FPT algorithms parameterized by $k + \ell$, where $\ell = \max_{D \in \mathcal{D}} |D|$, and $k + d$ have been actively studied recently for several specific domains.

This paper provides unified algorithmic frameworks to solve this problem. Specifically, for each parameterization $k + \ell$ and $k + d$, we provide an FPT oracle algorithm for the max-min diversification problem using oracles related to $\mathcal{D}$. We then demonstrate that our frameworks provide the first FPT algorithms on several new domains $\mathcal{D}$, including the domain of t-linear matroid intersection, almost 2-SAT, minimum edge s, t-flows, vertex sets of s, t-mincut, vertex sets of edge bipartization, and Steiner trees. We also demonstrate that our frameworks generalize most of the existing domain-specific tractability results.

Our main technical breakthrough is introducing the notion of *max-distance sparsifier* of $\mathcal{D}$, a domain on which the max-min diversification problem is equivalent to the same problem on the original domain $\mathcal{D}$. The core of our framework is to design FPT oracle algorithms that construct a constant-size max-distance sparsifier of $\mathcal{D}$. Using max-distance sparsifiers, we provide FPT algorithms for the max-min and max-sum diversification problems on $\mathcal{D}$, as well as k-center and k-sum-of-radii clustering problems on $\mathcal{D}$, which are also natural problems in the context of diversification and have their own interests.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Fixed-Parameter Tractability, Diversification, Clustering

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.46

Related Version *Full Version*: https://arxiv.org/abs/2411.02845

1 Introduction

1.1 Background and Motivation

The procedure for approaching real-world problems with optimization algorithms involves formulating the real-world motivations as mathematical problems and then solving them. However, real-world problems are complex, and the idea of a "good" solution cannot always be correctly formulated. The paradigm of *diversification*, introduced by Baste et al. [8] and Baste et al. [7], is a "formulation of the unformulatable problems", which formulates *diversity measures* for a set of multiple solutions, rather than attempting to formulate the "goodness" of a single solution. By computing a set of solutions that maximize this measure, the algorithm provides effective options to evaluators who have the correct criteria for judging the "goodness" of a solution.

Let U be a finite set, $k \in \mathbb{Z}_{\geq 1}$ and $d \in \mathbb{Z}_{\geq 0}$. Let $\mathcal{D} \subseteq 2^U$ be the feasible domain of some combinatorial problem. The following problem frameworks, defined by two types of diversity measures, have been studied extensively.


© Soh Kumabe;
licensed under Creative Commons License CC-BY 4.0



Max-Min Diversification Problem on $\mathcal{D}$: Does there exist a k-tuple $(D_1, \ldots, D_k) \in \mathcal{D}^k$ of sets in $\mathcal{D}$ such that $\min_{1 \leq i < j \leq k} |D_i \triangle D_j| \geq d$?[1]

Max-Sum Diversification Problem on $\mathcal{D}$: Does there exist a k-tuple $(D_1, \ldots, D_k) \in \mathcal{D}^k$ of sets in $\mathcal{D}$ such that $\sum_{1 \leq i < j \leq k} |D_i \triangle D_j| \geq d$?

These problems ensure diversity by aiming to output solutions that are as dissimilar as possible in terms of Hamming distance.

Parameterized algorithms for diversification problems have been actively studied. Particularly, FPT algorithms for the max-min diversification problems parameterized by $k + \ell$, where $\ell = \max_{D \in \mathcal{D}} |D|$ [7, 8, 22, 26, 31], as well as by $k + d$ [17, 21, 22, 23, 24, 29], have been the focus of research. Since assuming $d \leq 2\ell$ does not lose generality in the max-min diversification problem, the latter addresses a more general situation than the former. Since the max-sum diversification problem is empirically more tractable than the max-min diversification problem, for some time hereafter, we will restrict our discussion to the max-min diversification problem.

This research provides general algorithmic frameworks for FPT algorithms solving max-min (and max-sum) diversification problems for both parameterizations $k + \ell$ and $k + d$. Our frameworks are very general and can be applied to all domains [5, 6, 7, 8, 17, 21, 22, 23, 25, 26, 31] for which FPT algorithms parameterized by $k + \ell$ and $k + d$ are currently known for the case that diversity measure is defined using an unweighted Hamming distance. Moreover, our frameworks further provide the first algorithms for several domains where such algorithms were previously unknown.

Our main technical breakthrough is introducing a notion of *max-distance sparsifier* as an intermediate step, which, for the max-min diversification problem, essentially works as a *core-set* [2]. The formal definition is given in Section 1.3. The critical fact is that, when $\mathcal{K}$ is a max-distance sparsifier of $\mathcal{D}$, the max-min diversification problem on $\mathcal{D}$ is equivalent to the same problem on $\mathcal{K}$. Our framework constructs a constant-size max-distance sparsifier $\mathcal{K}$ of $\mathcal{D}$ using the oracles on $\mathcal{D}$, enabling us to solve the max-min diversification problems on $\mathcal{D}$ by brute-force search on $\mathcal{K}$.

The power of max-distance sparsification is not limited to solving diversification problems. Specifically, the following *k-center* [27] and *k-sum-of-radii clustering problems on $\mathcal{D}$* [11] can also be solved via max-distance sparsification.

***k*-Center Clustering Problem on $\mathcal{D}$:** Does there exist a k-tuple of subsets $(D_1, \ldots, D_k) \in \mathcal{D}^k$ such that for all $D \in \mathcal{D}$, there exists an $i \in \{1, \ldots, k\}$ satisfying $|D_i \triangle D| \leq d$?

***k*-Sum-of-Radii Clustering Problem on $\mathcal{D}$:** Does there exist a k-tuple of subsets $(D_1, \ldots, D_k) \in \mathcal{D}^k$ and a k-tuple of non-negative integers $(d_1, \ldots, d_k) \in \mathbb{Z}^k_{\geq 0}$ with $\sum_{i \in \{1, \ldots, k\}} d_i \leq d$ such that for all $D \in \mathcal{D}$, there exists an $i \in \{1, \ldots, k\}$ satisfying $|D_i \triangle D| \leq d_i$?

When $\mathcal{D}$ is an explicitly given set of points, parameterized algorithms for these problems have been extensively studied in the area of clustering [3, 4, 12, 15, 16, 20, 28]. Furthermore, approximation algorithms for the *relational k-means* [13, 19, 30] and *relational k-center* [1] problems are investigated, which are the k-means and k-center clustering problem defined on

[1] We define $Z_1 \triangle Z_2 := (Z_1 \setminus Z_2) \cup (Z_2 \setminus Z_1)$.

a point set represented as a *join* of given relational databases. Their setting is similar to ours as the point set are implicitly given and its size can be exponential. This research adds several new combinatorial domains to the literature on clustering problems on implicitly given domains, and also introduces a parameterized view. These problems are also natural in the context of diversification, since in real situations, the concept of diversity often means that the extracted elements cover the entire space comprehensively rather than being mutually dissimilar. This motivation is formulated by clustering problems, which extract a list of sets in $\mathcal{D}$ such that for each set in $\mathcal{D}$, there is an extracted set near to it.

1.2 Our Results

This paper consists of two parts. In the first part, we design general frameworks for solving diversification and clustering problems. In the second part, we apply our frameworks to several specific domains $\mathcal{D}$. Table 1 shows a list of domains on which our frameworks provide the first or an improved algorithm for the max-min diversification problem.

1.2.1 The Frameworks

We define the following $(-1,1)$-*optimization oracle on* $\mathcal{D}$ and the *exact extension oracle on* $\mathcal{D}$.

> **$(-1,1)$-Optimization Oracle on $\mathcal{D}$:** Let U be a finite set, $\mathcal{D} \subseteq 2^U$, and $w \in \{-1,1\}^U$ be a weight vector. Return a set $D \in \mathcal{D}$ that maximizes $\sum_{e \in D} w_e$.

> **Exact Extension Oracle on $\mathcal{D}$:** Let U be a finite set, $r \in \mathbb{Z}_{\geq 0}$, $\mathcal{D} \subseteq 2^U$, and $C \in \mathcal{D}$. Let $X, Y \subseteq U$ be two disjoint subsets of U. If there exists a set $D \in \mathcal{D}$ such that $|D \triangle C| = r$, $X \subseteq D$, and $Y \cap D = \emptyset$, return one such set. If no such set exists, return $\perp$.

When $C = \emptyset$ and $X = \emptyset$, we specifically call the exact extension oracle on $\mathcal{D}$ the *exact empty extension oracle on* $\mathcal{D}$.

> **Exact Empty Extension Oracle on $\mathcal{D}$:** Let U be a finite set, $r \in \mathbb{Z}_{\geq 0}$, $\mathcal{D} \subseteq 2^U$, and $Y \subseteq U$. If there exists a set $D \in \mathcal{D}$ such that $|D| = r$ and $Y \cap D = \emptyset$, return one such set. If no such set exists, return $\perp$.

Let $\mathcal{P}_{\mathcal{D}}$ be the max-min/max-sum diversification problem or the k-center/k-sum-of-radii clustering problem on $\mathcal{D}$. Our main result is FPT algorithms for solving $\mathcal{P}_{\mathcal{D}}$ using these oracles. We construct frameworks for both types of parameterizations, $k + \ell$ and $k + d$. The result for the parameterization by $k + \ell$ is as follows.

▶ **Theorem 1.** *There exists an oracle algorithm solving $\mathcal{P}_{\mathcal{D}}$ using the exact empty extension oracle on $\mathcal{D}$, where the number of oracle calls and time complexity are both FPT parameterized by $k + \ell$ and for each call of an oracle, r and $|Y|$ are bounded by constants that depend only on $k + \ell$.*

The result for the parameterization by $k + d$ is as follows.

▶ **Theorem 2.** *There exists a randomized oracle algorithm solving $\mathcal{P}_{\mathcal{D}}$ using the $(-1,1)$-optimization oracle on $\mathcal{D}$ and the exact extension oracle on $\mathcal{D}$, where the number of oracle calls and time complexity are both FPT parameterized by $k + d$ and for each call of the exact extension oracle, $r + |X| + |Y|$ are bounded by constants that depend only on $k + d$.*

Table 1 List of new results for the max-min diversification problem obtained by our frameworks. The first column represents the domain $\mathcal{D}$. The second column represents parameterization. For the formal definition of each domain, see the full version.

Domain	Parameter
t-Linear Matroid Intersection	$k+\ell+t$
Almost 2-SAT	$k+\ell$
Independent Set on Certain Graphs	$k+\ell$
Min Edge s,t-Flow	$k+d$
Steiner Tree	$k+d+\lvert T\rvert$
Vertex Set of Min s,t-Cut	$k+d$
Vertex Set of Edge Bipartization	$k+d+s$

1.2.2 Applications of Theorem 1

On most domains $\mathcal{D}$, the exact empty extension oracle can be designed by almost the same way as an algorithm to extract a single solution from $\mathcal{D}$. For example, consider the case where $\mathcal{D}$ is the *ℓ-path domain*, i.e., a domain of sets of edges on paths of length ℓ. In this case, the exact empty extension oracle on $\mathcal{D}$ is equivalent to the problem of finding an ℓ-path in the graph obtained by removing all edges in Y from the input. Combining Theorem 1 with this empirical fact, we can claim that, for most domains $\mathcal{D}$, the diversification and clustering problems parameterized by $k+\ell$ on $\mathcal{D}$ are as easy as determining the non-emptiness of $\mathcal{D}$.

To demonstrate that Theorem 1 yields existing tractability results, we design the oracles for the domains of the vertex cover [7, 8], t-hitting set [7], feedback vertex set [7], and common independent set of two matroids [17, 22]. We also apply our framework on new domains, t-represented linear matroid intersection, almost 2-SAT, and independent set on *subgraph-closed IS-FPT* graph classes. Here, a graph class is *subgraph-closed IS-FPT* if it is closed under taking subgraphs and the problem of finding independent set of size ℓ is FPT parameterized by ℓ. The following theorem summarizes our results, where the precise definitions of each domain are given in the full version.

▶ **Theorem 3.** *Let $\mathcal{D}$ be the domain of vertex covers, t-hitting sets, feedback vertex sets, t-represented linear matroid intersections, almost 2-SATs, or independent sets on subgraph-closed IS-FPT graph classes. Then, max-min and max-sum diversification problems and k-center and k-sum-of-radii clustering problems admit an FPT algorithm, where the parameterization is $k+\ell$ except for the t-hitting set and t-represented linear matroid intersection, which are parameterized by $k+\ell+t$.*

Theorem 1 also generalizes existing frameworks for diversification. Baste et al. [8] provided an algorithmic framework for diversification using a *loss-less kernel* [9, 10], which, roughly speaking, is a kernel that completely preserves the information of the solution space. Since loss-less kernels are known for very limited domains, their framework requires very strong assumptions. Our framework has broader applicability than theirs because it relies on a weaker oracle, as the exact empty extension oracle can be constructed using a loss-less kernel. Hanaka et al. [26] developed a color-coding-based framework for diversification. The oracle they use can be regarded as the exact empty extension oracle with an additional colorfulness constraint. Our framework again has broader applicability than theirs because our oracle can be constructed using theirs. Moreover, our framework also treats clustering problems, which these two do not.

1.2.3 Applications of Theorem 2

FPT algorithms for the max-min diversification problems on $\mathcal{D}$ parameterized by $k+d$ are known for the cases where $\mathcal{D}$ is the family of matroid bases [17, 22], perfect matchings [22], and shortest paths [23]. The result for the perfect matchings is later extended to the matchings of specified size, not necessarily perfect [17]. Additionally, for the cases where $\mathcal{D}$ is the family of interval schedulings [26] and the longest common subsequences of an absolute constant number of strings [31], FPT algorithms parameterized by $k+\ell$ are known. Both of these two can be generalized to the domain of dynamic programming problems, which we define in the full version. Furthermore, the problem of finding a pair of a branching and an in-branching such that the Hamming distance between them is at least d is investigated as the name of *d-distinct branchings problem* [5, 6, 25], which admits FPT algorithm parameterized by d. This problem can naturally be extended to the case that selects k_1 branchings and k_2 in-branching, rather than one each. We give FPT algorithms parameterized by $k+d$ for all those problems, where $k=k_1+k_2$ for the extended version of d-distinct branchings problem. We also give FPT algorithms on domains of minimum edge s,t-flows, Steiner trees, vertex sets of s,t-mincut, and vertex sets of edge bipartization, which are domains where no FPT algorithm for the max-min diversification problem is previously known. Remark that the domain of shortest paths [23] is the special case of the minimum edge s,t-flow domain and the minimum Steiner tree domain. The following theorem summarizes our results, where the precise definitions of each domain are given in the full version.

▶ **Theorem 4.** *Let $\mathcal{D}$ be the domain of matroid bases, branchings, matchings of specified size, minimum edge s,t-flows, minimum Steiner trees, vertex sets of s,t-mincut, vertex sets of edge bipartization, and dynamic programming problems. Then, max-min and max-sum diversification problems and k-center and k-sum-of-radii clustering problems admit an FPT algorithm, where the parameterization is $k+d$ except for the Steiner tree, which is parameterized by $k+d+|T|$ for the terminal set T, and edge bipartization, which is parameterized by $k+\ell+s$, where s is the minimum number of edges to be removed to make the given graph bipartite. Furthermore, the extended version of d-distinct branching problem also admits FPT algorithm parameterized by $k+d$.*

Eiben et al. [17] provided a technique called *determinantal sieving*, which is a general tool to give and speed up parameterized algorithms, including that for diversification problems. Particularly, they provided a framework to solve the diversification problem by using an oracle that, roughly speaking, counts the number of solutions modulo 2. Using their framework, they improved the running times of FPT algorithms for max-min diversification problems on matchings and matroid bases, as well as the d-distinct branchings problem. Although not stated explicitly, their framework seems to yield FPT algorithms parameterized by $k+d$ when $\mathcal{D}$ is the dynamic programming domain, thereby improving the parameterizations in the results of [26] and [31], respectively, as well as the extended version of d-distinct branchings problems. We are not sure whether our framework generalizes theirs, that is, whether our oracle can be constructed using their oracle. However, we strongly believe that our framework has broader applicability because their framework assumes counting oracles, which is often hard even modulo 2. In contrast, our framework uses optimization-type oracles, which are generally more tractable than counting. Indeed, our framework provides an FPT algorithm with the same parameterization for every domain which they explicitly considered. Moreover, we do not think their framework can give an FPT algorithm for the max-min diversification problem on the domains of minimum edge s,t-flows, vertex sets of minimum s,t-cut, and vertex sets of edge bipartization. Furthermore, our framework can be applied not only to diversification problems but also to clustering problems.

1.3 Framework Overview

In this section, we provide an overview of the entire flow of our frameworks. Our frameworks first construct max-distance sparsifiers using the corresponding oracles and then solve diversification and clustering problems using them.

1.3.1 From d-Limited k-Max-Distance Sparsifier to Diversification and Clustering

We begin by defining the max-distance sparsifiers. The key for Theorem 1 is designing the following *k-max-distance sparsifier of $\mathcal{D}$*.

▶ **Definition 5** (k-max-distance sparsifier)**.** *Let $k \in \mathbb{Z}_{\geq 1}$. Let U be a finite set and $\mathcal{D}, \mathcal{F} \subseteq 2^U$. We say that $\mathcal{K} \subseteq \mathcal{D}$ is a* k-max-distance sparsifier of $\mathcal{D}$ with respect to $\mathcal{F}$ *if for any $(F_1, \ldots, F_k) \in \mathcal{F}^k$ and $(z_1, \ldots, z_k) \in \mathbb{Z}^k_{\geq 0}$, the two conditions*

- *There exists $D \in \mathcal{D}$ such that for each $i \in \{1, \ldots, k\}$, $|F_i \triangle D| \geq z_i$.*
- *There exists $K \in \mathcal{K}$ such that for each $i \in \{1, \ldots, k\}$, $|F_i \triangle K| \geq z_i$.*

are equivalent. Unless specifically noted, when we write k-max-distance sparsifier of $\mathcal{D}$*, we mean the case where $\mathcal{D} = \mathcal{F}$.*

Similarly, the key for Theorem 2 is designing the following *d-limited k-max-distance sparsifier of $\mathcal{D}$*.

▶ **Definition 6** (d-limited k-max-distance sparsifier)**.** *Let $k \in \mathbb{Z}_{\geq 1}$ and $d \in \mathbb{Z}_{\geq 0}$. Let U be a finite set and $\mathcal{D}, \mathcal{F} \subseteq 2^U$. We say that $\mathcal{K} \subseteq \mathcal{D}$ is a* d-limited k-max-distance sparsifier of $\mathcal{D}$ with respect to $\mathcal{F}$ *if for any $(F_1, \ldots, F_k) \in \mathcal{F}^k$ and $(z_1, \ldots, z_k) \in \{0, \ldots, d\}^k$, the two conditions*

- *There exists $D \in \mathcal{D}$ such that for each $i \in \{1, \ldots, k\}$, $|F_i \triangle D| \geq z_i$.*
- *There exists $K \in \mathcal{K}$ such that for each $i \in \{1, \ldots, k\}$, $|F_i \triangle K| \geq z_i$.*

are equivalent. Unless specifically noted, when we write d-limited k-max-distance sparsifier of $\mathcal{D}$*, we mean the case where $\mathcal{D} = \mathcal{F}$.*

The difference between the two sparsifiers is that the domain of $(z_1, \ldots, z_k)$ is $\mathbb{Z}^k_{\geq 0}$ in the former case, while it is $\{0, \ldots, d\}^k$ in the latter. By definition, any k-max-distance sparsifier is also a d-limited k-max-distance sparsifier for any $d \in \mathbb{Z}_{\geq 0}$. We can prove that given a d-limited $(k-1)$-max-distance sparsifier of $\mathcal{D}$ with size bounded by a constant that depends only on $k+d$, we can construct FPT algorithms parameterized by $k+d$ for the max-min/max-sum diversification problems on $\mathcal{D}$. Similarly, we can prove that given a $(d+1)$-limited k-max-distance sparsifier of $\mathcal{D}$ with size bounded by a constant that depends only on $k+d$, we can construct FPT algorithms parameterized by $k+d$ for the k-center/k-sum-of-radii clustering problems on $\mathcal{D}$. Therefore, to prove Theorems 1 and 2, it suffices to construct FPT oracle algorithms for designing k-max-distance sparsifiers and d-limited k-max-distance sparsifiers, respectively.

1.3.2 Computing k-Max-Distance Sparsifier

The remaining task towards Theorem 1 is to provide an FPT algorithm parameterized by $k+\ell$ that constructs a k-max-distance sparsifier of $\mathcal{D}$ with size bounded by a constant that depends only on $k+\ell$. The key lemma toward this is that, if $\mathcal{K}$ contains a sufficiently large *sunflower* (see Section 3 for the definition) consisting of sets of the same size, then we can safely remove one of them from $\mathcal{K}$ while preserving the property that $\mathcal{K}$ is a k-max-distance

sparsifier (actually, for the sake of simplifying the framework, we prove a slightly stronger statement). Starting with $\mathcal{K} = \mathcal{D}$ and exhaustively removing such sets leads to a $\mathcal{K}$ that is still a k-max-distance sparsifier of $\mathcal{D}$, and by using the well-known sunflower lemma (Lemma 13), its size is bounded by a constant.

However, this observation is still not sufficient to obtain an FPT algorithm. The reason is that $\mathcal{D}$ generally has exponential size, and removing sets one by one would require an exponential number of steps. Instead, our algorithm starts with $\mathcal{K} = \emptyset$ and exhaustively adds sets of $\mathcal{D}$ to $\mathcal{K}$ until $\mathcal{K}$ becomes a k-max-distance sparsifier. In this way, the number of steps is bounded by a constant. The remaining task is to choose a set to be added at each step. For this task, we design an FPT algorithm using constant number of calls of the exact empty extension oracle.

1.3.3 Computing d-Limited k-Max-Distance Sparsifier

The algorithm in the previous section alone is insufficient to prove Theorem 2 since ℓ is unbounded and the sunflower-lemma-based bound for the number of steps cannot be used. Our algorithm divides $\mathcal{D}$ into at most k clusters, computes a d-limited k-max-distance sparsifier for each cluster, and outputs their union. Let $p > 2d$ be a constant that depends only on $k + d$. We first find $\mathcal{C} \subseteq \mathcal{D}$ satisfying the following properties: (i) $|\mathcal{C}| \leq k$, (ii) for all distinct $C, C' \in \mathcal{C}$, $|C \triangle C'| > 2d$, and (iii) for all $D \in \mathcal{D}$, there exists $C \in \mathcal{C}$ such that $|D \triangle C| \leq p$. If such a family does not exist, a trivial d-limited k-max-distance sparsifier of $\mathcal{D}$ will be found, and we output it and terminate.

We provide an algorithm for computing $\mathcal{C}$. Our algorithm starts with $\mathcal{C} = \emptyset$ and exhaustively adds sets in $\mathcal{D}$ to $\mathcal{C}$ until $\mathcal{C}$ satisfies the above conditions or its size exceeds k. To choose the elements to be added, we randomly sample $w \in \{-1, 1\}^U$ and call a $(-1, 1)$-optimization oracle. We can prove for sufficiently large constant p that if there exists $D \in \mathcal{D}$ such that $|D \triangle C| > p$ for any $C \in \mathcal{C}$, with a constant probability, the $(-1, 1)$-optimization oracle will find a $D \in \mathcal{D}$ such that $|D \triangle C| > 2d$ for any $C \in \mathcal{C}$. Thus, if $\mathcal{C}$ does not meet the conditions, by calling the $(-1, 1)$-optimization oracle a sufficient number of times, we can find a set to add to $\mathcal{C}$ with high probability.

Here, we provide an algorithm for computing a d-limited k-max-distance sparsifier of $\mathcal{D}$ using $\mathcal{C}$. For each cluster $\mathcal{D}_C := \{D \in \mathcal{D} \colon |D \triangle C| \leq p\}$, let $\mathcal{D}^*_C := \{D \triangle C \colon D \in \mathcal{D}_C\}$. The algorithm computes a k-max-distance sparsifier of each $\mathcal{D}^*_C$ and outputs their union. For technical reasons, we actually compute a slightly more general object, but we will not delve into the details here. Since each $\mathcal{D}^*_C$ consists only of sets whose size is at most p, the k-max-distance sparsifier of $\mathcal{D}^*_C$ can be constructed using the algorithm in Section 1.3.2. The exact empty extension oracle on $\mathcal{D}^*_C$ corresponds to the exact extension oracle on $\mathcal{D}$.

Here, we note the difference between our framework and that used by Fomin et al. [22] and Funayama et al. [23] to provide FPT algorithms for the max-min diversification problem on $\mathcal{D}$ when $\mathcal{D}$ is the family of perfect matchings and shortest paths, respectively. Their algorithms also start by dividing $\mathcal{D}$ into clusters. However, their algorithms perform stricter clustering than ours. Specifically, in their clustering, clusters $\mathcal{D}_C$ corresponding to different $C \in \mathcal{C}$ must be well-separated. In contrast, we allow clusters to overlap. This simplifies the clustering step compared to their approach at the cost of a more challenging task afterward. We resolve this more challenging task by introducing and designing the d-limited k-max-distance sparsifier.

1.4 Organization

The rest of this paper is organized as follows. In Section 2, we provide FPT algorithms for solving diversification and clustering problems on $\mathcal{D}$ using a constant-size d-limited k-max-distance sparsifier of $\mathcal{D}$. In Section 3, we prove Theorem 1 by providing an FPT oracle algorithm parameterized by $k + \ell$ that computes a constant-size k-max-distance sparsifier of $\mathcal{D}$. In Section 4, we prove Theorem 2 by providing an FPT oracle algorithm parameterized by $k + d$ that computes a constant-size d-limited k-max-distance sparsifier of $\mathcal{D}$. The discussion in Section 4 internally uses the results from Section 3. In the full version, we apply the results of Theorems 1 and 2 to several domains $\mathcal{D}$ to obtain FPT algorithms for diversification and clustering problems. The full version also contains further related work and the proofs of the lemmas marked with an asterisk.

2 From Sparsifier to Diversification and Clustering

In this section, we provide FPT algorithms for diversification and clustering problems using a d-limited k-max-distance sparsifier of constant size.

2.1 Diversification

Let U be a finite set, $d \in \mathbb{Z}_{\geq 0}$, $k \in \mathbb{Z}_{\geq 1}$, and $\mathcal{D} \subseteq 2^U$. For diversification problems, we have the following.

▶ **Lemma 7.** *Let $\mathcal{K} \subseteq \mathcal{D}$ be a d-limited $(k-1)$-max-distance sparsifier of $\mathcal{D}$ and $(D_1, \ldots, D_k) \in \mathcal{D}^k$. Then, there is a k-tuple $(K_1, \ldots, K_k) \in \mathcal{K}^k$ such that $\min(d, |D_i \triangle D_j|) \leq \min(d, |K_i \triangle K_j|)$ holds for all $1 \leq i < j \leq k$. Particularly, if there exists a solution to the max-min/max-sum diversification problem on $\mathcal{D}$, then there exists a solution consisting only of sets in $\mathcal{K}$.*

Proof. Assume $(D_1, \ldots, D_k) \in \mathcal{D}^k \backslash \mathcal{K}^k$ and let $i \in \{1, \ldots, k\}$ be an index such that $D_i \notin \mathcal{K}$. It is sufficient to prove that there is a set $K_i \in \mathcal{K}$ such that $\min(d, |D_i \triangle D_j|) \leq \min(d, |K_i \triangle D_j|)$ holds for all $j \in \{1, \ldots, k\} \setminus \{i\}$. For $j \in \{1, \ldots, k\} \setminus \{i\}$, let $z_j := \min(d, |D_i \triangle D_j|)$. Since $\mathcal{K}$ is a d-limited $(k-1)$-max-distance sparsifier of $\mathcal{D}$, there exists $K_i \in \mathcal{K}$ such that $\min(d, |K_i \triangle D_j|) \geq \min(d, z_j) = \min(d, |D_i \triangle D_j|)$ holds for all $j \in \{1, \ldots, k\} \setminus \{i\}$. ◀

Considering an algorithm that exhaustively searches for a subfamily of $\mathcal{K}$ of size k, we can state the following.

▶ **Lemma 8.** *Assume there exists an FPT algorithm parameterized by $k + d$ to compute a d-limited $(k-1)$-max-distance sparsifier of $\mathcal{D}$ with size bounded by a constant that depends only on $k + d$. Then, there exists an FPT algorithm parameterized by $k + d$ for the max-min/max-sum diversification problem on $\mathcal{D}$.*

We note that, under the slightly stronger assumption, the discussion in this section can directly be extended to the case where the sets $D_1, \ldots, D_k$ are taken from different domains. Specifically, let $\mathcal{D}_1, \ldots, \mathcal{D}_k \subseteq 2^U$ and assume $(k-1)$-max-distance sparsifiers $\mathcal{K}_1, \ldots, \mathcal{K}_k$ of these domains with respect to 2^U are computed. Then, we can determine whether there exists a k-tuple $(D_1, \ldots, D_k) \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_k$ such that $\min_{1 \leq i < j \leq k} |D_i \triangle D_j| \geq d$ (or $\sum_{1 \leq i < j \leq k} |D_i \triangle D_j| \geq d$) by exhaustive search on $\mathcal{K}_1 \times \cdots \times \mathcal{K}_k$.

2.2 Clustering

Let U be a finite set, $d \in \mathbb{Z}_{\geq 0}$, $k \in \mathbb{Z}_{\geq 1}$, and $\mathcal{D} \subseteq 2^U$. Here, we provide an FPT algorithm for the k-center and k-sum-of-radii clustering problems using a $(d+1)$-limited k-max-distance sparsifier $\mathcal{K}$ of $\mathcal{D}$. For $Z \subseteq U$ and $r \in \mathbb{Z}_{\geq 0}$, the *ball* of radius r centered at Z is defined as $\mathcal{B}(Z, r) := \{Z' \subseteq U : |Z \triangle Z'| \leq r\}$. The algorithm first guesses a partition of $\mathcal{K}$ into k clusters $\mathcal{K}_1, \ldots, \mathcal{K}_k$. Since $|\mathcal{K}|$ is constant, the cost of this guess is constant. Then, for each $i \in \{1, \ldots, k\}$, the algorithm computes the minimum radius r_i such that there is a set $D_i \in \mathcal{D}$ satisfying $\mathcal{K}_i \subseteq \mathcal{B}(D_i, r_i)$. If $r_i > d$, the algorithm asserts it instead of computing the specific value of r_i. The k-center clustering problem and k-sum-of-radii clustering problem on $\mathcal{D}$ are solved by checking whether the maximum and sum, respectively, of the r_is is at most d. We show the correctness of this algorithm by proving the following.

▶ **Lemma 9.** *Let $(D_1, \ldots, D_k) \in \mathcal{D}^k$ and $(r_1, \ldots, r_k) \in \{0, \ldots, d\}^k$. Assume $\mathcal{K}_i \subseteq \mathcal{B}(D_i, r_i)$ holds for all $i \in \{1, \ldots, k\}$. Then, for all $D \in \mathcal{D}$, there is an index $i \in \{1, \ldots, k\}$ such that $D \in \mathcal{B}(D_i, r_i)$.*

Proof. Assume the contrary. Then, there is a set $D \in \mathcal{D}$ such that for all $i \in \{1, \ldots, k\}$, $|D_i \triangle D| \geq r_i + 1$. Since $\mathcal{K}$ is a $(d+1)$-limited k-max-distance sparsifier of $\mathcal{D}$, there is a set $K \in \mathcal{K}$ such that for all $i \in \{1, \ldots, k\}$, $|D_i \triangle K| \geq r_i + 1$. Hence, $K \notin \mathcal{K}_i$ for all $i \in \{1, \ldots, k\}$, contradicting the fact that $(\mathcal{K}_1, \ldots, \mathcal{K}_k)$ is a partition of $\mathcal{K}$. ◀

We now provide an algorithm to decide whether there exists $D \in \mathcal{D}$ with $\mathcal{K}_i \subseteq \mathcal{B}(D, r_i)$ for each $i \in \{1, \ldots, k\}$ and $r_i \in \{0, \ldots, d\}$. If the domain $\mathcal{D}$ is 2^U, this problem is equivalent to the *closest string problem* on binary strings, for which a textbook FPT algorithm parameterized by $d + |\mathcal{K}_i|$ is known [14]. Our algorithm is a modified version of this. An element $e \in U$ is *bad* if there exist both $K \in \mathcal{K}_i$ with $e \in K$ and $K \in \mathcal{K}_i$ with $e \notin K$. The following lemma is fundamental.

▶ **Lemma 10** ([14])**.** *If there are more than $d|\mathcal{K}_i|$ bad elements, no $D \in \mathcal{D}$ satisfies $\mathcal{K}_i \subseteq \mathcal{B}(D, d)$.*

Let B be the set of bad elements, and assume $|B| \leq d|\mathcal{K}_i|$. The algorithm first guesses $B' \subseteq B$. The cost of this guess is $2^{d|\mathcal{K}_i|}$. Then, it determines whether there exists $D \in \mathcal{D}$ such that $D \cap B = B'$ and $\mathcal{K}_i \subseteq \mathcal{B}(D, r_i)$. Let $K^* = \operatorname{argmax}_{K \in \mathcal{K}_i} |(K \cap B) \triangle B'|$. Then, we can claim the following.

▶ **Lemma 11.** *For $D \in \mathcal{D}$ such that $D \cap B = B'$, $\max_{K \in \mathcal{K}_i} |K \triangle D| = |K^* \triangle D|$.*

Proof. Let $K \in \mathcal{K}_i$. Then, $|K \triangle D| = |(K \cap B) \triangle (D \cap B)| + |(K \setminus B) \triangle (D \setminus B)|$. From the definition of B, the value of $|(K \setminus B) \triangle (D \setminus B)|$ is equal among all $K \in \mathcal{K}_i$. Thus, the maximum value of $|K \triangle D|$ for $K \in \mathcal{K}_i$ is achieved by the set K that maximizes $|(K \cap B) \triangle (D \cap B)| = |(K \cap B) \triangle B'|$. ◀

Now, it is sufficient to solve the problem of determining whether there exists $D \in \mathcal{D}$ such that $D \cap B = B'$ and $|K^* \triangle D| \leq r_i$. This corresponds to the exact extension oracle on $\mathcal{D}$ with $r = r_i$, $X = B'$, $Y = B \setminus B'$, and $C = K^*$. Therefore, we can claim the following:

▶ **Lemma 12.** *Assume there exists an FPT algorithm parameterized by $k + d$ that computes a $(d+1)$-limited k-max-distance sparsifier of $\mathcal{D}$ with size bounded by a constant that depends only on $k + d$, and the exact extension oracle on $\mathcal{D}$ whose time complexity is FPT parameterized by $r + |X| + |Y|$. Then, there exists an FPT algorithm parameterized by $k + d$ for the k-center/k-sum-of-radii clustering problem on $\mathcal{D}$.*

Algorithm 1 k-max-distance sparsification of $\mathcal{D}$.

```
Procedure KSparsify(k, r)
    Input: k ∈ Z_{≥1}, r ∈ Z_{≥0}
    Let K := ∅;
    while true do
        Let R := ⋃_{K∈K} K, f := false;
        for ℓ' ∈ {0, ..., l} do
            Let 𝔖 be the family of all sunflowers S ⊆ K such that |S| = kr + 1 and
              each S ∈ S satisfies |S| = ℓ';
            for Y ⊆ R that intersects with all K ∈ K with |K| = ℓ' and the cores of all
              sunflowers of 𝔖 do
                Let D = ExactEmptyExtension(ℓ', Y);
                if D ≠ ⊥ and f = false then
                    Add D to K and f := true;
        if f = false then
            break;
    return K;
```

3 Framework for k-Max-Distance Sparification

In this section, we complete the proof of Theorem 1 by providing an FPT algorithm that uses the exact empty extension oracle on $\mathcal{D}$ to obtain a k-max-distance sparsifier of $\mathcal{D}$. For further use, we show a slightly more extended result. Let $r \in \mathbb{Z}_{\geq 0}$. We construct a k-max-distance sparsifier of $\mathcal{D}$ with respect to $\mathcal{B}(\emptyset, r)$ for $r \geq \ell$. Since $\mathcal{D} \subseteq \mathcal{B}(\emptyset, \ell) \subseteq \mathcal{B}(\emptyset, r)$ for $r \geq \ell$, this is also a k-max-distance sparsifier of $\mathcal{D}$ (with respect to $\mathcal{D}$). A set family $\mathcal{S} := \{S_1, \ldots, S_t\}$ is called a *sunflower* if there exists a set called *core* C such that for any $1 \leq i < j \leq t$, $S_i \cap S_j = C$. The following is well-known.

▶ **Lemma 13** (Sunflower Lemma [14, 18]). *Let U be a finite set, $\ell, t \in \mathbb{Z}_{\geq 0}$, and $\mathcal{K} \subseteq 2^U$ be a family consisting only of sets of size at most ℓ. If $|\mathcal{K}| > \ell!(t-1)^\ell$, then $\mathcal{K}$ contains a sunflower of size t.*

For $t \in \mathbb{Z}_{\geq 0}$, $\mathcal{T} \subseteq 2^U$, and $Z \in 2^U \setminus \mathcal{T}$, a sunflower $\mathcal{S} \subseteq \mathcal{T}$ is a *(Z,t)-sunflower of $\mathcal{T}$* if it satisfies the following three conditions.

- $|\mathcal{S}| = t$,
- For each $S \in \mathcal{S}$, $|S| = |Z|$, and
- The core of $\mathcal{S}$ is a subset of Z.

The following lemma is the core of our framework.

▶ **Lemma 14 (*).** *Let U be a finite set, $\mathcal{D} \subseteq 2^U$, and $\mathcal{K} \subseteq \mathcal{D}$ be a k-max-distance sparsifier of $\mathcal{D}$ with respect to $\mathcal{B}(\emptyset, r)$. Let $Z \in \mathcal{K}$ and assume there is a $(Z, kr+1)$-sunflower $\mathcal{S}$ of $\mathcal{K} \setminus \{Z\}$. Then, $\mathcal{K} \setminus \{Z\}$ is also a k-max-distance sparsifier of $\mathcal{D}$ with respect to $\mathcal{B}(\emptyset, r)$.*

Our algorithm is given in Algorithm 1, where ExactEmptyExtension(ℓ', Y) represents the exact empty extension oracle on $\mathcal{D}$ with arguments ℓ' and Y. The algorithm starts with $\mathcal{K} := \emptyset$ and repeatedly adds $Z \in \mathcal{D} \setminus \mathcal{K}$ such that there is no $(Z, kr+1)$-sunflower of $\mathcal{K}$ to $\mathcal{K}$. The following lemma shows this algorithm stops after a constant number of iterations.

▶ **Lemma 15** (*). *The number of iterations of the loop starting from line 3 in Algorithm 1, as well as the size of the output family, is at most* $(\ell+1)!(kr+1)^\ell$.

In particular, at each step of the algorithm, since $|R| \leq |\mathcal{K}|l \leq (\ell+1)!(kr+1)^\ell \ell$, the size of Y chosen in line 7 is bounded by a constant. The time complexity is bounded as follows.

▶ **Lemma 16** (*). *Algorithm 1 makes at most* $2^{2^{O(\ell \log(klr))}}$ *calls of* ExactEmptyExtension$(\cdot)$ *and has a time complexity of* $2^{2^{O(\ell \log(klr))}}$.

The correctness of the algorithm is shown as follows.

▶ **Lemma 17** (*). *Algorithm 1 outputs a k-max-distance sparsifier of $\mathcal{D}$ with respect to $\mathcal{B}(\emptyset, r)$.*

4 Framework for d-Limited k-Max-Distance Sparsification

4.1 Overall Flow

In this section, we complete the proof of Theorem 2 by providing FPT algorithm that uses the $(-1,1)$-optimization oracle and the exact extension oracle on $\mathcal{D}$ to obtain a d-limited k-max-distance sparsifier of $\mathcal{D}$. Actually, for further applications, we construct the slightly more general object of d-limited k-max-distance sparsifier of $\mathcal{D}$ with respect to 2^U, not with respect to $\mathcal{D}$ itself. Our framework consists of two steps. Let $p \in \mathbb{Z}_{\geq 0}$ be an integer with $2d < p$. The first step achieves one of the following.

- Find a set $\mathcal{C} \subseteq \mathcal{D}$ with size at most k such that $\mathcal{D} \subseteq \bigcup_{C\in\mathcal{C}} \mathcal{B}(C,p)$.
- Find a set $\mathcal{C} \subseteq \mathcal{D}$ of size $k+1$ such that $|C \triangle C'| > 2d$ holds for any distinct $C, C' \in \mathcal{C}$.

We do this by using the following *approximate far set oracle*, which will be designed in Section 4.2.

> **Approximate Far Set Oracle:** Let U be a finite set, $d \in \mathbb{Z}_{\geq 0}$, $\mathcal{D} \subseteq 2^U$, and $\mathcal{C} \subseteq \mathcal{D}$. The approximate far set oracle returns one of the following.
> - A set of $\mathcal{D}$ that does not belong to $\bigcup_{C\in\mathcal{C}} \mathcal{B}(C, 2d)$.
> - $\perp$. This option can be chosen only when $\mathcal{D} \subseteq \bigcup_{C\in\mathcal{C}} \mathcal{B}(C,p)$.

Starting with $\mathcal{C} := \emptyset$, we repeat the following steps. If the approximate far set oracle returns $\perp$, terminate the loop. Otherwise, add the element found by the oracle to $\mathcal{C}$. If the oracle returns $\perp$ within k iterations, the first condition is achieved. If not, the set $\mathcal{C}$ after $k+1$ iterations satisfies the second condition. In the latter case, the following lemma shows that $\mathcal{C}$ is a d-limited k-max-distance sparsifier.

▶ **Lemma 18** (*). *Let $r \in \mathbb{Z}_{\geq 0}$. Let $\mathcal{C}$ be a subset of $\mathcal{D}$ of size $k+1$ such that for any distinct $C, C' \in \mathcal{C}$, $|C \triangle C'| \geq 2d$. Then, $\mathcal{C}$ is a d-limited k-max-distance sparsifier of $\mathcal{D}$ with respect to 2^U.*

Now, we assume the first condition. Let $\mathcal{C}$ be a subset of $\mathcal{D}$ of size at most k such that $\mathcal{D} \subseteq \bigcup_{C\in\mathcal{C}} \mathcal{B}(C,p)$. The second step involves constructing a d-limited k-max-distance sparsifier of $\mathcal{D}_C := \mathcal{D} \cap \mathcal{B}(C,p)$ with respect to $\mathcal{B}(C, p+d)$ for each $C \in \mathcal{C}$. We prove that the union of all such d-limited k-max-distance sparsifiers obtained in this manner is a d-limited k-max-distance sparsifier of $\mathcal{D}$ with respect to 2^U.

▶ **Lemma 19** (*). *Assume $\mathcal{D} = \bigcup_{C\in\mathcal{C}} \mathcal{D}_C$. For each $C \in \mathcal{C}$, let $\mathcal{K}_C \subseteq \mathcal{D}_C$ be a d-limited k-max-distance sparsifier of $\mathcal{D}_C$ with respect to $\mathcal{B}(C, p+d)$. Then, $\mathcal{K} := \bigcup_{C\in\mathcal{C}} \mathcal{K}_C$ is a d-limited k-max-distance sparsifier of $\mathcal{D}$ with respect to 2^U.*

Next, we reduce the computation of d-limited k-max-distance sparsifiers to the computation of k-max-distance sparsifiers of families consisting of constant-size sets, which was discussed in Section 3. For $C \in \mathcal{C}$, let $\mathcal{D}^*_C := \{D \triangle C \mid D \in \mathcal{D}_C\}$. By the definition of $\mathcal{D}_C$, we have $\mathcal{D}^*_C \subseteq \mathcal{B}(\emptyset, p)$. The following holds.

▶ **Lemma 20 (*).** *Let $C \in \mathcal{C}$. A subset $\mathcal{K}_C \subseteq \mathcal{D}_C$ is a d-limited k-max-distance sparsifier of $\mathcal{D}_C$ with respect to $\mathcal{B}(C, p+d)$ if and only if $\mathcal{K}^*_C := \{K \triangle C \mid K \in \mathcal{K}_C\}$ is a d-limited k-max-distance sparsifier of $\mathcal{D}^*_C$ with respect to $\mathcal{B}(\emptyset, p+d)$.*

If $\mathcal{K}^*_C$ is a k-max-distance sparsifier of $\mathcal{D}^*_C$ with respect to $\mathcal{B}(\emptyset, p+d)$, then it is also a d-limited k-max-distance sparsifier of $\mathcal{D}^*_C$ with respect to $\mathcal{B}(\emptyset, p+d)$ for any $d \in \mathbb{Z}_{\geq 0}$. Therefore, a d-limited k-max-distance sparsifier $\mathcal{K}$ of $\mathcal{D}$ with respect to $\mathcal{B}(C, p+d)$ can be computed as

$$\mathcal{K} := \bigcup_{C \in \mathcal{C}} \{K^* \triangle C \colon K^* \in \mathcal{K}^*_C\}.$$

From the discussion in Section 3, $\mathcal{K}^*_C$ can be obtained by calling the exact empty extension oracle on $\mathcal{D}^*_C$ a constant number of times that depends on k, $\ell = p$, and $r = p + d$. The exact empty extension oracle for $\mathcal{D}^*_C$ is equivalent to the exact extension oracle on $\mathcal{D}_C$ when the inputs C, X, Y are taken to be $C, Y \cap C, Y \setminus C$, respectively. Therefore, $\mathcal{K}^*_C$ can be obtained by calling the exact extension oracle on $\mathcal{D}_C$ a constant number of times that depends only on k and p.

4.2 Designing the Approximate Far Set Oracle

Here, we design a randomized algorithm parameterized by $|\mathcal{C}|$ and d for the approximate far set oracle. Our algorithm repeats the following sufficient number of times: It selects a weight vector $w \in \{-1, 1\}^U$ uniformly at random and finds a set $D \in \mathcal{D}$ that maximizes $w(D) := \sum_{e \in D} w_e$. If the found D does not belong to $\bigcup_{C \in \mathcal{C}} \mathcal{B}(C, 2d)$, it outputs this D and terminates. If no such D is found after a sufficient number of iterations, it returns $\perp$. We now prove the correctness of the algorithm. We can claim the following.

▶ **Lemma 21 (*).** *Assume $\max_{D \in \mathcal{D}} w(D) > \max_{C \in \mathcal{C}} w(C) + 2d$. Then, the D that attains the maximum on the left-hand side does not belong to $\bigcup_{C \in \mathcal{C}} \mathcal{B}(C, 2d)$.*

The following lemma is the core of the analysis.

▶ **Lemma 22 (*).** *Assume $p \geq (4d+2)^2 \cdot 2^{k-1}$ and $|\mathcal{C}| \leq k$. Let $D \in \mathcal{D}$ and assume $D \notin \bigcup_{C \in \mathcal{C}} \mathcal{B}(C, p)$. Then,*

$$\Pr\left[w(D) > \max_{C \in \mathcal{C}} w(C) + 2d\right] \geq 2^{-2^{O(k)}}.$$

By repeating the sampling of w a sufficient number of times, we can state the following.

▶ **Lemma 23.** *Let $\epsilon > 0$, $\mathcal{D}, \mathcal{C} \subseteq 2^U$, $k \in \mathbb{Z}_{\geq 1}$, and $d \in \mathbb{Z}_{\geq 0}$. Assume $|\mathcal{C}| \leq k$. Then, there exists a randomized algorithm that runs in time $2^{2^{O(k)}} \log \epsilon^{-1}$ and satisfies the following.*

- *If there exists $D \in \mathcal{D}$ such that $D \notin \bigcup_{C \in \mathcal{C}} \mathcal{B}(C, (4d+2)^2 \cdot 2^k)$, the algorithm returns a set $D' \in \mathcal{D}$ satisfying $D' \notin \bigcup_{C \in \mathcal{C}} \mathcal{B}(C, 2d)$ with probability at least $1 - \epsilon$.*
- *If not, the algorithm returns either $\perp$ or a set $D' \in \mathcal{D}$ satisfying $D' \notin \bigcup_{C \in \mathcal{C}} \mathcal{B}(C, 2d)$.*

Combining Lemma 23 with the results from Sections 3 and 4.1, we have the following.

▶ **Lemma 24 (*).** *Let $\epsilon > 0$, $\mathcal{D} \subseteq 2^U$, $k \in \mathbb{Z}_{\geq 1}$, and $d \in \mathbb{Z}_{\geq 0}$. Then, there exists a randomized algorithm that, with probability $1-\epsilon$, computes a d-limited k-max-distance sparsifier of $\mathcal{D}$ with respect to 2^U with size at most $2^{2^{O(k+\log d)}}$ in time $\left(2^{2^{2^{O(k+\log d)}}} + 2^{2^{O(k)}} \log \epsilon^{-1}\right) poly(|U|)$. It uses at most $2^{2^{O(k)}} \log \epsilon^{-1}$ calls to the $(-1,1)$-optimization oracle on $\mathcal{D}$ and at most $2^{2^{2^{O(k+\log d)}}}$ calls to the exact extension oracle on $\mathcal{D}$ such that $r \leq (4d+2)^2 2^{k-1}$ and $|X|, |Y| \leq 2^{2^{O(k+\log d)}}$.*

References

1. Pankaj Agarwal, Aryan Esmailpour, Xiao Hu, Stavros Sintos, and Jun Yang. Computing a well-representative summary of conjunctive query results. *Proceedings of the ACM on Management of Data*, 2(5):1–27, 2024. doi:10.1145/3695835.
2. Pankaj Agarwal, Sariel Har-Peled, and Kasturi Varadarajan. Approximating extent measures of points. *Journal of the ACM (JACM)*, 51(4):606–635, 2004. doi:10.1145/1008731.1008736.
3. Amihood Amir, Jessica Ficler, Liam Roditty, and Oren Sar Shalom. On the efficiency of the hamming c-centerstring problems. In *Combinatorial Pattern Matching (CPM)*, pages 1–10. Springer, 2014. doi:10.1007/978-3-319-07566-2_1.
4. Sayan Bandyapadhyay, William Lochet, and Saket Saurabh. FPT constant-approximations for capacitated clustering to minimize the sum of cluster radii. In *International Symposium on Computational Geometry (SoCG)*, volume 258, pages 12:1–12:14, 2023. doi:10.4230/LIPICS.SOCG.2023.12.
5. Jørgen Bang-Jensen, Kristine Vitting Klinkby, and Saket Saurabh. k-distinct branchings admits a polynomial kernel. In *European Symposium on Algorithms (ESA)*, pages 11:1–11:15, 2021. doi:10.4230/LIPICS.ESA.2021.11.
6. Jørgen Bang-Jensen, Saket Saurabh, and Sven Simonsen. Parameterized algorithms for non-separating trees and branchings in digraphs. *Algorithmica*, 76:279–296, 2016. doi:10.1007/S00453-015-0037-3.
7. Julien Baste, Michael Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Artificial Intelligence*, 303:103644, 2022. doi:10.1016/J.ARTINT.2021.103644.
8. Julien Baste, Lars Jaffke, Tomáš Masařík, Geevarghese Philip, and Günter Rote. FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12(12):254, 2019. doi:10.3390/A12120254.
9. Clément Carbonnel and Emmanuel Hebrard. Propagation via kernelization: The vertex cover constraint. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 147–156. Springer, 2016. doi:10.1007/978-3-319-44953-1_10.
10. Clément Carbonnel and Emmanuel Hebrard. On the kernelization of global constraints. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 578–584, 2017. doi:10.24963/IJCAI.2017/81.
11. Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. In *ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 2001. doi:10.1145/380752.380753.
12. Xianrun Chen, Dachuan Xu, Yicheng Xu, and Yong Zhang. Parameterized approximation algorithms for sum of radii clustering and variants. In *AAAI Conference on Artificial Intelligence*, volume 38, pages 20666–20673, 2024. doi:10.1609/AAAI.V38I18.30053.
13. Ryan Curtin, Benjamin Moseley, Hung Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. RK-means: Fast clustering for relational data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2742–2752. PMLR, 2020. URL: http://proceedings.mlr.press/v108/curtin20a.html.

14 Marek Cygan, Fedor Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

15 Erik Demaine, Fedor Fomin, MohammadTaghi Hajiaghayi, and Dimitrios Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms (TALG)*, 1(1):33–47, 2005. doi:10.1145/1077464.1077468.

16 Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. On the parameterized complexity of clustering problems for incomplete data. *Journal of Computer and System Sciences*, 134:1–19, 2023. doi:10.1016/J.JCSS.2022.12.001.

17 Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–423. SIAM, 2024. doi:10.1137/1.9781611977912.16.

18 Paul Erdös and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.

19 Aryan Esmailpour and Stavros Sintos. Improved approximation algorithms for relational clustering. *Proceedings of the ACM on Management of Data*, 2(5):213:1–213:27, 2024. doi:10.1145/3695831.

20 Andreas Emil Feldmann and Dániel Marx. The parameterized hardness of the k-center problem in transportation networks. *Algorithmica*, 82:1989–2005, 2020. doi:10.1007/S00453-020-00683-W.

21 Fedor Fomin, Petr Golovach, Lars Jaffke, Geevarghese Philip, and Danil Sagunov. Diverse pairs of matchings. *Algorithmica*, 86(6):2026–2040, 2024. doi:10.1007/S00453-024-01214-7.

22 Fedor Fomin, Petr Golovach, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. Diverse collections in matroids and graphs. *Mathematical Programming*, 204(1):415–447, 2024. doi:10.1007/S10107-023-01959-Z.

23 Ryo Funayama, Yasuaki Kobayashi, and Takeaki Uno. Parameterized complexity of finding dissimilar shortest paths. *arXiv preprint arXiv:2402.14376*, 2024. doi:10.48550/arXiv.2402.14376.

24 Tatsuya Gima, Yuni Iwamasa, Yasuaki Kobayashi, Kazuhiro Kurita, Yota Otachi, and Rin Saito. Computing diverse pair of solutions for tractable sat. *arXiv preprint arXiv:2412.04016*, 2024. doi:10.48550/arXiv.2412.04016.

25 Gregory Gutin, Felix Reidl, and Magnus Wahlström. k-distinct in-and out-branchings in digraphs. *Journal of Computer and System Sciences*, 95:86–97, 2018. doi:10.1016/J.JCSS.2018.01.003.

26 Tesshu Hanaka, Yasuaki Kobayashi, Kazuhiro Kurita, and Yota Otachi. Finding diverse trees, paths, and more. In *AAAI Conference on Artificial Intelligence*, volume 35, pages 3778–3786, 2021. doi:10.1609/AAAI.V35I5.16495.

27 Wen-Lian Hsu and George Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.

28 Tanmay Inamdar and Kasturi Varadarajan. Capacitated sum-of-radii clustering: An FPT approximation. In *European Symposium on Algorithms (ESA)*, 2020.

29 Neeldhara Misra, Harshil Mittal, and Ashutosh Rai. On the parameterized complexity of diverse SAT. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 322, pages 50:1–50:18, 2024. doi:10.4230/LIPICS.ISAAC.2024.50.

30 Benjamin Moseley, Kirk Pruhs, Alireza Samadian, and Yuyan Wang. Relational algorithms for k-means clustering. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 198, pages 97:1–97:21, 2021. doi:10.4230/LIPICS.ICALP.2021.97.

31 Yuto Shida, Giulia Punzi, Yasuaki Kobayashi, Takeaki Uno, and Hiroki Arimura. Finding diverse strings and longest common subsequences in a graph. In *Symposium on Combinatorial Pattern Matching (CPM)*, volume 296, pages 27:1–27:19, 2024. doi:10.4230/LIPICS.CPM.2024.27.

Fast and Lightweight Distributed Suffix Array Construction

Manuel Haag
Karlsruhe Institute of Technology, Germany

Florian Kurpicz
Karlsruhe Institute of Technology, Germany

Peter Sanders
Karlsruhe Institute of Technology, Germany

Matthias Schimek
Karlsruhe Institute of Technology, Germany

Abstract

The suffix array contains the lexicographical order of all suffixes of a text. It is one of the most well-studied text indices with applications in bioinformatics, compression, and pattern matching. The main bottleneck of distributed-memory suffix array construction algorithms is their memory requirements. Even careful implementations require 30×–60× the input size as working memory. We present a scalable and lightweight distributed-memory adaptation of the difference cover (DCX) suffix array construction algorithm. Our approach relies on novel bucketing and random chunk redistribution techniques which reduce our memory requirement to 20×–26× the input size for medium-sized inputs and to 14×–15× for large-sized inputs. Regarding running time, we achieve speedups of up to 5× over current state-of-the-art distributed suffix array construction algorithms.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed Computing, Suffix Array Construction

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.47

Related Version *Previous Version*: `https://arxiv.org/abs/2412.10160`

Supplementary Material
Software: `https://github.com/mschimek/distributed_suffix_sorting/`
archived at `swh:1:dir:d268977edf8b2c9a7427b598a7f3e87493fb2566`
Software: `https://github.com/mschimek/distributed_suffix_sorting_evaluation`
archived at `swh:1:dir:991feb9b718b7cc1b22f14047efbd281a9fc0e3a`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500). The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (`www.gauss-centre.eu`) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (`www.lrz.de`).
The authors gratefully acknowledge the computing time provided on the high-performance computer HoreKa by the National High-Performance Computing Center at KIT (NHR@KIT). This center is jointly supported by the Federal Ministry of Education and Research and the Ministry of Science, Research and the Arts of Baden-Württemberg, as part of the National High-Performance Computing (NHR) joint funding program (`https://www.nhr-verein.de/en/our-partners`). HoreKa is partly funded by the German Research Foundation (DFG).

© Manuel Haag, Florian Kurpicz, Peter Sanders, and Matthias Schimek;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The suffix array [30, 22] is one of the most studied text indices. Given a text T of length n, its suffix array SA contains a permutation of the integers $1, \ldots, n$ such that the corresponding suffixes appear in lexicographical order. More concretely, SA[i] is the starting position of the i-th smallest suffix of T, or equivalently, we have $T[\mathrm{SA}[i]..n] \leq T[\mathrm{SA}[j]..n]$ for any $1 \leq i \leq j \leq n$. To compute the suffix array, we have to (implicitly) sort all suffixes of the text. Therefore, the task of constructing the suffix array is sometimes referred to as suffix sorting. Suffix arrays can be constructed in linear time requiring only constant working space in addition to the space for the suffix array [23, 29].

Suffix arrays have numerous applications in pattern matching and text compression [34]. They are a very powerful full-text index and are used as a space-efficient replacement [1] of the suffix tree, which is considered to be one of the most powerful full-text indices. Furthermore, suffix arrays can be used to compute the Burrows-Wheeler transform [12], which is the backbone of many compressed full-text indices [17, 21].

One problem when considering texts as input is that the amount of textual data to be processed is ever-increasing with no sign of slowing down. For example, the English Wikipedia contains around 60 million pages and grows by around 2.5 million pages each year.[1] A snapshot of all public source code repositories on GitHub requires more than 21 TB to store.[2] Furthermore, the capability to sequence genomic data is increasing exponentially, due to technical advances [37]. All these examples show the importance of scalable algorithms for the analysis of textual information many of which use the suffix array as a building block.

In this paper, we consider distributed-memory suffix sorting. Here, we can utilize many processing elements (PEs) that are connected via a network, e.g., high-performance clusters or cloud computing. In this setting, the main obstacle when computing suffix arrays is the immense amount of working memory required by the current state-of-the-art algorithms. Even carefully engineered implementations require around 30×–60× the input size as working space [19, 20]. Additionally, there is a significant space-time trade-off. The memory-efficient algorithms tend to be slower. We thus ask the question:

Is there a scaling, fast, and *memory-efficient suffix array construction algorithm in distributed memory?*

Summary of our Contributions. We answer this question positively by providing a scalable, fast, *and* space-efficient distributed-memory suffix array construction algorithm (SACA), using a bucketing approach in conjunction with a randomized (chunk-based) redistribution scheme for load balancing with provable performance guarantees. Our algorithm is the fastest (on all inputs but one) distributed-memory suffix array construction algorithm on up to 12 288 cores and requires only 20×–26× the input size as working space. As a side result of independent interest, we improve the load balancing for a distributed prefix-doubling algorithm, enhancing its performance significantly. This might be of interest for future work, as prefix doubling algorithms can more easily be extended to also compute the longest common prefix (LCP) array, another important data structure for text processing.

Paper Outline. First, in Section 2, we introduce some basic concepts required for suffix array construction and distributed-memory algorithms. Next, in Section 3, we discuss previous work on suffix array construction. In Section 4, we present the main result of this paper.

[1] `https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia`, last accessed 2024-12-11.
[2] `https://archiveprogram.github.com/arctic-vault/`, last accessed 2024-12-11.

We start with a description of the distributed-memory variant of the DCX [26] suffix array construction algorithm in Section 4.2. In Section 4.2.1, we demonstrate how a previously developed technique for space-efficient string sorting [31, 28] can be applied to the DCX suffix sorting to obtain a more lightweight algorithm. Subsequently, in Section 4.2.2, we introduce a *randomized chunking scheme* to provide provable load-balancing guarantees for our space-efficient (suffix) sorting approach, followed by a brief analysis of the overall algorithm in Section 4.2.3. Finally, an experimental evaluation of our highly-engineered implementation using MPI is presented in Section 5, followed by a conclusion in Section 6.

2 Preliminaries

We assume a distributed-memory machine model consisting of p processing elements (PEs) allowing single-ported point-to-point communication. The cost of exchanging a message of h *machine words* between any two PEs is $\alpha + \beta h$, where α accounts for the message start-up overhead and β quantifies the time to exchange one machine word. Let h be the maximum number of words a PE sends or receives, then collective operations like *broadcast, prefix sum, (all-)reduce, and (all-)gather* can be implemented in time $\mathcal{O}(\alpha \log p + \beta h)$ [35]. An alltoall exchange using direct messaging is possible in $\mathcal{O}(\alpha p + \beta h)$ [35].

The input to our algorithms is a text T consisting of $n-1$ characters over an alphabet Σ. By $T[i]$, we denote the i-th character of T for $0 \leq i < n-1$. We assume $T[n-1]$ to be a sentinel character $\$ \notin \Sigma$ with $\$ < z$ for all $z \in \Sigma$. The i-th suffix of T, $s_i = T[i, n)$, is the substring starting at the i-th character of T. Due to the sentinel element all suffixes are prefix-free. The *suffix array* SA contains the lexicographical ordering of all suffixes of T. More precisely, SA is an array of length n with $\mathrm{SA}[i]$ containing the index of the i-th smallest suffix of T. A length-l- (or simply l)-prefix of a suffix with starting position i is the substring $T[i, l)$.

In our distributed setting, we assume that each PE i holds a subarray T_i of T as input such that T is the concatenation of all local input arrays T_i, i.e., $T = T_0 \circ \ldots \circ T_{p-1}$. Furthermore, we assume the input to be well-balanced, i.e., $|T_i| = \Theta(n/p)$. For our DCX algorithm, we assume a suitable padding of up to X sentinel characters at the end of the text.

3 Related Work

There has been extensive research on the construction of suffix arrays in the sequential, external-memory, shared-memory parallel, and (to a somewhat lesser extent) in the distributed-memory setting. All suffix array construction algorithms are based on three general algorithmic techniques: *prefix doubling*, *induced copying*, and *recursion* or combinations thereof. In the following, we give a brief overview of these techniques. For a more comprehensive overview, we refer to the most recent surveys [7, 9].

Prefix-Doubling. In algorithms based on prefix doubling, the suffixes are iteratively sorted by their length-h prefix starting with $h = 1$. Now, all suffixes that share a common h-prefix are said to be in the same h-group and have an h-rank corresponding to the number of suffixes in lexicographically smaller h-groups. By sorting all suffixes based on their h-group, we can compute the corresponding suffix array SA_h. Note that this suffix array does not necessarily have to be unique, as the order of suffixes within an h-group is not unique. If for some h, all h-groups contain only a single suffix, we have $\mathrm{SA}_h = \mathrm{SA}$. Therefore, the idea is to increase h until all h-ranks are unique. To this end, during each iteration, the length of the

considered prefixes is doubled. Fortunately, we do not have to compare the prefixes explicitly. Instead, during iteration $i > 0$, for a suffix s_j starting at index j, the rank of its length-h prefix can be inferred from the pair of ranks $(\text{rank}_{h/2}[j], \text{rank}_{h/2}[j + h/2])$ computed in the previous iterations for the suffixes s_j and $s_{j+h/2}$. This yields the correct result as comparing two suffixes s_i and s_j by their length-h prefix can be achieved by (a) comparing their first $h/2$ characters and – if they tie – (b) comparing the following $h/2$ characters, which in turn is equivalent to lexicographically comparing the ranks of the corresponding $h/2$-prefixes. Using the overlap of suffixes in a text, prefix-doubling boils down to at most $\mathcal{O}(\log n)$ rounds in which n pairs of integers have to be sorted. Thus, this approach has an overall complexity in $\mathcal{O}(n \log n)$ in the sequential setting, when using integer sorting algorithms. The first suffix array construction algorithm [30] is based on prefix-doubling. In the sequential setting, this approach has not received much attention, due to its non-linear running time. However, in distributed memory, the fastest currently known suffix array construction algorithm is based on prefix doubling [20].

Induced-Copying. Induced-copying algorithms sort a (small) subset of suffixes and then *induce* the order of all other suffixes using the subset of sorted suffixes. First, all suffixes are classified using one of two [25, 33] classification schemes. Next, all suffixes necessary for inducing the order of the remaining suffixes are sorted directly. Then, the properties of the classification allow us to induce the order of the remaining suffixes based on their class, starting characters, and preceding or succeeding directly sorted suffix. The inducing part of these algorithms usually consists of just two scans of the text, where for each position only one or two characters have to be compared. Combined with a recursive approach, induced copying algorithms can compute the suffix array in linear time requiring only constant working space in addition to the space for the suffix array [23, 29]. This combination is also very successful, as it is used by the fastest sequential suffix array construction algorithms [4, 18, 24, 32]. Interestingly, there is only one linear-time suffix array construction algorithm based on induced copying that does not also rely on recursion [5]. In distributed memory, induced copying algorithms are space-efficient [19].

Recursive Algorithms. The third and final technique is to use recursion to solve subproblems of ever decreasing sizes. Here, the general idea is to partition the input into multiple different (potentially overlapping) substrings. A subset of these substrings can then be sorted using an integer sorting algorithm (in linear time). If all substrings are unique, we can compute a suffix array together with the remaining suffixes not yet sorted. Otherwise, we recurse on the non-unique ranks of the substrings as new input. We then use the suffix array from the recursive problem to compute the unique ranks from the original subset of substrings. The first linear-time suffix array construction algorithm is purely based on recursion [26]. This algorithm is also the foundation of the distributed-memory suffix array construction algorithm presented in this paper. It already has been considered in distributed memory [6, 10]. However, all implementations are direct adaptations of the sequential algorithm to distributed memory.

4 A Space-Efficient Variant of Distributed DCX

In this section, we describe the general idea of the sequential DC3 algorithm [26]. Then, we give a canonical transformation of the sequential DC3 algorithm to a distributed-memory algorithm. Here, we also consider the more general form – the DCX algorithm. Finally, we discuss how to optimize this canonical transformation into a scaling, fast, and memory-efficient distributed suffix array construction algorithm.

4.1 The Sequential DCX Algorithm

The *skew* or **D**ifference **C**over 3 algorithm (DC3) – and its generalization DCX – is a recursive suffix array construction algorithm that exhibits linear running time (in the sequential setting). As our main contribution is a fast and lightweight distributed variant of the algorithm, we now briefly discuss its key ideas, beginning with the eponymous concept of a *difference cover*.

Let X be a positive integer. A subset $D_X \subseteq \{0, \ldots, X-1\}$ is a difference cover modulo X if for every $r \in \{0, \ldots, X-1\}$ there are $i, j \in D_X$ such that $i - j \bmod X = r$. Put differently, D_X *covers* the set $\{0, \ldots, X-1\}$ in the sense that we have $\{0, \ldots, X-1\} = \{i - j \bmod X \mid i, j \in D_X\}$.

A key property of a difference cover modulo X, on which the DCX algorithm relies, is that for all $i, j \in \mathbb{N}$, there is a $0 \leq l < X$ such that $(i+l) \bmod X \in D_X$ and $(j+l) \bmod X \in D_X$. $X = 3$ is the smallest X for which a non-trivial difference cover exists, with $D_3 = \{1, 2\}$.

DCX Algorithm. We now discuss the main steps of the DCX algorithm. It starts by classifying the suffixes s_j of a given text T into different sets, based on their starting position j. Suffixes with index $(j \bmod X) \in D_X$ constitute the *(difference cover) sample* S_{D_X}.

For now, assume that we know a relative ordering of the suffixes in S_{D_X} within the text. Since D_X is a difference cover, for *any* two suffixes s_i and s_j, there is an $l < X$ such that s_{i+l} and s_{j+l} are in S_{D_X} for which we already know a relative ordering. Hence, for comparing s_i and s_j, it is sufficient to compare the l-prefixes of s_i and s_j, using the ranks of s_{i+l} and s_{j+l} to break ties. That is, we compare tuples $(T[i, i+l), \text{rank}[i+l])$ and $(T[j, j+l), \text{rank}[j+l])$. For $X = 3$, this rank-*inducing* can be achieved using linear-time integer sorting.

It remains to discuss how the relative ordering of the sample suffixes can be obtained. We start by sorting the X-prefixes of the sample suffixes. If they are unique, we are already done, as we can take their rank for the ordering. Otherwise, we replace the sample suffixes j with the rank of their X-prefix and reorder them by $(j \bmod X, j \operatorname{div} X)$. This results in an auxiliary text T' that contains the renamed difference cover sample suffixes in original text order within each equivalence class in D_X. We can now recursively apply the algorithm to this text T', yielding a suffix array SA'. From SA' we can retrieve a relative ordering of the sample suffixes with regard to the original text T.

For DC3, the number of sample suffixes is $\leq 2/3n$. As all other operations can be achieved with work linear in the size of the input, the overall complexity of the algorithm is in $\mathcal{O}(n)$.

4.2 The Distributed DCX Algorithm

Our distributed suffix array construction is a simple and practical distributed variant of the DCX algorithm for $X \geq 3$. Algorithm 1 shows a high-level pseudocode for the algorithm.

We now discuss the algorithm in some more detail. The input to the algorithm on PE i is the local chunk T_i of the input text T.

1. **Sort the Difference Cover Sample.** In the first phase of the algorithm, we select, on each PE i, the difference cover (DC) sample suffixes starting at (global) positions j with $(j \bmod X) \in D_X$. As in the sequential setting, we want to compute unique ranks of these suffixes first. For that, we first globally sort the X-prefixes of all DC sample suffixes. If all of them are unique, this already constitutes the relative ordering of the sample suffixes within the final suffix array. This rank information can then be used to compare any two suffixes s_i and s_j (see Section 4.1) and we continue with step three. Otherwise, we have to recurse on the sample suffixes as described in the following step two of the algorithm.

Algorithm 1 High-level overview of a simple distributed variant of the DCX algorithm.

Input: Text T_i on PE i.
Output: Local part of distributed suffix array of T.

1 $o_i = \texttt{PrefixSum}(|T_i|)$ `// global text index offset`
2 $C_i = \langle 0 \leq j < |T_i| \mid (j + o_i \bmod X) \in D_X \rangle$ `// DC sample positions`
3 $S_i = \langle (\underbrace{T_i[j, j+X)}_{X\text{-prefix}}, \underbrace{j + o_i}_{\text{global idx}}) \mid j + o_i \in C_i \rangle$ `// (X-prefix,idx) of DC samples`
4 globally **sort** S_i by first entry
5 **if** *all first entries of S are unique* **then**
6 $\quad R_i = \langle (\texttt{rank}(t, S), j) \mid t = (\text{prefix}, j) \in S_i \rangle$ `// unique ranks of DC samples`
7 **else**
8 $\quad P_i = \langle (\texttt{rank}(t, S), j) \mid t = (\text{prefix}, j) \in S_i \rangle$ `// replace X-prefix with rank`
9 $\quad$ globally **sort** P_i by $(j \bmod X, j \operatorname{div} X)$
10 $\quad SA'_i \leftarrow$ recursively call DCX on $T'_i = \langle r \mid (r, j) \in P_i \rangle$
11 $\quad R_i = \langle (j, \underbrace{\texttt{mapback}(SA[j])}_{\text{restore idx of DC samples in } T}) \mid 0 \leq j < |SA'_i| \rangle$
12 globally **sort** R_i by second component `// bring R_i in text order`
`// construct (X-prefix, <ranks>) tuples for all suffixes`
13 $S_i = \langle (\underbrace{T_i[j, \ldots, j+X)}_{X\text{-prefix}}, \underbrace{R_i[\texttt{next}(j,\ 1)], \ldots, R_i[\texttt{next}(j,\ v)]}_{\text{ranks of DC samples following } j}, \underbrace{j + o_i}_{\text{global idx}})) \mid 0 \leq j < |T_i| \rangle$
14 globally **sort** S_i by appropriate comparison function (see [26])
15 output last entry of S_i as suffix array SA_i

2. **Compute Unique Ranks Recursively.** If the ranks are not already unique, we locally create an array P_i by replacing each entry $(X-\text{prefix}, j)$ of S_i with $(\text{rank}[j], j)$. The distributed array $S = \bigcup S_i$ contains the DC sample suffixes sorted by their X-prefix. The rank$[j]$ of the j-th sample suffix is the global rank of its X-prefix within S. Equal prefixes obtain the same rank. Since S is sorted, the ranks can be computed by a prefix sum. Afterwards, we globally sort P_i by $(j \bmod X, j \operatorname{div} X)$. This rearranges the newly renamed sample suffixes in their original order by respecting the equivalence class of their starting index within D_X. We then recursively call the DCX algorithm on the text T'_i where T'_i contains the new names of the sample suffixes from P_i dropping the index. From the suffix array SA' of T', we can determine the rank of each sample suffix j using the `mapback` function which maps the index of a DC sample in T' back to its original position in T. Due to the construction of T', the unique ranks of the DC sample suffixes within T' also yield a relative ordering of the DC sample suffixes in T [26].
3. **Sort All Suffixes.** Now, we construct for each suffix s_j in T a tuple containing: (1) the X-prefixes of the suffix, (2) the previously computed ranks of the following $|D_X|$ DC samples after s_j in text order, and (3) its global index j within T. Globally sorting theses tuples using the previously discussed comparison function for suffixes s_i, s_j yields the suffix array of the original text T.

Existing distributed DC$\{3, 7, 13\}$ implementations [27, 6] broadly follow this straightforward adaptation of the algorithm to the distributed setting. However, this approach is not space-efficient. Materializing the X-prefixes of the suffixes for the final sorting step results in a memory blow-up proportional to X compared to the actual input. Consequently, sorting suffixes on distributed machines using DCX with large X is not feasible due to the limited main memory, even though DCX with $X > 3$ shows better performance on many real-world inputs [19]. We propose a technique to overcome this problem in the following Section 4.2.1.

4.2.1 Bucketing

In the sequential or shared-memory parallel setting, X-prefixes of suffixes can be sorted space-efficiently as each such element e can be represented as a pointer to the starting position of the suffix within the input text. This space-efficient sorting, however, is no longer possible in distributed memory. If we want to globally sort a distributed array of suffix-prefixes, we have to fully materialize and exchange them – resulting in a memory blow-up of at least a factor X. A simple idea to prevent this blow-up is to use a partitioning strategy which divides the elements of the distributed array into multiple buckets using splitter elements and processes only one bucket at a time.

We now describe a generalization of a bucketing technique for space-efficient sorting which has been previously proposed for scalable distributed string sorting [31, 28].

Whenever a distributed array of elements with a space-efficient representation has to be globally sorted, we first determine $q+1$ global splitter elements $s_0, s_1, \ldots s_q$ with $s_0 = -\infty$ and $s_q = \infty$. We then locally partition the array into q buckets, such that element e with $s_k < e \leq s_{k+1}$ is placed in bucket k. We then execute q global sorting steps. In each step k, we materialize and communicate the elements from bucket k using a distributed sorting algorithm. Assuming that the splitters are chosen such that the global number of elements in each bucket is n/q and the elements within each bucket are equally distributed among the PEs (see Section 4.2.2 how this can be ensured), we only have to materialize $n/(pq)$ elements per bucket and PE instead of n/p elements per PE when using only one sorting phase.

By choosing q proportional to the memory blow-up caused by materializing an element, we can keep the overall memory consumption of this sorting approach in $\mathcal{O}(n/p)$.

4.2.2 Load-Balanced Bucketing via Random Chunk Redistribution

The global number of elements per bucket can be balanced by judiciously choosing the splitter elements, e.g., by regular or random sampling, or even using multi-sequence selection [36, 2]. However, the number of elements per PE within a bucket can vary greatly depending on the input. Assume an input which is already globally sorted with $q < p$ buckets. In this setting, all n/p elements located on the first PE have to be materialized when processing the first bucket. This results in memory blow-up and poor load-balancing across the PEs. Increasing the number of buckets q can only address the memory consumption issue but does not help with load-balancing.

A standard technique to resolve this kind of problem is a random redistribution of the elements to be sorted. However, this is not directly possible for elements which are stored in a space-efficient manner as in our case. We propose to solve this problem by randomly redistributing not single prefixes of suffixes but whole chunks of the input text before sorting. To be more precise, we partition T into consecutive chunks C_j of size c, i.e., $T = C_0 \circ \ldots \circ C_{n/c-1}$. For the sake of simplicity, we assume chunks to be aligned with PE boundaries. Each chunk is then sent to a random PE.

▶ **Theorem 1** (Random Chunk Redistribution). *When redistributing chunks of size c uniformly at random across p PEs, with $q < p$ buckets each containing n/q elements, the expected number of elements from a single bucket received by a PE is $n/(pq)$.*

Furthermore, the probability that any PE receives $2n/(pq)$ or more elements from the same bucket is at most $1/p^\gamma$ for $n \geq 8c(\gamma+2)pq\ln(p)/3$ and $\gamma > 0$.

Proof. Let Y_i^k denote the number of elements belonging to bucket k which are assigned to PE i. In the following, we will determine the expected value of Y_i^k and show that $\mathbb{P}[Y_i^k \geq 2\mathbb{E}[Y_i^k]]$ is small. This will then be used to derive the above-stated bounds.

Let c_j^k be the number of elements belonging to bucket k in chunk j. For the sake of simplicity, we assume all buckets to be of equal size, thus, $\sum_{j=0}^{n/c-1} c_j^k = n/q$. We define

$$X_{j,i}^k = \begin{cases} c_j^k & \text{if chunk } j \text{ is assigned to PE } i \\ 0 & \text{otherwise,} \end{cases}$$

for chunk j with $0 \leq j < n/c$, PE i with $0 \leq i < p$, and bucket k with $0 \leq k < q$. Thus, the random variable $X_{j,i}^k$ indicates the number of elements from bucket k received by PE i if chunk j is assigned to this PE. Hence, we can express Y_i^k as the sum over all $X_{j,i}^k$, i.e., $Y_i^k = \sum_{j=0}^{n/c-1} X_{j,i}^k$. As all chunks are assigned uniformly at random and there are p PEs, we furthermore have $\mathbb{E}[X_{j,i}^k] = c_j^k/p$. By the linearity of expectation, we can derive the expected value of Y_i^k as

$$\mathbb{E}[Y_i^k] = \mathbb{E}\left[\sum_{j=0}^{n/c-1} X_{j,i}^k\right] = \sum_{j=0}^{n/c-1} \mathbb{E}[X_{j,i}^k] = \sum_{j=0}^{n/c-1} \frac{c_j^k}{p} = \frac{n}{pq}.$$

For each bucket k, we now bound the probability $\mathbb{P}[Y_i^k \geq 2n/(pq)]$ that PE i receives two times its expected number of elements or more. We have

$$\mathbb{P}\left[Y_i^k \geq \frac{2n}{pq}\right] = \mathbb{P}\left[\sum_{j=0}^{n/c-1} X_{j,i}^k \geq \frac{2n}{pq}\right] = \mathbb{P}\left[\sum_{j=0}^{n/c-1} X_{j,i}^k - \mathbb{E}[X_{j,i}^k] \geq \frac{n}{pq}\right].$$

As the value of $X_{i,j}^k$ is bounded by the chunk size c, the Bernstein inequality [11, Theorem 2.10, Corollary 2.11] yields the following bound

$$\mathbb{P}\left[\sum_{j=0}^{n/c-1} X_{j,i}^k - \mathbb{E}[X_{j,i}^k] \geq \frac{n}{pq}\right] \leq \exp\left(-\frac{\left(\frac{n}{pq}\right)^2}{2\left(\sum_{j=0}^{n/c-1} \mathbb{E}[(X_{j,i}^k)^2] + \frac{cn}{3pq}\right)}\right). \tag{1}$$

Since we find $\mathbb{E}[(X_{j,i}^k)^2] = (c_j^k)^2/p$, it follows that

$$\sum_{j=0}^{n/c-1} \mathbb{E}[(X_{j,i}^k)^2] = \sum_{j=0}^{n/c-1} (c_j^k)^2/p \leq \frac{1}{p} \sum_{j=0}^{n/(qc)-1} c^2 = \frac{cn}{pq},$$

as the sum of the squares of a set of elements $0 \leq a_i \leq c$ with $\sum_i a_i = b$ and b divisible by c is maximized if they are distributed as unevenly as possible, i.e., $a_i = c$ for b/c elements and 0 for all others. We can use this estimation for an upper bound on the right-hand side of (1)

$$\exp\left(-\frac{\left(\frac{n}{pq}\right)^2}{2\left(\sum_{j=0}^{n/c-1} \mathbb{E}[(X_{j,i}^k)^2] + \frac{cn}{3pq}\right)}\right) \leq \exp\left(-\frac{\left(\frac{n}{pq}\right)^2}{2\left(\frac{cn}{pq} + \frac{cn}{3pq}\right)}\right) = \exp\left(-\frac{3n}{8pqc}\right). \tag{2}$$

Combining these estimations, we obtain the bound

$$\mathbb{P}\left[Y_i^k \geq \frac{2n}{pq}\right] \leq \exp\left(-\frac{3n}{8pqc}\right) \leq \exp\left(-(\gamma+2)\ln p\right) = \frac{1}{p^{\gamma+2}}$$

for $n \geq 8pqc\ln(p)(\gamma+2)/3$. Using the union-bound argument yields the following estimation

$$\mathbb{P}\left[\bigcup Y_i^k \geq 2\frac{n}{pq}\right] \leq \sum_{i=0}^{p-1}\sum_{k=0}^{q-1} \mathbb{P}[Y_i^k \geq 2\frac{n}{pq}] \leq \sum_{i=0}^{p-1}\sum_{k=0}^{q-1} \frac{1}{p^{\gamma+2}} \leq \frac{1}{p^\gamma}.$$

Hence, we obtain $\frac{1}{p^\gamma}$ as an upper bound on the probability that any PE receives more than two times the expected number of elements $n/(pq)$ for any bucket when assuming $q \leq p$. ◀

Theorem 1 shows that combining a random chunk redistribution with the bucketing approach yields a space-efficient solution to the sorting problems occurring within our distributed variant of the DCX algorithm.

Within the DCX algorithm – depending on the actual step – we do not only redistribute chunks of the text but also corresponding rank entries and some bookkeeping information like the global index of a chunk. Furthermore, we require each chunk to have an *overlap* of X characters to ensure that an X-prefix for each element within a chunk can be constructed without communication. By choosing $c \geq X$, one can assure that the total size of a chunk remains in $\mathcal{O}(c)$. The redistribution can be achieved in time $\mathcal{O}(\alpha p + \beta n/p)$ as each PE sends and receives at most $\mathcal{O}(n/p)$ elements assuming the conditions of Theorem 1 are met.

4.2.3 Overall Analysis

We now briefly outline the running time of our algorithm using bucketing in conjunction with random chunk redistribution. All steps outside the recursion of distributed DCX are either global sorting steps, alltoall exchanges or local work linear in the size of the text per PE (see Algorithm 1). Additionally, we perform a constant number of allreduce and prefix sum operations, whose running time is dominated by the alltoall exchanges.

First, we consider the sorting steps. Assume that we want to sort a distributed array of m elements of type e with $\mathcal{O}(m/p)$ elements per PE, each of size $\mathcal{O}(e_s)$ while comparing any two elements takes time $\mathcal{O}(e_c)$. To facilitate the analysis, we assume a distributed (k-level) sorting routine with a running time

$$\mathrm{T}_{\text{sort}}(m,p,k) = \mathcal{O}\left(\underbrace{e_c \frac{m}{p}\log m}_{\text{local work}} + \underbrace{\alpha k \sqrt[k]{p}}_{\text{start-up latency}} + \underbrace{\beta k e_s \frac{m}{p}}_{\text{communication costs}}\right)$$

which also balances the output, i.e., after sorting, each PE holds $\Theta(m/p)$ elements.[3] A sorting step with $q < p$ buckets and random chunk redistribution can then be realized in time

$$\begin{aligned}\widetilde{\mathrm{T}}_{\text{sort}}(m,p,q,k) &= q\mathrm{T}_{\text{sort}}(\mathcal{O}(m/q),p,k) + \mathcal{O}\left(\alpha p + \beta\frac{m}{p}\right)\\ &= \mathcal{O}\left(\underbrace{e_c \frac{m}{p}\log m}_{\text{local work}} + \underbrace{\alpha(p + qk\sqrt[k]{p})}_{\text{start-up latency}} + \underbrace{\beta k e_s \frac{m}{p}}_{\text{communication costs}}\right)\end{aligned}$$

with probability $\geq 1 - 1/p^\gamma$ for any fixed $\gamma > 0$ and sufficiently large m, increasing only the latency term due to q sorting steps and an additional alltoall exchange. Note that the time required for the computation of the bucket splitters and subsequent classification is dominated by the running time mentioned above when using regular sampling with $\Theta(q)$ samples per PE and the distributed (k-level) sorting routine.

In this setting, we see that all sorting steps (and the alltoall exchanges) in DCX are dominated by the final sorting of all suffixes, where we have $e_s = \mathcal{O}(X \log\sigma + \sqrt{X}\log n)$[4] and $e_c = \mathcal{O}(X)$. Due to the geometric decrease of the input size within the recursion, we

[3] Note that for example the running time of adaptive multi-level sample sort (AMS) [2, Theorem 3] using k levels of indirection comes close to this. AMS also balances the output.

[4] Within the recursive calls, we have $e_s = \mathcal{O}(X \log n)$ as the alphabet size is now $\mathcal{O}(n)$. However, as the problem size also decreases by a factor $\Theta(\sqrt{X})$ [13, 26], this increase in alphabet size can be accounted for by the $\mathcal{O}(\sqrt{X}\log n)$ term in e_s of the first level.

reach an overall problem size of $\mathcal{O}(n/p)$ after $\mathcal{O}(\log p)$ iterations. Gathering all remaining data on a single PE and applying the sequential DCX algorithm yields the running time stated in Theorem 2.

▶ **Theorem 2.** *Using a distributed sorting routine with the above-stated properties, our lightweight distributed DCX algorithm has a running time*

$$DCX(n,p) = \mathcal{O}\left(\underbrace{X\frac{n}{p}\log n}_{\text{local work}} + \underbrace{\alpha \log(p)(p + qk\sqrt[k]{p})}_{\text{start-up latency}} + \underbrace{\beta X\frac{n}{p}\left(\log(\sigma) + \frac{\log n}{\sqrt{X}}\right)}_{\text{communication costs}}\right)$$

with probability $\geq 1 - 1/p^{\gamma}$ *for any fixed* $\gamma > 0$ *and sufficiently large* n.

4.2.4 Further Optimizations

In addition to the techniques described above, we also utilize discarding and packing, two techniques commonly used in distributed- and external-memory SACAs.

Discarding. After sorting the X-prefixes of the sample suffixes, we have to recursively apply the DCX algorithm (or any other suffix sorting algorithm) to a smaller subproblem if there are duplicate ranks. However, in order to obtain overall unique ranks for the sample suffixes, we do not have to recurse on all of them but can discard suffixes whose ranks are already unique after initial sorting. This *discarding* technique has been used in the external-memory setting [15] but has not been explored for distributed memory yet.

Packing. Packing is an optimization for small-sized (integer) alphabets proposed for distributed memory prefix-doubling by Flick et al. [20]. Assume $b = \lceil \log \sigma \rceil < B$, where B is the size of one machine word. Instead of using one machine word per character, we can pack up to B/b characters into one word and exchange/sort them at once.

5 Experimental Evaluation

For our extensive evaluation, we use up to 256 compute nodes (12 288 cores) of SuperMUC-NG where each node is equipped with one Intel Skylake Xeon Platinum 8174 processor with 48 cores and 96GB of main memory.[5] The internal interconnect is a fast OmniPath network with 100 Gbit/s.

We compare the following algorithms. See Table 1 for an overview.

ℓDCX. Our implementation of the lightweight distributed DCX algorithm described above. In the experiments, we use $X = 39$. In preliminary experiments, we found this value to provide the best trade-off between reduction ratio and running time per level. We use the bucketing technique for sorting the DC samples and also for the final sorting of all suffixes. For sorting the DC samples, we use 16 buckets in the first two levels of recursion. For the final merging, we use 64 buckets in the first two levels and 16 in a third level. In later levels of recursion, the remaining number of suffixes is small enough so that all elements can

[5] We also conducted preliminary experiments on the HoreKa supercomputer. As the results obtained there are in line with our findings from SuperMUC-NG presented here, we omit them.

be sorted at once. Bucket splitters are determined by centrally sorting a random sample of 2×10^4 elements and then drawing the splitters equidistantly from the sorted sample elements. Additionally, we employ our randomized redistribution of chunks with 10^4 chunks per PE. We use AMS-sort by Axtmann et al. [2, 3] as our distributed sorting routine. Due to the large number of single sorting steps, we require a low-latency sorting algorithm (at least for large PE configurations). Therefore, we use AMS with two levels as a default. As the discarding optimization has certain overheads, we only apply it when the reduction ratio is more than 0.7. Furthermore, we use 40-bit integers for storing rank information to reduce the memory footprint.

PSAC [20]. We explore two different original configurations. PSAC is the standard (more memory-efficient) configuration performing prefix-doubling as described in Section 3. Once the ratio of suffixes with non-unique h-prefix falls below $n/10$, it switches to another algorithm which tries to avoid global sorting steps as much as possible but has a somewhat higher memory consumption. This approach sorts each non-unique range of suffixes with identical h-prefix independently by splitting the PEs into different groups and can be regarded as a kind of discarding. The configuration PSAC$^+$ runs this second algorithm immediately. Additionally, we implemented a 40-bit integer version of their algorithm as well as a version which uses AMS for distributed sorting. Due to the small performance benefit of using AMS, we do not include these variants in the plots.

dPD [19]. This algorithm also relies on distributed prefix doubling with discarding, i.e., it does not continue sorting suffixes whose h-prefix is already unique. In contrast to PSAC$^+$, dPD redistributes the remaining non-unique suffixes across all PEs and keeps using a global sorting routine. dPD uses 40-bit integers.

The original sorting algorithm within dPD gathers suffixes with identical h-prefix on the same PE. While this facilitates the rank computation for the next prefix-doubling round, it introduces severe load-balancing problems in case of many identical h-prefixes. We provide an improved version of this algorithm, where identical h-ranks may span multiple consecutive PEs. In combination with AMS as a sorting subroutine, this yields a much more scalable algorithm with a substantially improved load-balancing. We refer to this new variant as dPD* (dPD^{*,1} and dPD^{*,2} indicates the usage of one- or two-level AMS sort).

B-DC$_X$ [6]. A straightforward adaptation of the sequential DCX algorithm to the distributed-memory setting for $X = \{3, 7, 13\}$. The implementation is limited to inputs of size 4 GB due to the internal usage of 32-bit integers.

Other Algorithms. There also exists a lightweight distributed suffix sorting algorithm based on induced copying [19]. Unfortunately, we were not able to run this algorithm successfully on our system. It also suffers from the same load-balancing issues as dPD. While this algorithm exhibits a relatively low memory blow-up factor of around 30 on up to 1 280 cores, the running-time of the algorithm is reported to be always slower than or on par with the default configuration of PSAC or the non-load-balanced dPD.

There also exists prefix-doubling and DC3/DC7 implementations [10] in the distributed external-memory BigData framework *Thrill* [8]. Although the framework supports MPI, their focus is on non-MPI data center clusters. We were not able to execute their algorithms on more than one compute node because Thrill relies on external local disk storage, which

Table 1 Overview of algorithms used in this experimental evaluation. All algorithms, except for libsais are distributed-memory algorithms.

Name	Brief Description	Reference
ℓDCX	Our main contribution described in Section 4	[here]
PSAC	Prefix doubling w/o discarding, last 10 % are sorted with PSAC$^+$	[20]
PSAC$^+$	Prefix doubling, each h-group is sorted independently (discarding-like)	[20]
PSAC$_{40}$	Same as PSAC but using 40-bit integers	[here]
PSAC$^+_{40}$	Same as PSAC$^+$ but using 40-bit integers	[here]
dPD	Prefix doubling w/ discarding, 40-bit integers, and PE-local ranks	[19]
dPD^{*,1}	Same as dPD but w/o PE-local ranks and using AMS with one level	[here]
dPD^{*,2}	Same as dPD but w/o PE-local ranks and using AMS with two levels	[here]
B-DC$_X$	Basic MPI implementation of DCX for $X = \{3, 7, 13\}$	[6]
libsais	Fastest sequential and shared-memory suffix sorting algorithm	[24]

is very limited on our cluster. On a single compute node (48 cores), their algorithms were more than a factor of two slower than ℓDCX on all data sets but DNA-Rep. Here, their DC7 implementation is only a factor of 1.8 slower than our algorithm.

Recently and independently, Ferguson also started working on a variant of a distributed-memory difference cover algorithm using a similar bucketing approach, implemented in the Chapel parallel programming language [16]. Their algorithm does not use random chunk redistribution and other optimizations present in our implementation. Preliminary comparisons executed by Ferguson on a low-latency Cray supercomputer demonstrated that our implementation was approximately twice as fast on a DNA dataset. In preliminary experiments on our system – which has higher communication latency – Ferguson's implementation exhibited less favorable scaling behavior. Additionally, it requires around 20×–80× the input size as working memory. Therefore, it is not included in our evaluation.

In addition to the above-mentioned distributed-memory algorithms, we also compared our algorithm to the currently fastest and highly-engineered sequential and shared-memory parallel suffix sorting implementation libsais [24].

General Settings and Data Sets. All algorithms are implemented in C++ (C for libsais) and compiled with `GCC 12.2.0` with optimization flags `-O3` and `-march=native`. We use IntelMPI 2021.11 for interprocess communication. Additionally, our DCX implementation uses the zero-overhead MPI bindings KaMPIng [40]. Reported times are the average of at least three runs. We generally observed only very low variations in running time.

We use inputs from the five data sets given in Table 2 with different characteristics regarding the alphabet size σ and the longest common prefix (LCP) distribution.

5.1 Weak-Scaling Experiments

Figure 1 shows the running times of weak-scaling experiments with 20 MB of text data per PE (960 MB per compute node). In Figure 2, we present the corresponding memory blow-up, i.e., the maximum peak memory aggregated over each compute node divided by the total input size on a compute node. We read the maximum *resident set size (rss)* of each MPI process to obtain these values.

Table 2 Data sets used in our evaluation. Statistics consider the first 50 GB of each data set.

Name	Brief Description	σ	avg. LCP	max LCP
CC [14]	Websites crawled by Common Crawl Project	243	$1{,}04 \times 10^4$	$1{,}84 \times 10^6$
Wiki [41]	XML data sets of Wikipedia	213	396.07	$1{,}58 \times 10^6$
Prot [39]	Protein data from *Universal Protein Resource*	26	179.31	$3{,}43 \times 10^4$
DNA [38]	DNA Data from *1000 Genomes Project*	4	25.15	$3{,}57 \times 10^3$
DNA-Rep	Concatenate first MB of DNA 245 760×	4	$2{,}50 \times 10^{10}$	$5{,}00 \times 10^{10}$

We see that ℓDCX is the fastest (distributed) algorithm on CC on all evaluated numbers of PEs. Compared to the previous algorithms, we achieve speedups of up to 4.3, while being at least 1.6× more memory-efficient. The improved prefix-doubling algorithm dPD* performs better, especially when using AMS with two levels (dPD^{*,2}). However, ℓDCX is still about a factor of 1.6 faster. A similar trend can be seen for Wiki, although less pronounced, with speedups of 3.2 and 1.2 on 256 nodes, respectively. This is due to the lower LCP values of Wiki compared to the CC data set. The prefix-doubling algorithms require fewer iterations and therefore perform better. This becomes even more evident for DNA and Prot, as these data sets have even lower LCP values. Additionally, the smaller alphabet size of these texts makes the packing optimization more effective. These factors explain the lower running time of all algorithms on these data sets. On DNA, our improved dPD* is fastest on 256 compute nodes and outperforms ℓDCX by a factor of 1.2.

Due to the repetitive nature of DNA-Rep, the data set has very high average LCP values and therefore is a particularly hard instance for prefix-doubling algorithms. Consequently, these algorithms were not able to terminate within the time limit for all PE configurations (or even crashed on some configurations). Our ℓDCX algorithm is, due to its algorithmic properties, largely insensitive to this input characteristic and shows a very good scaling behavior even on this hard input instance with speedups of up to 5.5 over its competitors.

The DC$\{3, 7, 13\}$ implementation by Bingmann (B-DC$_X$) does not scale well.[6] However, at least the DC$\{7, 13\}$ variants have competitive running times for CC and Wiki on up to two compute nodes. Nonetheless, their memory consumption reveals a key problem of a straightforward adaptation of DCX to distributed memory. Although using 32-bit integers internally, DC3 exhibits a blow-up of $\approx 50\times$ while DC7 and DC13 have a blow-up of $\geq 60\times$ and $\approx 80\times$, respectively.

The running times for the shared-memory parallel variant of libsais (using all 48 cores of one compute node) are given for the input of all p cores. We see that libsais is faster than ℓDCX when using only one compute node. This is not surprising as libsais leverages native OpenMP parallelization while our implementation of DCX uses interprocess-communication via MPI even when running on only one compute node. But from only two compute nodes on (4 compute nodes for DNA-Rep) ℓDCX clearly surpasses libsais effectively utilizing the additional cores. We cannot run libsais on inputs larger than $8 \times 48 \times 20$ MB due to the limited main memory of 96 GB per compute node.

Overall, our lightweight distributed DCX algorithm shows a very good scaling behavior and is either fastest or competitive with our improved version of prefix-doubling with discarding, which is fastest on the remaining inputs (again at least for large PE configurations).

[6] While the scaling behavior of the implementation could certainly be improved, we chose not to do so as our engineered space-efficient ℓDCX algorithm already supersedes it in terms of implementation.

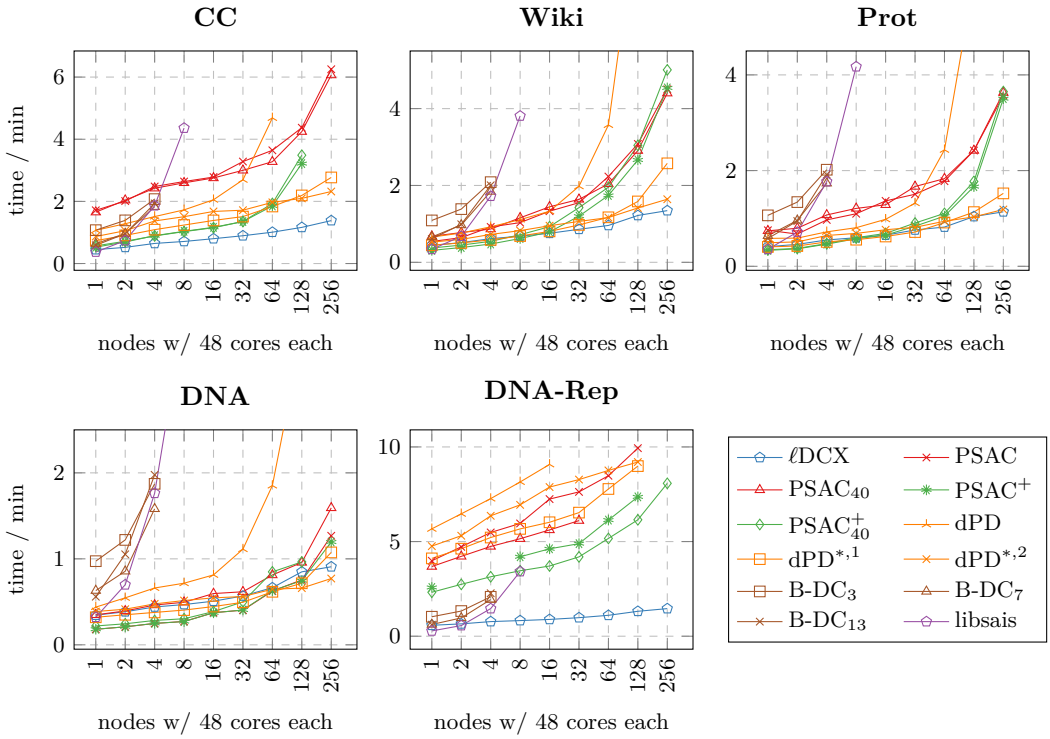


Figure 1 Running time in minutes of the suffix array construction algorithms in our weak-scaling experiments with 20 MB per PE. Missing data points are due to time limits or crashes.

Regarding the memory consumption (Figure 2), we see that ℓDCX is clearly the most memory-efficient one of all evaluated distributed algorithms due to the bucketing with randomized chunk redistribution approach. We see a slight increase in memory consumption for larger PE configurations which we assume to be caused by external factors such as the internal buffer allocation strategy of the MPI implementation and the release policy of memory allocators as we could not detect a significant increase in bucket imbalances.

5.2 Breakdown Experiments

Figure 3 shows the result of a breakdown test, where we run the algorithms on 16 compute nodes (768 cores) and increase the input size per PE until the algorithms can no longer finish due to memory limitations. We start the experiment with 20 MB (15.36 GB in total). The standard PSAC variants using 64-bit integers can handle up to 30 MB (at least for DNA). The 40-bit variants as well as the different variants of dPD can handle up to 40 MB of text per PE. Our ℓDCX implementation is, as expected, far more memory-efficient and can handle up to 130 MB of input per PE (6.24 GB per compute node). With inputs size larger than 100 MB per PE, the maximum memory consumption of ℓDCX is only between 14× to 15× the input size. We attribute this to fixed-size memory overheads of the MPI runtime becoming less important with increasing input size. These values come close to the theoretically expected memory consumption of our implementation.

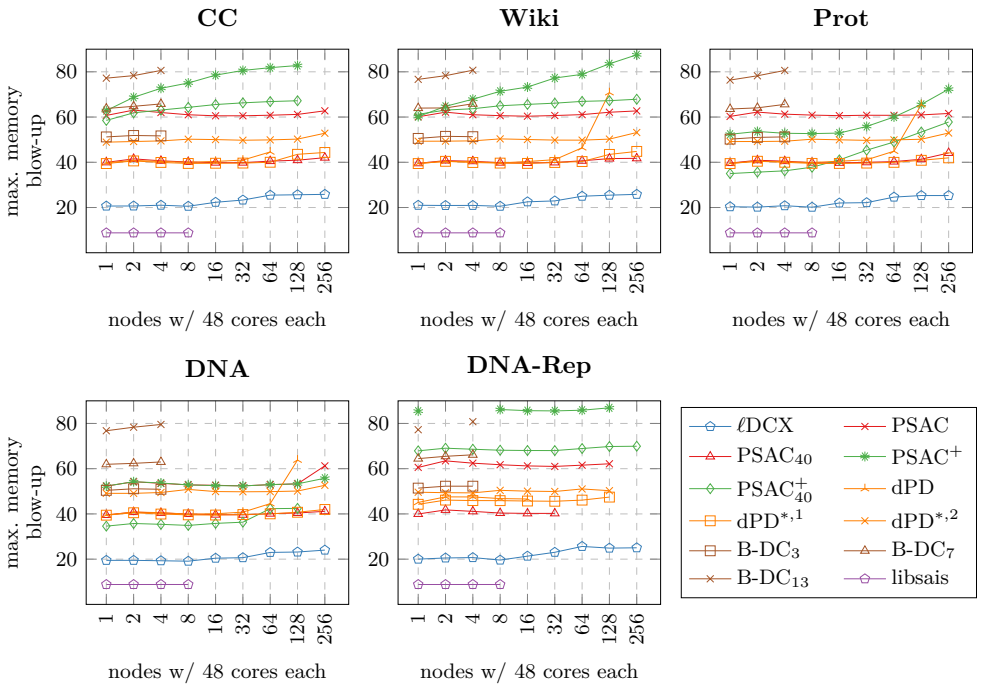


Figure 2 Memory blow-up of the suffix array construction algorithms in our weak-scaling experiments with 20 MB per PE.

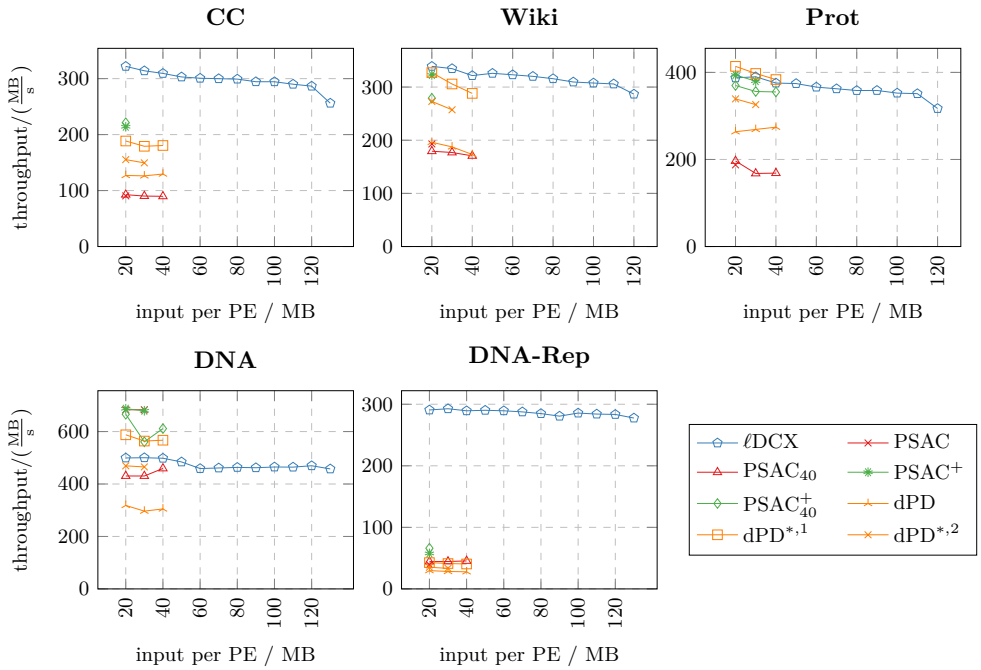


Figure 3 Breakdown test on 16 compute nodes (768 cores) with increasing input size.

6 Conclusion and Future Work

In this work, we present a fast and lightweight distributed variant of the DCX suffix sorting algorithm. We rely on a bucketing scheme for space-efficient distributed sorting to reduce the algorithm's memory consumption and combine it with a load-balancing approach based on random chunk redistribution. In our extensive experimental evaluation, we show that our algorithm is up to 5× faster than previous distributed suffix sorting algorithms while being substantially more memory-efficient. For future work, we plan to extend our bucketing and random chunk redistribution approach to other models of computing such as the distributed external-memory model and distributed multi-GPU architectures.

References

1 Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *J. Discrete Algorithms*, 2(1):53–86, 2004. doi:10.1016/S1570-8667(03)00065-0.

2 Michael Axtmann, Timo Bingmann, Peter Sanders, and Christian Schulz. Practical massively parallel sorting. In *SPAA*, pages 13–23. ACM, 2015. doi:10.1145/2755573.2755595.

3 Michael Axtmann and Peter Sanders. *Robust Massively Parallel Sorting*, pages 83–97. SIAM, 2017. doi:10.1137/1.9781611974768.7.

4 Johannes Bahne, Nico Bertram, Marvin Böcker, Jonas Bode, Johannes Fischer, Hermann Foot, Florian Grieskamp, Florian Kurpicz, Marvin Löbel, Oliver Magiera, Rosa Pink, David Piper, and Christopher Poeplau. Sacabench: Benchmarking suffix array construction. In *SPIRE*, volume 11811 of *Lecture Notes in Computer Science*, pages 407–416. Springer, 2019. doi:10.1007/978-3-030-32686-9_29.

5 Uwe Baier. Linear-time suffix sorting – A new approach for suffix array construction. In *CPM*, volume 54 of *LIPIcs*, pages 23:1–23:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.CPM.2016.23.

6 Timo Bingmann. pdcx. https://github.com/bingmann/pDCX, 2018.

7 Timo Bingmann. *Scalable String and Suffix Sorting: Algorithms, Techniques, and Tools*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2018. URL: https://publikationen.bibliothek.kit.edu/1000085031.

8 Timo Bingmann, Michael Axtmann, Emanuel Jöbstl, Sebastian Lamm, Huyen Chau Nguyen, Alexander Noe, Sebastian Schlag, Matthias Stumpp, Tobias Sturm, and Peter Sanders. Thrill: High-performance algorithmic distributed batch data processing with c++. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 172–183. IEEE, 2016. doi:10.1109/BIGDATA.2016.7840603.

9 Timo Bingmann, Patrick Dinklage, Johannes Fischer, Florian Kurpicz, Enno Ohlebusch, and Peter Sanders. Scalable text index construction. In *Algorithms for Big Data*, volume 13201 of *Lecture Notes in Computer Science*, pages 252–284. Springer, 2022. doi:10.1007/978-3-031-21534-6_14.

10 Timo Bingmann, Simon Gog, and Florian Kurpicz. Scalable construction of text indexes with thrill. In *IEEE BigData*, pages 634–643. IEEE, 2018. doi:10.1109/BIGDATA.2018.8622171.

11 Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration Inequalities – A Nonasymptotic Theory of Independence*. Oxford University Press, 2013. doi:10.1093/ACPROF:OSO/9780199535255.001.0001.

12 Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Systems Research Center, 1994.

13 Charles J Colbourn and Alan CH Ling. Quorums from difference covers. *Information Processing Letters*, 75(1-2):9–12, 2000. doi:10.1016/S0020-0190(00)00080-6.

14 Common Crawl. Common Crawl WET Files from CC-MAIN-2019-09. https://commoncrawl.org, 2019. Downloaded WET files from segments: https://data.commoncrawl.org/crawl-data/CC-MAIN-2019-09/segments/1550247479101.30/wet/CC-MAIN-20190215183319-20190215205319-#ID.warc.wet, where #ID ranges from 00000 to 000639, and https://data.commoncrawl.org/crawl-data/CC-MAIN-2019-09/segments/1550247479159.2/wet/CC-MAIN-20190215204316-20190215230316-#ID.warc.wet, where #ID ranges from 00000 to 00199. Only textual content was retained; HTML tags and Common Crawl metadata were removed.

15 Roman Dementiev, Juha Kärkkäinen, Jens Mehnert, and Peter Sanders. Better external memory suffix array construction. *ACM J. Exp. Algorithmics*, 12:3.4:1–3.4:24, 2008. doi:10.1145/1227161.1402296.

16 Michael Ferguson. ssort_chpl: Chapel-based suffix sorting module, 2025. Accessed: 2025-04-21. URL: https://github.com/femto-dev/femto/tree/main/src/ssort_chpl.

17 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *FOCS*, pages 390–398. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892127.

18 Johannes Fischer and Florian Kurpicz. Dismantling divsufsort. In *Stringology*, pages 62–76. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2017. URL: http://www.stringology.org/event/2017/p07.html.

19 Johannes Fischer and Florian Kurpicz. Lightweight distributed suffix array construction. In *ALENEX*, pages 27–38. SIAM, 2019. doi:10.1137/1.9781611975499.3.

20 Patrick Flick and Srinivas Aluru. Parallel distributed memory construction of suffix and longest common prefix arrays. In *SC*, pages 16:1–16:10. ACM, 2015. doi:10.1145/2807591.2807609.

21 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in bwt-runs bounded space. *J. ACM*, 67(1):2:1–2:54, 2020. doi:10.1145/3375890.

22 Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. New indices for text: Pat trees and pat arrays. In *Information Retrieval: Data Structures & Algorithms*, pages 66–82. Prentice-Hall, 1992.

23 Keisuke Goto. Optimal time and space construction of suffix arrays and LCP arrays for integer alphabets. In *Stringology*, pages 111–125. Czech Technical University in Prague, Faculty of Information Technology, Department of Theoretical Computer Science, 2019. URL: http://www.stringology.org/event/2019/p11.html.

24 Ilya Grebnov. libsais: A fast linear time suffix array and burrows-wheeler transform construction library. https://github.com/IlyaGrebnov/libsais, 2025. Accessed: 2025-04-19.

25 Hideo Itoh and Hozumi Tanaka. An efficient method for in memory construction of suffix arrays. In *SPIRE/CRIWG*, pages 81–88. IEEE, 1999. doi:10.1109/SPIRE.1999.796581.

26 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858.

27 Fabian Kulla and Peter Sanders. Scalable parallel suffix array construction. *Parallel Comput.*, 33(9):605–612, 2007. doi:10.1016/J.PARCO.2007.06.004.

28 Florian Kurpicz, Pascal Mehnert, Peter Sanders, and Matthias Schimek. Scalable distributed string sorting. In *ESA*, volume 308 of *LIPIcs*, pages 83:1–83:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.83.

29 Zhize Li, Jian Li, and Hongwei Huo. Optimal in-place suffix sorting. In *DCC*, page 422. IEEE, 2018. doi:10.1109/DCC.2018.00075.

30 Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In *SODA*, pages 319–327. SIAM, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320218.

31 Pascal Mehnert. Scalable distributed string sorting algorithms. Master's thesis, Karlsruhe Institute of Technology, Germany, 2024.

32 Yuta Mori. libdivsufsort. https://github.com/y-256/libdivsufsort, 2015.

33 Ge Nong, Sen Zhang, and Wai Hong Chan. Two efficient algorithms for linear time suffix array construction. *IEEE Trans. Computers*, 60(10):1471–1484, 2011. doi:10.1109/TC.2010.188.

34 Enno Ohlebusch. *Bioinformatics Algorithms: Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction*. Oldenbusch Verlag, 2013.

35 Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev. *Sequential and Parallel Algorithms and Data Structures – The Basic Toolbox*. Springer, 2019. doi:10.1007/978-3-030-25209-0.

36 Hanmao Shi and Jonathan Schaeffer. Parallel sorting by regular sampling. *J. Parallel Distributed Comput.*, 14(4):361–372, 1992. doi:10.1016/0743-7315(92)90075-X.

37 Zachary D Stephens., Skylar Y. Lee, Faraz Faghri, Roy H. Campbell, Chengxiang Zhai, Miles J. Efron, Ravishankar Iyer, Michael C. Schatz, Saurabh Sinha, and Gene E. Robinson. Big data: Astronomical or genomical? *PLOS Biology*, 13(7):1–11, July 2015. doi:10.1371/journal.pbio.1002195.

38 The 1000 Genomes Project. FASTQ Files from the 1000 Genomes Project. ftp://ftp.sra.ebi.ac.uk/vol1/fastq/DRR000/, 2025. Downloaded FASTQ files with numbers DRR000001 to DRR000439 (excluding DRR000394). Only the DNA sequence lines were retained; characters other than A, C, G, and T were removed.

39 The UniProt. FASTA Files from the Universal Protein Resource (UniProt). https://ftp.uniprot.org/pub/databases/uniprot/current_release/uniparc/fasta/active/. From https://ftp.uniprot.org/pub/databases/uniprot/current_release/uniparc/fasta/active/uniparc_active_p#ID.fasta.gz, where #ID ranges from 1 to 200. Only sequence representations were retained.

40 Tim Niklas Uhl, Matthias Schimek, Lukas Hübner, Demian Hespe, Florian Kurpicz, Daniel Seemaier, Christoph Stelz, and Peter Sanders. KaMPIng: Flexible and (near) zero-overhead C++ bindings for MPI. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '24. IEEE Press, 2024. doi:10.1109/SC41406.2024.00050.

41 Wikimedia Foundation. Wikipedia XML Dumps (March 2019) for de, en, es, fr. https://dumps.wikimedia.org/, 2019. Downloaded files available at https://dumps.wikimedia.org/#IDwiki/20190320/#IDwiki-20190320-pages-meta-current.xml.bz2, where #ID is *de*, *en*, *es*, and *fr*.

The Support of Bin Packing Is Exponential

Klaus Jansen
Department of Computer Science, Kiel University, Germany

Lis Pirotton
Department of Computer Science, Kiel University, Germany

Malte Tutas
Department of Computer Science, Kiel University, Germany

Abstract

Consider the classical Bin Packing problem with d different item sizes s_i and amounts of items a_i. The support of a Bin Packing solution is the number of differently filled bins. In this work, we show that the lower bound on the support of this problem is $2^{\Omega(d)}$. Our lower bound matches the upper bound of 2^d given by Eisenbrand and Shmonin [Oper.Research Letters '06] up to a constant factor. This result has direct implications for the time complexity of several Bin Packing algorithms, such as Goemans and Rothvoss [SODA '14], Jansen and Klein [SODA '17] and Jansen and Solis-Oba [IPCO '10].

To achieve our main result, we develop a technique to aggregate equality constrained ILPs with many constraints into an equivalent ILP with one constraint. Our technique contrasts existing aggregation techniques as we manage to integrate upper bounds on variables into the resulting constraint. We believe this technique can be useful for solving general ILPs or the d-dimensional knapsack problem.

2012 ACM Subject Classification Theory of computation → Integer programming; Theory of computation → Packing and covering problems

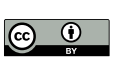
Keywords and phrases Bin Packing, Integer Programming, Support

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.48

Funding This work was supported by the German Research Foundation (DFG) – Project DFG JA 612/27-1.

1 Introduction

In their seminal work [8] Eisenbrand and Shmonin inspect integer conic combinations $b \in \mathbb{Z}^d$ of a finite set of integer vectors $X \subset \mathbb{Z}^d$. They provide upper bounds on the size of the smallest subset $X^* \subseteq X$ such that b is an integer conic combination of elements in X^*. This can be interpreted in the context of integer programming. Integer Programs (IPs) are composed of a matrix $A \in \mathbb{Z}^{d \times n}$, an (optional) cost vector $c \in \mathbb{Z}^n$, a target vector $b \in \mathbb{Z}^d$ and a solution vector $x \in \mathbb{Z}^n$. The goal is to compute the solution vector satisfying $Ax = b$ while minimizing the value $c^T x$. In this context, the set of integer vectors correspond to the columns of A and the size of the smaller subset X^* is the number of non-zero entries (the *support*) in the solution vector. More precisely, they show that for any polytope $P \subseteq \mathbb{R}^d$ and any integral vector $x \in \mathbb{Z}_{\geq 0}^{|P \cap \mathbb{Z}^d|}$ of multiplicities there exists a $x^* \in \mathbb{Z}_{\geq 0}^{|P \cap \mathbb{Z}^d|}$ such that $|\text{supp}(x^*)| \leq 2^d$ and $\sum_{p \in P \cap \mathbb{Z}^d} x_p \cdot p = \sum_{p \in P \cap \mathbb{Z}^d} x_p^* \cdot p$. They achieve this through a sophisticated exchange argument, transforming a solution with a large support into one with a bounded support. They show a second bound of $O(d \log(d\Delta))$, where $\Delta = ||A||_\infty$ is the largest number in the matrix A, in a similar fashion. These celebrated result have found application in a variety of applications. These include classical problems of computer science like scheduling on identical and uniform machines (using the second bound) [15, 19] and bin packing problem (using the first bound) [13, 18, 20].


© Klaus Jansen, Lis Pirotton, and Malte Tutas;
licensed under Creative Commons License CC-BY 4.0



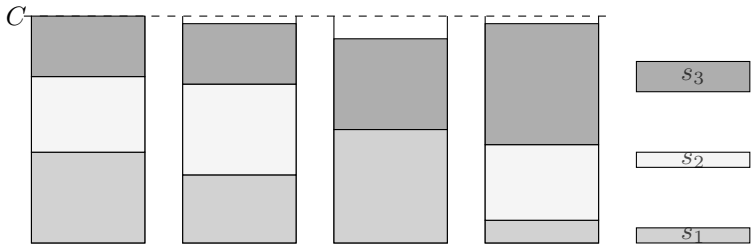


Figure 1 An illustration of the Bin Packing problem with three item types, four bins and capacity C. Colors indicate item types. Each bin has a unique *configuration*, i.e., combination of items, assigned to it. Only the leftmost bin is completely filled. The sizes of each item type are shown on the right.

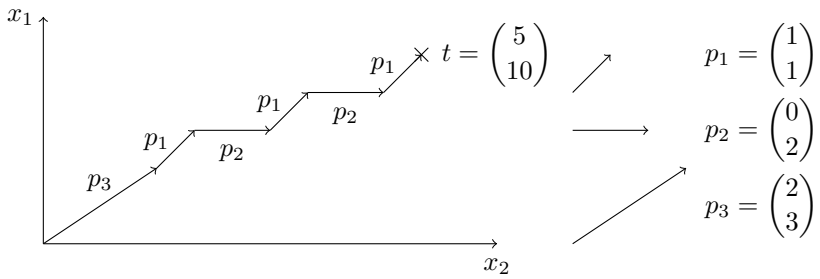


Figure 2 An illustration of the Cone and Polytope Intersection problem with three points $p_i \in P$ and the target $t \in Q$. The values of each vector are given. A possible solution is shown.

Improvements on the first bound would have direct implications for the time complexity of several algorithms. Consider the high-multiplicity Bin Packing (BP) problem with bin size C: We are given item sizes $s_1, \dots s_d \in [0, C]$, and multiplicities $a_i \in \mathbb{Z}_{\geq 0}, i \in [d]$. The task is to find the minimum number $B \in \mathbb{N}$ such that all items can be packed into B bins of size C. See Figure 1 for an illustration. For this problem, Jansen and Solis-Oba [20] give an additive approximation algorithm with guarantee OPT+1. The time complexity of their algorithm is $d^{O(d2^d)} \cdot 2^{O(8^d)} \cdot \text{poly}(enc)$, where $\text{poly}(enc)$ represents a polynomial in the input. In their paper, they ask whether the support bound by Eisenbrand and Shmonin can be improved to be polynomial in d. Consequently, this would reduce the dependency on d in the algorithm to be singly-exponential, i.e., result in a running time of $2^{\text{poly}(d)} \cdot \text{poly}(enc)$.

Following this result, Goemans and Rothvoss [13] provide an algorithm running in time $\text{enc}(s, a, C)^{2^{O(d)}}$. Here, $\text{enc}(s, a, C)$ denotes the bitsize of encoding the size vector s, the multiplicity vector a and the bin capacity C in binary. The time complexity of this algorithm would also be improved by a smaller support bound. For example, a polynomial support $\text{poly}(d)$ in d directly improves their running time to $\text{enc}(s, a, C)^{\text{poly}(d)}$. They achieve this result by solving a more general problem, the Cone and Polytope Intersection (CAPI) problem. Here, we are given two polytopes $P, Q \subseteq \mathbb{R}^d$, where P is bounded. The task is to decide whether there is a point in Q that can be expressed as a non-negative integer combination of integer points in P. Goemans and Rothvoss show that this problem captures the essence of high-multiplicity BP. They reduce BP to CAPI by setting the integer points in $P = \{\binom{x}{1} \in \mathbb{R}^{d+1}_{\geq 0} | s^T x \leq C\}$ to every possible configuration, i.e., combinations of items that fit in a single bin. They set $Q = \binom{a}{B}$ to be the point that corresponds to all items being taken. The final entry in P and Q is used to count the number of used bins B.

Jansen and Klein extend the work of Goemans and Rothvoss in [18]. They give an algorithm for bin packing that depends on the number V_I of vertices in the underlying knapsack polytope of a single bin. Their algorithm has a running time of $|V_I|^{2^{O(d)}}\text{enc}(s,a,C)^{O(1)}$. Thus, it improves upon Goemans and Rothvoss' result when V_I is small. However, it does not improve upon the result by Goemans and Rothvoss in the general sense, as the bound on V_I is $\Omega(\text{enc}(s,a,C))^d$ as given by Hayes and Larman [14].

All three of these results could be improved with a polynomial (in d) support bound for BP. However, our result shows that there is no polynomial support bound in d for bin packing, and thus answers the question posed by Jansen and Solis-Oba negatively. Goemans and Rothvoss made first strides in this direction in [13], where they show that the bound given by Eisenbrand and Shmonin is tight for polytopes, i.e., there is an exponential support when reaching a target with only column vectors or linear combinations of them. They construct a set of column vectors X with dimension d and $|X| = 2^{d-1}$. Define the polytope $P = conv(X)$, where $conv(X)$ is the convex hull of X. They then define $t = \sum_{x\in X} x$ and show that you can only reach t by taking every element of X exactly once. This shows a lower bound on the support of $2^{d-1} = 2^{\Omega(d)}$. However, their bound does not admit a knapsack constraint and can therefore not be applied to the bin packing problem directly. This is because the columns in the bin packing ILP originate from a knapsack constraint that determines the feasible packings of a single bin.

With so many different problems gaining important information from support bounds, it should come as no surprise that there are other bounds on the support of integer programs with different parameterizations. Berndt, Brinkop, Jansen, Mnich and Stamm show that the support is also bounded by $d \cdot (\log(3||A||_1) + \sqrt{\log(||A||_1)})$ [3]. They complement this result by showing a lower bound on the size of the support of $d \cdot (\lfloor \log(||A||_1 \rfloor + 1)$. Aliev, de Loera, Oertel and O'Neill [2] give two support bounds for linear diophantine equations of $r(A) + \lfloor \log(H(A)) \rfloor$, where $r(A)$ is the column rank of the matrix A and $H(A)$ is the determinant of the span of the $r(A)$-dimensional sublattice of $\mathbb{Z}^n$ formed by all integer points in the linear subspace spanned by the columns of A, i.e., $H(A) = \det(\text{span}_{\mathbb{R}}(A) \cap \mathbb{Z}^n)$. Their second bound is $\lfloor 2d \log(2\sqrt{d}\Delta) \rfloor$. Aliev, De Loera, Eisenbrand, Oertel and Weismantel [1] show the bound of $\lfloor 2d \log(2\sqrt{d}\Delta) \rfloor$ for general ILPs.

The main tool we use in the following is *aggregation.* Aggregation is a powerful tool that transforms an ILP with multiple constraints into a single constraint. Research into this technique began in the 70's, resulting in several different aggregation methods, see e.g. [4, 5, 9, 11, 12, 21, 22, 23, 24]. Elmaghraby and Wig [9] iteratively combine two equations into a single one. Applying this technique repeatedly yields a single constraint. Another approach is given by Padberg [22]. Instead of combining two equations iteratively, he aggregates all equations at the same time.

Our approach is based on the techniques by Padberg [22]. Our aggregation techniques also aggregate all constraints in a single step. However, we manage to include the upper bounds of the variables inside the resulting knapsack constraints, where Padbergs method leaves them outside the constraint. We believe that this type of aggregation can find application in different contexts, such as d-dimensional knapsack problems or general ILPs with d constraints. Here, the aggregation may yield faster algorithms, especially when one manages to reduce the size of the coefficients. This may be done by using techniques by Frank and Tardos [10], Eisenbrand, Hunkenschröder, Klein, Koutecký Levin and Onn [7] or Jansen, Kahler and Zwanger [16].

On the negative side, it is known that aggregation is not feasible for a set of inequalities, as was proven by Chvátal and Hammer [6]. They provide a simple two-line ILP that cannot be aggregated.

1.1 Our Contribution

In this work, we continue research into the support of Bin Packing solutions. We show that a set of points of Cone and Polytope Intersection can be embedded inside a knapsack polytope of dimension $O(d)$. Using this polytope, we show that the support of the high-multiplicity BP problem is exponential, requiring at least $2^{\Omega(d)}$ many different configurations in an optimum solution. This yields our main result, Theorem 1:

▶ **Theorem 1.** *There exists a high-multiplicity Bin Packing instance of dimension d such that the solution-vector x (a vector of configurations) contains at least $2^{\Omega(d)}$ non-zero entries, i.e., $|supp(x)| \in 2^{\Omega(d)}$.*

With this result, we answer an open question posed by Jansen and Solis-Oba in [20]. They provide an algorithm for the cutting stock problem. The running time of this algorithm could be improved to be singly-exponential if the support of bin packing is not exponential. However, our result answers this question negatively. In a similar vein, our result shows that the algorithm by Goemans and Rothvoss [13] can not be directly improved. The same holds for the algorithm by Jansen and Klein [17].

A major tool to achieve this result is the aggregation of multiple constraints without upper bounds on the variables into a single constraint. To the best of our knowledge, this type of constraint-aggregation has not been done before. Contrasting prior results, our method of aggregation manages to include upper bounds inside the constraints instead of copying them from the original problem. This technique is given in Section 3.

Overview

Here, we give a brief overview of the structure of the document and the structure of our main proof. We begin, in Section 3, by developing a technique with which we can aggregate an ILP containing only d equalities into one containing only a single equality. Next, in Section 4.1 we adapt an idea originally given by Goemans and Rothvoss in [13] to construct a polytope P of dimension $d' + 1$ and generate a target vector t. We provide an alternative proof to [13] that shows we require $2^d - 1$ points in P to reach t. Next, in Section 4.2, we construct an equality ILP with $O(d)$ constraints. Each constraint later defines an item size of the BP problem. We show that there are $2^d - 1$ feasible solutions to this ILP, with each solution x having the first $d + 1$ entries as vectors in P, i.e., there is one solution to the ILP for each vector in P. These solutions are the $2^d - 1$ configurations X^* of the BP problem. Then, we aggregate this ILP using the technique in Section 3. This yields an equality ILP with one row. We transform this into a knapsack constraint with right hand side C. Denote the set of all configurations with total size at most C as X. This constraint is then used to define the capacity of a bin in the BP problem. The total number of items of each type is given as the sum of all configurations in X^*. To place all these items in the optimal $2^d - 1$ bins, we know, by Section 4.1, that need to take every configuration in X^* exactly once. Taking any configuration in $X \setminus X^*$ would require at least one additional bin. This proves Theorem 1.

2 Preliminaries

For a positive integer n, we define $[n] \coloneqq \{1, 2, \ldots, n\}$ and $[n]_0 \coloneqq \{0, 1, \ldots, n\}$. A vector that contains a certain value in all entries is stylized, e.g. the zero vector 0_d is written as $\mathbb{0}_d$. We omit the dimension d if it is apparent from the context. The support of a d-dimensional vector v is the set of its indices with a non-zero entry. It is denoted by $\text{supp}(v) \coloneqq \{i \in [d] \mid v_i \neq 0\}$. We define the size of the support with $s(v) \coloneqq |\text{supp}(v)|$.

▶ **Definition 2** (Bin Packing). *Given is a set of d different item types with sizes $s_i \in (0, C]$ and amounts $a_i \in \mathbb{Z}_{\geq 0}$. Given a number $B \in \mathbb{Z}_{\geq 0}$, the goal is to decide whether all items can be assigned to one of B bins, while all bins are filled to at most C.*

A powerful tool to model countless optimization problems is that of integer programming.

▶ **Definition 3** (Integer Programming). *An integer linear program (ILP) is composed of a matrix $A \in \mathbb{Z}^{d \times n}$, an (optional) cost vector $c \in \mathbb{Z}^n$, a target vector $b \in \mathbb{Z}^d$ and a solution vector $x \in \mathbb{Z}^n_{\geq 0}$. The goal is to compute a solution vector satisfying $Ax = b$ while minimizing the value $c^T x$.*

Such integer programs can be used to express high-multiplicity Bin Packing. These are often referred to as *configuration ILPs*. A configuration is a feasible assignment $x \in \mathbb{Z}^d_{\geq 0}$ of items into a bin, i.e., one with total size at most $s^T x \leq C$. Denote the set of all feasible configurations as $\mathcal{C}$, and set $n := |\mathcal{C}|$. Set $c = \mathbb{1}_n$ and let each column of A represent a configuration, where row i corresponds to item type i. Finally set the right hand side $b := a$, i.e., the vector representing the total number of items of each size in the instance.

A useful tool that we use is the classical knapsack problem.

▶ **Definition 4** (Unbounded Knapsack). *Given a set I of n items each defined by a profit $p_i \in \mathbb{R}_{\geq 0}$ and a weight $w_i \in \mathbb{Z}_{\geq 1}$, along with a knapsack of capacity $C \in \mathbb{Z}_{\geq 1}$. Find a solution vector $x \in \mathbb{Z}^n_{\geq 0}$ such that $\sum_{i=0}^n w_i x_i \leq C$ and $\sum_{i=0}^n p_i x_i$ is maximal. We call $\sum_{i=0}^n w_i x_i = C$ the equality Knapsack constraint to the corresponding problem.*

Following this definition, we call the set of all solutions fulfilling $x_i \geq 0$, $\forall i \in [n]$ and $\sum_{i=0}^n w_i x_i \leq C$ the Knapsack Polytope.

3 Aggregation of an ILP

In this section, we show how to transform an ILP with multiple equality constraints into a single knapsack constraint. The novel aspect of our technique is that we also include the external upper bounds of the variables in the aggregation.

Consider an ILP of the form $\min\{c^T x \mid Ax = b, x \in \mathbb{Z}^n_{\geq 0}, x \leq u\}$ with $A = (a_{ij})_{i \in [d], j \in [n]} \in \mathbb{Z}^{d \times n}, c \in \mathbb{Z}^n, u \in \mathbb{Z}^n_{\geq 0}$ and $b \in \mathbb{Z}^d$ and let $\Delta := \|A\|_\infty$ be the largest absolute value in A.

For each variable $x_j, j \in [n]$, we define the following constraint: $x_j + s_j = u_j$, where $s_j \in \mathbb{Z}_{\geq 0}$ is a slack variable. This simulates the upper bound $x_j \leq u_j$. Define $U := \sum_{j=1}^n u_j$ as the overall upper bound on the sum of all variables. For this, add the constraint:

$$\sum_{i=j}^{n} (x_j + s_j) + s_{n+1} = U, \tag{1}$$

with $s_{n+1} \in \mathbb{Z}_{\geq 0}$ as a slack variable. Note that bounding the sum of variables by the sum of their upper bounds does not change the solution.

This results in a new ILP $A' \begin{pmatrix} x \\ s \end{pmatrix} = (b,\ u,\ U)$, where

$$A' := \begin{pmatrix} A & 0_{dn} & 0 \\ I_n & I_n & 0 \\ \mathbb{1}_n & \mathbb{1}_n & 1 \end{pmatrix} \in \mathbb{Z}^{(d+n+1) \times (2n+1)}.$$

Next, we use the bound U to define a large integer M, which we then use to aggregate the constraints.

$$\begin{aligned} M &:= \Delta U + \max(\|b\|_\infty, \|u\|_\infty) + \Delta + 2 \\ &> \Delta U + \max(\|b\|_\infty, \|u\|_\infty). \end{aligned} \tag{2}$$

By giving the upper-bound constraint (1) the largest weight M^{d+n}, we ensure that it may never be broken and thus, keeps the sum of the variables in its range. Multiplying both sides of the equations with $1, M, M^2, \ldots, M^{d+n}$ generates the following ILP where all variables have to be non-negative integers.

$$\begin{aligned} \textstyle\sum_{j=1}^{n} a_{1j}x_j &= b_1 \\ M\textstyle\sum_{j=1}^{n} a_{2j}x_j &= Mb_2 \\ &\vdots \\ M^{d-1}\textstyle\sum_{j=1}^{n} a_{mj}x_j &= M^{d-1}b_m \\ M^d(x_1 + s_1) &= M^d u_1 \\ &\vdots \\ M^{d+n-1}(x_n + s_n) &= M^{d+n-1}u_n \\ M^{d+n}(\textstyle\sum_{j=1}^{n}(x_j + s_j) + s_{n+1}) &= M^{d+n}U \\ x_j &\in \mathbb{Z}_{\geq 0} \quad \forall j \in [n] \\ s_j &\in \mathbb{Z}_{\geq 0} \quad \forall j \in [n+1] \end{aligned} \tag{3}$$

These constraints can also be formulated as $A' \begin{pmatrix} x \\ s \end{pmatrix} = (b,\ u,\ U)$, where

$$A' := \begin{pmatrix} A & 0_{dn} & 0 \\ I_n & I_n & 0 \\ \mathbb{1}_n & \mathbb{1}_n & 1 \end{pmatrix} \in \mathbb{Z}^{(d+n+1)\times(2n+1)}$$

An intuitive view of the aggregation technique is that the resulting single constraints solution is a base M number, where M is a large integer. There, the smallest digit represents the solution of the first constraint, the second constraints solution is represented by the second smallest digit and so on.

Next, we show that this ILP can be aggregated into a single equality knapsack constraint by proving that their solutions are identical. The proof is presented in the full version.

▶ **Lemma 5** (✂). *The vector* $\texttt{sol} = (x,\ s)^T \in \mathbb{Z}_{\geq 0}^{2n+1}$ *is a feasible integer solution to* (3) *if and only if* `sol` *is an integer solution to the following equation:*

$$\begin{aligned} \sum_{i=1}^{d}\big(M^{i-1}\sum_{j=1}^{n}(a_{ij}x_j)\big) + \sum_{j=1}^{n}\big(M^{d+j-1}(x_j + s_j)\big) + M^{d+n}\big(\sum_{j=1}^{n}(x_j + s_j) + s_{n+1}\big) \\ = \sum_{i=1}^{d}(M^{i-1}b_i) + \sum_{j=1}^{n}\big(M^{d+j-1}u_j\big) + M^{d+n}U \end{aligned} \tag{4}$$

4 Bounding the Support for Bin Packing

Assume we are given a knapsack polytope P of dimension $d' + 1$. Also assume that there exists a Bin Packing instance $\mathcal{I}$ with just one unique integer solution $x \in \mathbb{Z}_{\geq 0}^{d}$, $d = O(d')$ to the configIP and the $d' + 1$-dimensional integer points in P are equal to the first $d' + 1$

dimensions of the configurations. If we now require k different points in P to represent a vector t as a conic integer combination then x has at least k non-zero entries, which means that we need at least k different configurations in the Bin Packing solution. If now k is exponential in d', we have shown that there exists an exponential lower bound on the support of the Bin Packing problem, i.e., Theorem 1.

4.1 The support of closed polytopes

In this section, we give a knapsack polytope P of dimension $d+1$ and show that there exists a vector $t \in \mathbb{Z}^{d+1}$ such that we need at least $2^d - 1$ points in P to represent t as a conic integer combination. We adapt a construction pioneered by Goemans and Rothvoss in [13] and set

$$\begin{aligned} X &= \{\mathbb{0}\} \cup \{(x_1, \dots, x_d, (4k)^{\sum_{i=0}^{d-1} 2^i x_i}) \mid x_i \in \{0,1\} \forall i \in [d]\} \setminus \{(0, \dots, 0, 1)\}\} \\ &= \{a_0, a_1, \dots, a_k\} \end{aligned} \tag{5}$$

with $k = 2^d - 1$ and define $P := \text{conv}(X)$. E.g. for $d = 2$, we have

$$X = \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 4k \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ (4k)^2 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ (4k)^3 \end{pmatrix} \right\}.$$

Assume that the elements in X are sorted according to their length, i.e., $\|a_0\|_\infty = 0$ and $\|a_i\|_\infty = (4k)^i$ for all $i \in [k]$. Then the following lemma proves the goal of this section.

▶ **Lemma 6.** *Set $t = \sum_{i=0}^{k} a_i$. The conic combination $t = \sum_{i=1}^{k} a_i$ is unique (except for arbitrarily adding a_0). This implies that we need all $2^d - 1$ points in P exactly once to represent t as a conic integer combination.*

Proof. Each integer conic combination that represents t and includes $\mathbb{0}$ can be transformed into an integer conic combination of less points by removing $\mathbb{0}$. Therefore, we can assume that $\mathbb{0}$ is not part of the conic combination.

Now considering P, the only integer points in P are the elements in X, i.e., $P \cap \mathbb{Z}^{d+1} = X$. This holds due to the first d coordinates. For all $\mu_i \in (0,1)$ with $\sum_i \mu_i = 1$ it holds that $\mu_i \cdot 0 + \mu_j \cdot 1 \notin \mathbb{Z}$ and for each pair of vectors $(a_i, a_j) \in X$ with $i \neq j$, there is at least one coordinate ℓ, where with $a_i(\ell) = 0$ and $a_j(\ell) = 1$ or $a_i(\ell) = 1$ and $a_j(\ell) = 0$. Thus, there can not be any integer points other than X in $P \cap \mathbb{Z}^{d+1}$.

Now consider an arbitrary integer conic combination $\sum_{i=1}^{k} \mu_i a_i = t = \begin{pmatrix} 2^{d-1} \\ \vdots \\ 2^{d-1} \\ \sum_{i=1}^{k} (4k)^i \end{pmatrix}$.

Since $t_1 = \dots = t_{d-1} = 2^{d-1}$ and we do not include $\mathbb{0}$, we can not take a single vector more than $2^{d-1} \leq k$ times. Furthermore, we can not take more than dk vectors total, as each vector except $\mathbb{0}$ has an entry in at least one of the d dimensions, and each entry can only sum up to k. This leaves us with the following:

$$0 \leq \sum_{i=1}^{k} \mu_i \leq d \cdot 2^{d-1} \leq dk, \quad \text{and}$$
$$\mu_i \leq 2^{d-1} \leq k \quad \forall i \in [k]$$

i.e., that at most dk vectors can be summed up to reach t and each vector is taken $\leq k$ times.

We now show that $\mu_i = 1$ for all $i \in [k]$ is the only solution to $t = \sum_{i=1}^{k} \mu_i a_i$. First, observe that $t_{d+1} = \sum_{i=1}^{k} (4k)^i < 2(4k)^k$ holds. Therefore, we have $\mu_k < 2$. Next, observe that $k(4k)^{x-1} < (4k)^x - (4k)^{x-1} \Leftrightarrow (k+1)(4k)^{x-1} < (4k)^x$ holds for all $x \in \mathbb{Z}_{\geq 1}$ because $(k+1) < 4k$. Because of this, even after adding the maximum possible k copies of the $k-1$-th item, we have that $t_{d+1} - k(4k)^{k-1} > (4k)^{k-1}$. We can only compensate this difference by adding smaller items. Here, we can apply the argument iteratively, as we can only add k copies of column $(k-2)$ to sum up to a value $> (4k)^{k-1}$. Thus, $\sum_{i=1}^{k-1} k(4k)^i < (4k)^k$ holds. Therefore, we can not sum up to $(4k)^k$ without the last item and $\mu_k > 0$ directly follows. Taking $\mu_k < 2$ and $\mu_k > 0$ together, we have $\mu_k = 1$. Because of $\mu_k = 1$, we have $t_{d+1} = \sum_{i=1}^{k} -1(4k)^i$, and we only need to consider $\mu_i, i \in [k-1]$. Then, this argument can be applied iteratively, yielding $\mu_i = 1$ for all $i \in [k]$. ◀

By Lemma 6, we have shown that we require $2^d - 1$ different points in P to represent a vector t as a conic integer combination. In the following we build the bridge to Bin Packing and prove that there exists a Bin Packing instance where the first $d+1$ dimensions of the solution vector of its configIP equals the integer points in P. In total, our Bin Packing instance has $O(d)$ dimensions which then with Lemma 6 implies the exponential lower bound of Bin Packing and therefore Theorem 1.

4.2 Construction of the ILP

In this section, we now aim to construct an ILP with equality constraints, where the first $d+1$ coordinates in its solution vectors are unique and form the following set:

$$\{(x_0^{\texttt{bin}}, \dots, x_{d-1}^{\texttt{bin}}, \gamma^{\sum_{\ell=0}^{d-1} 2^\ell x_\ell^{\texttt{bin}}}) \mid x_\ell^{\texttt{bin}} \in \{0,1\} \forall \ell \in [d-1]_0\} \setminus \{(0, \dots, 0, 1)\}\}. \tag{6}$$

Note that this is exactly the set X of points in P in the prior section without $a_0 = \mathbb{0}$ and where $\gamma = 4k$ and that the solution vectors form the configurations in the configIP.

Also note that the last coordinate is of the form γ^i for some $i \in \mathbb{Z}_{\geq 1}$ and the first d coordinates are the binary encoding of i. We first define a variable $r(d)$, which we use to construct a set of constraints that is only feasible if $r(d) = \gamma^i$ holds for some $i \in \mathbb{Z}_{\geq 1}$. Additionally, a certain set of variables in the constraints matches the points in Equation (6). In second part of this section, we use the aggregation presented in Section 3 to transform these constraints into a single knapsack constraint. We provide a proof that the knapsack constraint is indeed equivalent to the proposed set of constraints (i.e., both allow the same set of solutions) and requires $O(d)$ many variables. With this equivalence, we then have that the first $d+1$ dimensions of the configurations in the configIP form the set Equation (6). In combination with the Lemma 6, this then implies that there exists an exponential lower bound on the support of the Bin Packing problem.

Next, assume that we are given a dimension $d \in \mathbb{Z}_{\geq 1}$ and a base $\gamma \in \mathbb{Z}_{\geq 2}$. We introduce a set of constraints with $O(d)$ variables which has exactly $2^d - 1$ integer solutions. For each $i \in [2^d - 1]$, the variable $r(d)$ of the solution vector is forced to take the value γ^i. Additionally, the variables $x^{\texttt{bin}}(\ell) \in \{0,1\}, \ell \in [d-1]_0$ match the binary encoding of i, i.e., $i = \sum_{\ell=0}^{d-1} 2^\ell x^{\texttt{bin}}(\ell)$.

For a brief overview, our constraints behave as follows. First, the value $r(d)$ is some integer in the interval $[2, \gamma^{2^d-1}]$. Let us assume that $r(d) = \gamma^i$ for some i. Later, we show that we only allow those integers that equal γ^i where i is a positive integer. Next, for each $\ell = d-1, \dots, 0$, we introduce the variable $y(\ell)$ that helps us to correctly determine the bit $x^{\texttt{bin}}(\ell)$ of the binary encoding. We also introduce the variable $z(\ell)$ that helps us to determine

the $r(\ell)$ which will then be used recursively. For a more detailed version of the procedure of converting a decimal number $i \in \mathbb{Z}_{\geq 1}$ to binary, we refer to Algorithm 1. Note that the number we aim to convert is in the exponent of γ.

Algorithm 1 Decimal to Binary.

Input: Dimension $d \in \mathbb{Z}_{\geq 1}$; Base $\gamma \in \mathbb{Z}_{\geq 2}$; Exponent $i \in \mathbb{Z}_{\geq 1}$
Output: Binary encoding $x^{\texttt{bin}}$ of i

Set $r(d) \coloneqq \gamma^i$ with $i \in \mathbb{Z}_{\geq 1}$ and $\gamma^i \leq \gamma^{2^d-1}$.
for $\ell = d-1, \dots, 0$ **do**
  if $r(\ell+1) \geq \gamma^{2^\ell}$ **then**
    Record a 1 in position ℓ, i.e., $x^{\texttt{bin}}(\ell) \coloneqq 1$.
    Subtract 2^ℓ from the current exponent, i.e., $r(\ell) \coloneqq \frac{r(\ell+1)}{\gamma^{2^\ell}}$.
  else
    Record a 0 in position ℓ, i.e., $x^{\texttt{bin}}(\ell) \coloneqq 0$.
    Do not change the current exponent, i.e., $r(\ell) \coloneqq r(\ell+1)$.
return the vector $x^{\texttt{bin}}(\ell), \ell \in [d-1]_0$

The following set of constraints simulates this algorithm. Let α be some integer number larger than 3. Later, α represents the total number of constraints. Each set of constraints $\texttt{C}_{\ell,k}, k \in [13]$ is defined for all $\ell \in [d-1]_0$. They simulate the algorithm above. Additionally, $\texttt{C}_{\alpha-2}$ bounds $r(d)$, i.e., ensures its binary encoding has at most d bits. $\texttt{C}_0$ guarantees that in the end, there is no remainder left. This constraint is also needed to guarantee that the exponent i is integer (Lemma 12). Also, we do not want to allow $i = 0$, respectively $x^{\texttt{bin}}(\ell) \neq 0$. That is done by $\texttt{C}_{\alpha-3}$. Thus, in total, we have $13d + 3$ constraints.

▶ **Constraints 1.** *For each $\ell \in [d-1]_0$, we have the following set of constraints:*

$$
\begin{aligned}
y(\ell) &\geq 0 && (C_{\ell,1})\\
y(\ell) &\leq (r(\ell+1)/\gamma^{2^\ell}) + 1/(\gamma^{2^\ell}+1) && (C_{\ell,2})\\
y(\ell) &\geq (r(\ell+1)/\gamma^{2^\ell}) - (\gamma^{2^\ell}-1)/\gamma^{2^\ell} && (C_{\ell,3})\\
x^{bin}(\ell) &\geq 0 && (C_{\ell,4})\\
x^{bin}(\ell) &\leq 1 && (C_{\ell,5})\\
x^{bin}(\ell) &\leq y(\ell) && (C_{\ell,6})\\
y(\ell) &\leq (\gamma^{2^\ell}+1)x^{bin}(\ell) && (C_{\ell,7})\\
r(\ell) &\geq 0 && (C_{\ell,8})\\
(\gamma^{2^\ell}-1)z(\ell) + r(\ell) &= r(\ell+1) && (C_{\ell,9})\\
z(\ell) &\geq 0 && (C_{\ell,10})\\
z(\ell) &\geq -\gamma^{2^\ell} + \gamma^{2^\ell}x^{bin}(\ell) + r(\ell) && (C_{\ell,11})\\
z(\ell) &\leq \gamma^{2^\ell}x^{bin}(\ell) && (C_{\ell,12})\\
z(\ell) &\leq r(\ell) && (C_{\ell,13})
\end{aligned}
$$

Also, we add the additional constraints $C_0, C_{\alpha-3}$ and $C_{\alpha-2}$

$$
\begin{aligned}
r(0) &= 1 && (C_0)\\
r(d) &\geq 2 && (C_{\alpha-3})\\
r(d) &\leq \gamma^{2^d-1} && (C_{\alpha-2})
\end{aligned}
$$

All of the constraints are necessary to ensure that $r(d)$ takes some value of the form $\gamma^i, i \in \mathbb{Z}_{\geq 1}$, while $x^{\texttt{bin}}$ is the binary encoding of i. Moreover, the constraints forbid any other assignment of $r(d)$ and $x^{\texttt{bin}}$. All other variables are forced to take a specific value based on i. In the full version, we give an example that we cannot omit the variables $z(\ell)$, as that would allow other solutions with $r(d) = \gamma^i, i \in \mathbb{Z}_{\geq 1}$.

For easier understanding, we now present an example.

▶ **Example 7.** Let $\gamma \in \mathbb{Z}_{\geq 2}$ and set $d := 3$. Then, for $i \in [7]$, we get the following values for the variables.

i	1	2	3	4	5	6	7
$r(3)$	γ^1	γ^2	γ^3	γ^4	γ^5	γ^6	γ^7
$y(2)$	0	0	0	1	γ^1	γ^2	γ^3
$y(1)$	0	1	γ^1	0	0	1	γ^1
$y(0)$	1	0	1	0	1	0	1
$x^{\mathtt{bin}}(2)$	0	0	0	1	1	1	1
$x^{\mathtt{bin}}(1)$	0	1	1	0	0	1	1
$x^{\mathtt{bin}}(0)$	1	0	1	0	1	0	1
$r(2)$	γ^1	γ^2	γ^3	$\gamma^{4-4} = 1$	$\gamma^{5-4} = \gamma^1$	$\gamma^{6-4} = \gamma^2$	$\gamma^{7-4} = \gamma^3$
$r(1)$	γ^1	$\gamma^{2-2} = 1$	$\gamma^{3-2} = \gamma^1$	1	γ^1	$\gamma^{2-2} = 1$	$\gamma^{3-2} = \gamma^1$
$r(0)$	$\gamma^{1-1} = 1$	1	$\gamma^{1-1} = 1$	1	$\gamma^{1-1} = 1$	1	$\gamma^{1-1} = 1$
$z(2)$	0	0	0	1	γ^1	γ^2	γ^3
$z(1)$	0	1	γ^1	0	0	1	γ^1
$z(0)$	1	0	1	0	1	0	1

Now, we state properties for the variables $y(\ell), x^{\mathtt{bin}}(\ell), r(\ell), z(\ell)$ based on the value of $r(\ell+1)$. After that, we prove that the vector $x^{\mathtt{bin}}$ is indeed the binary encoding of i.

$\mathtt{C}_{\ell,1}, \mathtt{C}_{\ell,4}, \mathtt{C}_{\ell,8}$ and $\mathtt{C}_{\ell,10}$ are lower bounds of the variables $y(\ell), x^{\mathtt{bin}}(\ell), r(\ell)$ and $z(\ell)$. We omit these constraints later on by restricting all variables of the constructed ILP to be non-negative.

$\mathtt{C}_{\ell,1}$ to $\mathtt{C}_{\ell,3}$ ensure that the currently most significant bit is assigned correctly. As the values of $y(\ell)$ are assigned from the most significant bit to the least significant bit, this ensures the correctness of the binary encoding. More specifically, we have the following property.

▶ **Observation 8.** *Constraints $\mathtt{C}_{\ell,1}$ to $\mathtt{C}_{\ell,3}$ imply*

$$y(\ell) \begin{cases} = 0, & \text{if } r(\ell+1) < \gamma^{2^\ell} \\ \geq 1 & \text{if } r(\ell+1) \geq \gamma^{2^\ell} \end{cases}$$

Proof. We show this by analyzing the cases separately:

Case 1: Assume $r(\ell+1) < \gamma^{2^\ell}$. $\mathtt{C}_{\ell,2}$ states $y(\ell) \leq {}^{r(\ell+1)}/_{\gamma^{2^\ell}} + {}^{1}/_{(\gamma^{2^\ell}+1)}$. With the assumption and the fact that $r(\ell+1)$ is integer, we get $y(\ell) \leq {}^{(\gamma^{2^\ell}-1)}/_{\gamma^{2^\ell}} + {}^{1}/_{(\gamma^{2^\ell}+1)}$. Now, ${}^{1}/_{(\gamma^{2^\ell}+1)} < {}^{1}/_{\gamma^{2^\ell}}$ implies $y(\ell) < {}^{\gamma^{2^\ell}}/_{\gamma^{2^\ell}} = 1$. Since $y(\ell)$ is also restricted to integer values we get $y(\ell) = 0$ with $\mathtt{C}_{\ell,1}$. Now, we only need to show that the other constraint is also fulfilled for $y(\ell) = 0$. For $\mathtt{C}_{\ell,3}$ we get $0 = y(\ell) \geq {}^{r(\ell+1)}/_{\gamma^{2^\ell}} - {}^{(\gamma^{2^\ell}-1)}/_{\gamma^{2^\ell}}$, since $r(\ell+1) \leq \gamma^{2^\ell} - 1$ which completes this part of the proof.

Case 2: Assume $r(\ell+1) \geq \gamma^{2^\ell}$. $\mathtt{C}_{\ell,3}$ states $y(\ell) \geq {}^{r(\ell+1)}/_{\gamma^{2^\ell}} - {}^{(\gamma^{2^\ell}-1)}/_{\gamma^{2^\ell}}$. With the assumption, we get $y(\ell) \geq 1 - {}^{(\gamma^{2^\ell}-1)}/_{\gamma^{2^\ell}} > 0$. Since $y(\ell)$ is restricted to integer values we get $y(\ell) \geq 1$. Now, we only need to show that the other constraints are also fulfilled for $y(\ell)$. This clearly holds for $\mathtt{C}_{\ell,1}$. For $\mathtt{C}_{\ell,2}$ we get $1 \leq y(\ell) \leq {}^{r(\ell+1)}/_{\gamma^{2^\ell}} + {}^{1}/_{(\gamma^{2^\ell}+1)}$, since ${}^{r(\ell+1)}/_{\gamma^{2^\ell}} \geq 1$ and ${}^{1}/_{(\gamma^{2^\ell}+1)} \leq 1$. This finalizes the proof. ◀

In particular, we have that the value of $y(\ell) = \lfloor {}^{r(\ell+1)}/_{\gamma^{2^\ell}} + {}^{1}/_{(\gamma^{2^\ell}+1)} \rfloor$ is unique for given $r(\ell+1)$. This is due to the fact that the feasible interval (given by $\mathtt{C}_{\ell,2}$ and $\mathtt{C}_{\ell,3)}$) is of size $1 - {}^{1}/_{\gamma^{2^\ell}} \leq {}^{1}/_{(\gamma^{2^\ell}+1)} + {}^{(\gamma^{2^\ell}-1)}/_{\gamma^{2^\ell}} < 1$. The second inequality implies that there exists at most one integer value in the interval. The first inequality in combination with the fact that $r(\ell+1)$ is integer, we also have that there must exist an integer value in the interval.

Now, we are able to show that the $x^{\mathtt{bin}}(\ell), \ell \in [d-1]_0$ matches the binary encoding of i. $\mathtt{C}_{\ell,6}$ ensures that $x^{\mathtt{bin}}(\ell) = 0$ if $y(\ell) = 0$, while $\mathtt{C}_{\ell,7}$ provides that $x^{\mathtt{bin}}(\ell) > 0$ if $y(\ell) > 0$. Taken together with $\mathtt{C}_{\ell,5}$ and the fact that $x^{\mathtt{bin}}(\ell)$ is integer, we observe the following.

▶ **Observation 9.** *Constraints $\mathtt{C}_{\ell,1}$ to $\mathtt{C}_{\ell,7}$ imply*

$$x^{\mathtt{bin}}(\ell) = \begin{cases} 0, & \text{if } r(\ell+1) < \gamma^{2^\ell} \\ 1, & \text{if } r(\ell+1) \geq \gamma^{2^\ell} \end{cases}$$

Proof. We show this by again separately analyzing the cases:

Case 1: Assume $r(\ell+1) < \gamma^{2^\ell}$. With Observation 8, we know $y(\ell) = 0$. $\mathtt{C}_{\ell,6}$ now implies $x^{\mathtt{bin}}(\ell) = 0$. This also fulfills $\mathtt{C}_{\ell,4}, \mathtt{C}_{\ell,5}$ and $\mathtt{C}_{\ell,7}$.

Case 2: Assume $r(\ell+1) \geq \gamma^{2^\ell}$. Observation 8 and $\mathtt{C}_{\ell,7}$ now imply $x^{\mathtt{bin}}(\ell) \geq 1$. With $\mathtt{C}_{\ell,5}$ we get $x^{\mathtt{bin}}(\ell) = 1$. This also fulfills $\mathtt{C}_{\ell,4}$ and $\mathtt{C}_{\ell,6}$ and finalizes the proof.

Note that this also upper bounds the value of $y(\ell)$. Since $x^{\mathtt{bin}}(\ell) \leq 1$, $\mathtt{C}_{\ell,7}$ implies $y(\ell) \leq \gamma^{2^\ell} + 1$. ◀

And finally, $\mathtt{C}_{\ell,10}$ to $\mathtt{C}_{\ell,13}$ represent the remainder calculation and ensure that the constraints can be used recursively. More concretely, we have the following behavior.

▶ **Observation 10 (✂).** *Constraints $\mathtt{C}_{\ell,8}$ to $\mathtt{C}_{\ell,13}$ imply*

$$r(\ell) = \begin{cases} r(\ell+1), & \text{if } r(\ell+1) < \gamma^{2^\ell} \\ {}^{r(\ell+1)}/_{\gamma^{2^\ell}}, & \text{if } r(\ell+1) \geq \gamma^{2^\ell} \end{cases}$$

Also, for each $\ell \in [d]_0$, the remainder is bounded by $r(\ell) \leq \gamma^{2^\ell} - 1$.

Proof Outline. We prove this statement via case distinction and induction over ℓ. Due to space concerns the proof can be found in the full version. ◀

Now, we prove that if the Constraints 1 are feasible, then $r(d)$ is of the form $\gamma^i, i \in \mathbb{Z}_{\geq 1}$ and $x^{\mathtt{bin}}$ is the binary encoding of i. We also show that the constraints do not permit any other integer solutions.

▶ **Lemma 11.** *Let $d \in \mathbb{Z}_{\geq 1}$ and $\gamma \in \mathbb{Z}_{\geq 2}$. If there exists an $i \in \mathbb{Z}_{\geq 1}$ such that $r(d) = \gamma^i, i \in \mathbb{Z}_{\geq 1}$, with $\gamma^i \leq \gamma^{2^d-1}$, then $x^{\mathtt{bin}}$ is the binary encoding of i.*

Proof. Assume, there exists an $i \in \mathbb{Z}_{\geq 1}$ such that $r(d) = \gamma^i$ and $\gamma^i \leq \gamma^{2^d-1}$. We prove that $x^{\mathtt{bin}}$ is the binary encoding of i by induction over $\ell = d-1, \ldots, 0$.

Base Case: By the assumption $r(d) = \gamma^i$, we have that $r(d)$ is correctly set.

Inductive Step: Let $\ell \in [d-1]$ and assume that $r(\ell)$ is the correct remainder in the ℓ-th iteration. Following the procedure described at the beginning of this section, we know that $x^{\mathtt{bin}}(\ell-1)$ is required to be 1 if $r(\ell) \geq \gamma^{2^{\ell-1}}$ and 0 otherwise. This property holds due to Observations 8 and 9. By Observation 10, we know that the next remainder $r(\ell-1)$ is correctly set to ${}^{r(\ell)}/_{\gamma^{2^{\ell-1}}}$ if $r(\ell) \geq \gamma^{2^{\ell-1}}$ and to $r(\ell)$ otherwise. ◀

Now, we know that $x^{\texttt{bin}}$ is the binary encoding of i, if there exists $i \in \mathbb{Z}_{\geq 1}$ with $\gamma^i \leq \gamma^{2^d-1}$ and $r(d) = \gamma^i$. To finalize that the constraints fulfill the desired property, we need to show that the constraints are infeasible for any other value of $r(d)$.

▶ **Lemma 12.** *Let $d \in \mathbb{Z}_{\geq 1}$ and $\gamma \in \mathbb{Z}_{\geq 2}$. If $r(d) = \gamma^i$ for some $i \notin \mathbb{Z}_{\geq 1}$ holds, then the Constraints 1 are infeasible.*

Proof. Assume $r(d) = \gamma^i$ for some $i \notin \mathbb{Z}_{\geq 1}$. Note that we can formulate any integer number k in the form $\gamma^{\log k / \log \gamma}$. The proof is split into different cases. For each case, we lead the constraints to a contradiction.

Case 1: First, we show that the set of constraints is infeasible when i is negative. Let $d \in \mathbb{Z}_{\geq 1}$ and $\gamma \in \mathbb{Z}_{\geq 2}$. Set $j \coloneqq -i$. By the assumption, we have $r(d) = \gamma^i = 1/\gamma^j$. This contradicts the constraint that $r(d)$ is required to be integer.

Case 2: We now show that the set of constraints is infeasible when i is not integer. Let $\gamma \in \mathbb{Z}_{\geq 2}$ and $i \notin \mathbb{Z}_{\geq 0}$. Again set $r(d) = \gamma^i$. We now show by induction that the exponent $p(\ell)$ of γ is not integer for all $\ell \in [d]$. This then leads to a contradiction. This contradiction appears either at the additional constraint $r(0) = 1$ or already at constraint $\mathtt{C}_{k,9}$ for some $k \in [d-1]_0$.

Base Case: By definition of i in this case, i is not integer and with the assumption $r(d) = \gamma^i = \gamma^{p(d)}$ we get $i = p(d)$ and therefore $p(d)$ is not integer. Note that the assumption that the value $\gamma^{p(d)}$ is assigned to the variable $r(d)$ implies that $\gamma^{p(d)}$ must be integer.

Inductive Step: Let $\ell \in [d]$ and assume $p(\ell)$ is not integer and $r(\ell) = \gamma^{p(\ell)}$ is integer. Now we can meet the following two cases. Either, $r(\ell) < \gamma^{2^{\ell-1}}$ or $r(\ell) \geq \gamma^{2^{\ell-1}}$. In the first case, Observation 10 gives $r(\ell-1) = r(\ell)$ which directly implies $p(\ell-1) = p(\ell)$. Thus, $p(\ell-1)$ is not integer and $r(\ell-1)$ is integer. In the second case, with Observation 10 we get $r(\ell-1) = r(\ell)/\gamma^{2^{\ell-1}} = \gamma^{p(\ell)}/\gamma^{2^{\ell-1}} = \gamma^{p(\ell)-2^{\ell-1}}$. Since $p(\ell)$ is not integer and $2^{\ell-1}$ is integer, we get that $p(\ell-1) = p(\ell) - 2^{\ell-1}$ is not integer. If now $r(\ell-1) = \gamma^{p(\ell-1)}$ is integer, we repeat the inductive step. However, if $r(\ell-1) = \gamma^{p(\ell-1)}$ is not integer, there is no feasible integer assignment for $r(\ell-1)$ as $\mathtt{C}_{\ell,9}$ is an equality constraint.

If $r(\ell)$ is integer for all $\ell \in [d]$, we get that $p(0)$ is not integer. This implies $r(0) = \gamma^{p(0)}$. This is a contradiction to $r(0) = 1$.

Case 3: The only case left is $\gamma^i = 1$, i.e., $i = 0$. Again, by the assumption, we get $r(d) = \gamma^i = 1$ which directly contradicts $\mathtt{C}_{\alpha-3}$ which states $r(d) \geq 2$. ◀

Thus, we have shown that the Constraints 1 are feasible if and only if variable $r(d)$ takes a value of the form γ^i for given $\gamma \in \mathbb{Z}_{\geq 2}$ and positive integer i (Lemma 12). Additionally, $x^{\texttt{bin}}$ matches the binary encoding of i, i.e., it holds that $i = \sum_{\ell=0}^{d-1} 2^\ell x^{\texttt{bin}}(\ell)$ (Lemma 11).

The attentive reader might have noticed that in Example 7 the values of $y(\ell)$ and $z(\ell)$ match for all values of i and ℓ. In the full version, we show that we can not simply replace $z(\ell)$ with $y(\ell)$ in $\mathtt{C}_{\ell,9}$.

4.3 Aggregation of the Constraints

Before we can aggregate the constraints into a single knapsack constraint, we need to transform the inequalities into equations such that we achieve an ILP of the form considered in Section 3. We can do that by introducing slack variables. Also, we omit the lower bounds $\mathtt{C}_{\ell,1}, \mathtt{C}_{\ell,4}, \mathtt{C}_{\ell,8}$ and $\mathtt{C}_{\ell,10}$ by restricting all variables to only take non-negative values. This results in the following new set of constraints:

▶ **Constraints 2.**

$$\begin{aligned}
r(0) &= 1 && \\
(\gamma^{2^{\ell+1}}+\gamma^{2^\ell})y(\ell)-(\gamma^{2^\ell}+1)r(\ell+1)+s_1(\ell) &= \gamma^{2^\ell} && \forall \ell \in [d-1]_0\\
-\gamma^{2^\ell}y(\ell)+r(\ell+1)+s_2(\ell) &= \gamma^{2^\ell}-1 && \forall \ell \in [d-1]_0\\
x^{\texttt{bin}}(\ell)+s_3(\ell) &= 1 && \forall \ell \in [d-1]_0\\
x^{\texttt{bin}}(\ell)-y(\ell)+s_4(\ell) &= 0 && \forall \ell \in [d-1]_0\\
y(\ell)-(\gamma^{2^\ell}+1)x^{\texttt{bin}}(\ell)+s_5(\ell) &= 0 && \forall \ell \in [d-1]_0\\
(\gamma^{2^\ell}-1)z(\ell)+r(\ell)-r(\ell+1) &= 0 && \forall \ell \in [d-1]_0\\
\gamma^{2^\ell}x^{\texttt{bin}}(\ell)-z(\ell)+r(\ell)+s_7(\ell) &= \gamma^{2^\ell} && \forall \ell \in [d-1]_0\\
-\gamma^{2^\ell}x^{\texttt{bin}}(\ell)+z(\ell)+s_8(\ell) &= 0 && \forall \ell \in [d-1]_0\\
z(\ell)-r(\ell)+s_9(\ell) &= 0 && \forall \ell \in [d-1]_0\\
r(d)-s_{\alpha-3} &= 2 && \\
r(d)+s_{\alpha-2} &= \gamma^{2^d-1} &&
\end{aligned}$$

Now, we add the upper bounds of the variables. With Observation 10, we have $r(\ell) \leq \gamma^{2^\ell}-1$. In combination with Observation 8, we get $y(\ell) \leq \lfloor (\gamma^{2^{\ell+1}}-1)/\gamma^{2^\ell} + 1/(\gamma^{2^\ell}+1) \rfloor \leq \gamma^{2^\ell}$. $x^{\texttt{bin}}(\ell)$ is bounded by 1 due to $\texttt{C}_{\ell,5}$ and it holds that $z(\ell) \leq \min(\gamma^{2^\ell}, \gamma^{2^\ell}-1) = \gamma^{2^\ell}-1$ by $\texttt{C}_{\ell,12}$ and $\texttt{C}_{\ell,13}$. The slack variables are bounded by the coefficients, the variables and the right-hand-side of their constraint. For instance, since $\gamma^{2^\ell}x^{\texttt{bin}}(\ell)-z(\ell)+r(\ell)+s_7(\ell)=\gamma^{2^\ell}$, we have $s_7(\ell)=\gamma^{2^\ell}-\gamma^{2^\ell}x^{\texttt{bin}}(\ell)+z(\ell)-r(\ell)$. With $x^{\texttt{bin}}(\ell)\geq 0$, $r(\ell)\geq 0$ and the bound on $z(\ell)$, this implies $s_7(\ell) \leq \gamma^{2^\ell}+\gamma^{2^\ell}-1 = 2\gamma^{2^\ell}-1$.

Now, as done in Section 3, we define U to be the sum of the upper bounds of all variables (including slacks) and introduce the constraint which restricts the sum of all variables by U. Let $v \coloneqq \sum_{\ell=0}^{d-1}(y(\ell)+z(\ell)+r(\ell)+\sum_{j=1}^{9}(s_j(\ell)))+r(d)+s_{\alpha-3}+s_{\alpha-2}$ be the sum of all variables. Thus, our additional constraint is $v+s_\alpha = U$. Setting $M \coloneqq 2\gamma^{2^d}\cdot U + \gamma^{2^d-1}+2\gamma^{2^d}+2$ and multiplying the ILP with the vector $(1, M, M^2, \ldots, M^{\alpha-1})$ results in an ILP of the form (3) which is equivalent to the Constraints 2.

Lemma 5 now implies that vector $\texttt{sol} = (x^{\texttt{bin}}, y, z, r, s)^T \in \mathbb{Z}_{\geq 0}^{12d+4}$ is a feasible solution to Constraints 2 if and only if $\texttt{sol}$ is a solution to

$$\begin{aligned}
r(0)+\textstyle\sum_{\ell=0}^{d-1}\Big(& M^{9\ell+1}\big((\gamma^{2^{\ell+1}}+\gamma^{2^\ell})y(\ell)-(\gamma^{2^\ell}+1)r(\ell+1)+s_1(\ell)\big)\\
&+M^{9\ell+2}\big(-\gamma^{2^\ell}y(\ell)+r(\ell+1)+s_2(\ell)\big)\\
&+M^{9\ell+3}\big(x^{\texttt{bin}}(\ell)+s_3(\ell)\big)\\
&+M^{9\ell+4}\big(x^{\texttt{bin}}(\ell)-y(\ell)+s_4(\ell)\big)\\
&+M^{9\ell+5}\big(y(\ell)-(\gamma^{2^\ell}+1)x^{\texttt{bin}}(\ell)+s_5(\ell)\big)\\
&+M^{9\ell+6}\big((\gamma^{2^\ell}-1)z(\ell)+r(\ell)-r(\ell+1)\big)\\
&+M^{9\ell+7}\big(\gamma^{2^\ell}x^{\texttt{bin}}(\ell)-z(\ell)+r(\ell)+s_7(\ell)\big)\\
&+M^{9\ell+8}\big(-\gamma^{2^\ell}x^{\texttt{bin}}(\ell)+z(\ell)+s_8(\ell)\big)\\
&+M^{9\ell+9}\big(z(\ell)-r(\ell)+s_9(\ell)\big)\Big)\\
&+M^{\alpha-3}\big(r(d)-s_{\alpha-3}\big)\\
&+M^{\alpha-2}(r(d)+s_{\alpha-2})\\
&+M^{\alpha-1}(v+s_\alpha)\\
&=1+\textstyle\sum_{\ell=1}^{d-1}\Big(M^{9\ell+1}\gamma^{2^\ell}+M^{9\ell+2}(\gamma^{2^\ell}-1)+M^{9\ell+3}+M^{9\ell+7}\gamma^{2^\ell}\Big)\\
&\quad +M^{\alpha-3}\cdot 2+M^{\alpha-2}\gamma^{2^d-1}+M^{\alpha-1}U.
\end{aligned} \tag{7}$$

This equation can be transformed into an unbounded knapsack constraint by grouping coefficients such that each variable only occurs once. These coefficients then define the item size vector s. For example, the sizes of the item $x^{\texttt{bin}}(\ell), \ell \in [d-1]_0$ is $s_{x^{\texttt{bin}}(\ell)} = M^{9\ell+3} + M^{9\ell+4} + M^{9\ell+7} \cdot \gamma^{2^\ell} - M^{9\ell+8} \cdot \gamma^{2^\ell}$. To complete the transformation, set the lower bound of each variable to 0 and exchange the equality into $\leq$, i.e., the sum of item sizes and multiplicites is at most the right hand side. Denote the polytope as $P = \{x \in \mathbb{R}^d | x_i, s^T x \leq C\}$. All feasible configurations are then integer points in P, i.e., $p \in P \cap \mathbb{Z}^d$.

4.4 Main proof

With this, we are ready to prove our main theorem.

▶ **Theorem 1.** *There exists a high-multiplicity* Bin Packing *instance of dimension d such that the solution-vector x (a vector of configurations) contains at least $2^{\Omega(d)}$ non-zero entries, i.e., $|supp(x)| \in 2^{\Omega(d)}$.*

Proof. Let $d' \in \mathbb{Z}_{\geq 0}$. The dimension of the Bin Packing instance has dimension d, i.e., number of different item sizes, $d = 12d' + 4$. Denote the capacity of a single bin as C and set the sizes according to (7), and denote them as the size vector s. Set C to be the right hand side of the knapsack constraint (7). Define the knapsack polytope $P = \{x \in \mathbb{R}^d | x_i, s^T x \leq C\}$. Let X be the set of all solutions in $P \cap \mathbb{Z}^d$, i.e., with total size less than or equal to C. Define $X^* = \{x \in X | s^T x = A\}$ to be the set of all solutions with size exactly C. See Figure 3 for an illustration. Thus, X is the set of all feasible configurations. By the definition of the knapsack constraint and Lemma 12, we know that $|X^*| = 2^{d'} - 1 =: k$, as (7) has exactly one solution for every possible combination of $x^{\texttt{bin}}(\ell), \ell \in [d-1]_0$ with $x^{\texttt{bin}}(\ell) \in \{0, 1\}$. Define the total amount of items y as the sum of all configurations, i.e., $y = \sum_{x \in X^*} x$. Thus, the total size of all items to be placed is $y^T \cdot s = kC$.

Clearly, the optimal solution to this Bin Packing instance uses exactly k bins. One possible solution is to take each configuration in X^* exactly once. In the following we show that this is the only solution using elements in X and using exactly k bins. Now, define the projection $\pi : x \to (x^{\texttt{bin}}(0), \dots x^{\texttt{bin}}(d'-1), r(d'))$. This projection reduces every configuration to just the binary encoding and the value $r(d')$. It holds that $\pi(y) = \begin{pmatrix} 2^{d-1} \\ \vdots \\ 2^{d-1} \\ \sum_{i=1}^{k} (4k)^i \end{pmatrix}$. Recall that we set $\gamma = 4k$. We can not consider configurations in $X \setminus X^*$ as they have size $< C$. As the total size of the right hand size is exactly kC taking one of these configurations while still using k bins would require another bin to be filled $> C$, which is infeasible due to (7). Thus, there are no other feasible configurations, as only those of the form (after applying π) exactly like in Lemma 6, i.e., those in X^*, have size exactly C. Thus, when applying the projection π, we have met the conditions of applying Lemma 6 and know that the only optimal solution takes each of the k vectors exactly once.

Therefore, after removing the projection, the only optimal solution to the Bin Packing instance uses every configuration in X^* exactly once, and therefore we have $|\text{supp}(x)| = k = 2^{d'} - 1$. As we have $d = 12d' + 4$ we have $|\text{supp}(x)| \in 2^{\Omega(d)}$. ◀

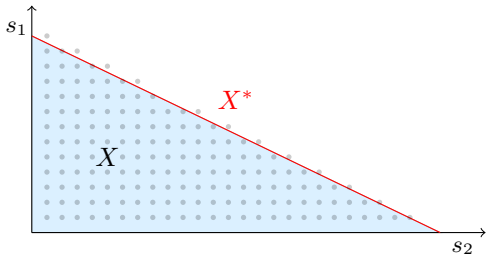


Figure 3 A 2-dimensional example of the knapsack constraint with item types s_1, s_2. The shaded area denotes the knapsack polytope P. The points resemble integer points and show all feasible configurations. Integer points in the shaded region are the configurations in the set X. Integer points on the red line form the set X^* of configurations. Only configurations in X^* fill the knapsack constraint with equality.

References

1 Iskander Aliev, Jesús A. De Loera, Friedrich Eisenbrand, Timm Oertel, and Robert Weismantel. The support of integer optimal solutions. *SIAM J. Optim.*, 28(3):2152–2157, 2018. doi:10.1137/17M1162792.

2 Iskander Aliev, Jesús A. De Loera, Timm Oertel, and Christopher O'Neill. Sparse solutions of linear diophantine equations. *SIAM J. Appl. Algebra Geom.*, 1(1):239–253, 2017. doi:10.1137/16M1083876.

3 Sebastian Berndt, Hauke Brinkop, Klaus Jansen, Matthias Mnich, and Tobias Stamm. New support size bounds for integer programming, applied to makespan minimization on uniformly related machines. In *ISAAC*, volume 283 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.13.

4 Gordon H. Bradley. Transformation of integer programs to knapsack problems. *Discrete Mathematics*, 1(1):29–45, 1971. doi:10.1016/0012-365X(71)90005-7.

5 Gordon H. Bradley, Peter L. Hammer, and Laurence A. Wolsey. Coefficient reduction for inequalities in 0-1 variables. *Math. Program.*, 7(1):263–282, 1974. doi:10.1007/BF01585527.

6 Václav Chvátal and Peter Hammer. Aggregation of inequalities in integer programming. *Ann. Discrete Math*, 1:145–162, 1977.

7 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. Reducibility bounds of objective functions over the integers. *Oper. Res. Lett.*, 51(6):595–598, 2023. doi:10.1016/J.ORL.2023.10.001.

8 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006. doi:10.1016/J.ORL.2005.09.008.

9 SE Elmaghraby and MK Wig. On the treatment of stock cutting problems as diophantine programs. *Operations Research Report*, 61, 1970.

10 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Comb.*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

11 Fred W. Glover and Djangir A. Babayev. New results for aggregating integer-valued equations. *Ann. Oper. Res.*, 58(3):227–242, 1995. doi:10.1007/BF02032133.

12 Fred W. Glover and Robert E. D. Woolsey. Aggregating diophantine equations. *Z. Oper. Research*, 16(1):1–10, 1972. doi:10.1007/BF01917186.

13 Michel X. Goemans and Thomas Rothvoss. Polynomiality for bin packing with a constant number of item types. *J. ACM*, 67(6):38:1–38:21, 2020. doi:10.1145/3421750.

14 A. C. Hayes and David G. Larman. The vertices of the knapsack polytope. *Discret. Appl. Math.*, 6(2):135–138, 1983. doi:10.1016/0166-218X(83)90067-7.

15 Klaus Jansen. An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. *SIAM J. Discret. Math.*, 24(2):457–485, 2010. doi:10.1137/090749451.

16 Klaus Jansen, Kai Kahler, and Esther Zwanger. Exact and approximate high-multiplicity scheduling on identical machines. *CoRR*, abs/2404.17274, 2024. doi:10.48550/arXiv.2404.17274.

17 Klaus Jansen and Kim-Manuel Klein. A robust AFPTAS for online bin packing with polynomial migration. *SIAM J. Discret. Math.*, 33(4):2062–2091, 2019. doi:10.1137/17M1122529.

18 Klaus Jansen and Kim-Manuel Klein. About the structure of the integer cone and its application to bin packing. *Math. Oper. Res.*, 45(4):1498–1511, 2020. doi:10.1287/MOOR.2019.1040.

19 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Math. Oper. Res.*, 45(4):1371–1392, 2020. doi:10.1287/MOOR.2019.1036.

20 Klaus Jansen and Roberto Solis-Oba. A polynomial time $OPT + 1$ algorithm for the cutting stock problem with a constant number of object lengths. *Math. Oper. Res.*, 36(4):743–753, 2011. doi:10.1287/MOOR.1110.0515.

21 Kenneth E Kendall and Stanley Zionts. Solving integer programming problems by aggregating constraints. *Operations Research*, 25(2):346–351, 1977.

22 Manfred W. Padberg. Equivalent knapsack-type formulations of bounded integer linear programs: An alternative approach. *Naval Research Logistics Quarterly*, 19(4):699–708, 1972. doi:10.1002/nav.3800190410.

23 Pierre-Louis Poirion. Optimal constraints aggregation method for ILP. *Discret. Appl. Math.*, 262:148–157, 2019. doi:10.1016/J.DAM.2019.02.014.

24 I.G. Rosenberg. Aggregation of equations in integer programming. *Discrete Mathematics*, 10(2):325–341, 1974. doi:10.1016/0012-365X(74)90126-5.

Efficient Top-Down Updates in AVL Trees

Vincent Jugé
LIGM, CNRS, Univ Gustave Eiffel, Marne-la-vallée, France
IRIF, Université Paris-Cité & CNRS, France

Abstract

Since AVL trees were invented in 1962, two major open questions about rebalancing operations, which found positive answers in other balanced binary search trees, were left open: can these operations be performed top-down (with a fixed look-ahead), and can they use an amortised constant number of write operations per update? We propose an algorithm that solves both questions positively.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases AVL trees, data structures, amortised complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.49

1 Introduction

Balanced search trees are among the most fundamental and basic data structures in computer science. Their formidable story started with the invention of AVL trees by Adel'son-Vel'skii and Landis [1] in 1962, who proposed a data structure and a maintenance algorithm thanks to which the insertion, deletion and research of an element in a linearly ordered set of size n was feasible in time $\mathcal{O}(\log(n))$.

AVL trees are binary search trees in which each node x maintains some form of *balance*: the heights of its children differ from each other by at most 1. Satisfying this property at each node ensures that the tree is of height $\mathcal{O}(\log(n))$. The number of comparisons required to search an element in a binary search tree being linear in the height of the tree, it is thus logarithmic in n.

However, for instance, inserting a node into the sub-tree rooted at the left child of our node x may increase the height of that sub-tree, damaging the balance of x. Thus, the challenge of maintenance algorithms consists in efficiently modifying the parenthood relations inside the tree in order to ensure that each node remains balanced.

Unfortunately, in spite of their excellent worst-case height, AVL trees suffer from updating algorithms that are significantly less efficient than those of other balanced binary search tree structures such as weight-balanced trees [9], red-black trees [4], half-balanced trees [11], splay trees [12] or, more recently, weak AVL trees [5]. Indeed, as illustrated in Table 1, they have two shortcomings, from which almost all other algorithms are exempt:

- starting from an empty tree, performing n updates in a row may trigger $\Theta(n\log(n))$ rotations [2] or, more generally, write operations, which may also consist in modifying the balance of a node without changing the tree structure;
- updates following an insertion or deletion must be performed in a bottom-up fashion; the inventors of weak AVL trees even said, in [5, p. 23], that "Top-down insertion or deletion with fixed look-ahead in an AVL tree is problematic. (We do not know of an algorithm; we think there is none.)"

Requiring only $\Theta(1)$ amortised write operations per update instead of $\Theta(\log(n))$ would clearly improve the efficiency of AVL trees. As mentioned in [4, 5], being able to use top-down algorithms is also important, because requiring a bottom-up pass is costly: either it forces each node to maintain a link toward its parent, or it must be performed *via* recursive calls or


© Vincent Jugé;
licensed under Creative Commons License CC-BY 4.0

Table 1 Amortised number of rewrite operations per update with n nodes, and existence of top-down update algorithms. Next to each piece of information are indicated references where it is proved.

Tree family	Amortised write cost/update	Top-down update algorithm
AVL (state of the art)	$\Theta(\log(n))$ [2]	no
AVL (this article)	$\Theta(1)$	yes
Weight-balanced	$\Theta(1)$ [3]	yes [7, 8]
Red-black	$\Theta(1)$ [4]	yes [4]
Half-balanced	$\Theta(1)$ [11]	for insertions [10]
Splay	$\Theta(\log(n))$ [12]	yes [12]
Weak AVL	$\Theta(1)$ [5]	yes [5]

storing visited nodes in a stack. This second approach is even more expensive in distributed settings, because each process should lock the entire branch it is visiting, preventing any update from starting before the previous update has ended.

A way of circumventing these deficiencies of AVL trees is to use weak AVL trees instead [5]. By having less demanding balance criteria, this variant of AVL offers update algorithms that require $\mathcal{O}(1)$ rotations in the worst case, and $\mathcal{O}(1)$ amortised write operations (which may include updating information on the balance of nodes) per update. Furthermore, these algorithms can be made top-down, with the following exception, which is common to all binary search tree data structures: instead of deleting an internal node x, we must delete a node y whose key immediately precedes or follows the key of x, an approach called Hibbard deletion [6], and then replace the key of x by that of y. In AVL trees, y must be a leaf or the parent of a leaf, which reduces the problem of node deletion to that of leaf deletion, up to to keeping a pointer on an internal node we may wish to delete.

In this article, we prove that top-down insertion and deletion with fixed look-ahead in an AVL tree *is* feasible; having a look-ahead of d means that the algorithm, when it reads or writes information at a node x, is forbidden to later read or write informations at nodes that do not descend from the d^{th} ancestor of x. At the same time, this can also be write-efficient, thus bridging a gap of theoretical importance: it was somewhat unsettling to observe that this seminal balanced tree structure was also the one for which no efficient algorithms had been found, thus leading specialists in this field to conjecture that such algorithms did not exist.

We proceed step-by-step: in Section 2, we briefly present textbook algorithms for updating AVL trees; in Section 3, we propose a bottom-up algorithm that requires $\mathcal{O}(1)$ amortised write operations per update; finally, in Section 4, we propose a top-down algorithm that requires $\mathcal{O}(1)$ amortised write operations per update. Note that, unlike algorithms for AVL trees or some other structures (e.g., red-black trees), a given update may require these algorithms to perform $\Theta(\log(n))$ rotations: our efficiency criterion is based on the amortised number of write operations (and, in particular, rotations) only, not on the maximal number of such operations that a single update may require.

Both algorithms have flavours similar to those used when designing weak AVL trees [5]: they rely on making sure that "transient" rebalancing operations decrease some integer-valued potential and, for the top-down algorithm, on being able to start and stop long enough yet finite sequences of such transient operations. This illustrates the usefulness of the approach used to design these algorithms, and also suggests that additional results similar

to [5, Theorem 7.4] should also be valid here. The main difficulties, however, reside first in designing a suitable notion of potential, and then in managing to start and stop these sequences of transient operations; this task is easy when dealing with insertions only, and quite challenging when taking deletions into account.

2 State-of-the-art

2.1 Preliminaries

A *binary search tree* is an ordered rooted tree $\mathcal{T}$ in which each tree node x has one *key* $\mathsf{key}(x)$, one *left child* x_1 and one *right child* x_2, which can be *tree nodes* or *null nodes*; null nodes represent the empty tree and have no key nor children, and tree nodes whose two children are null nodes are called *leaves*. By definition, the *descendants* of x consist of x, its children x_1 and x_2, and all their descendants (if they are tree nodes); these descendants form a sub-tree of $\mathcal{T}$ rooted at x and denoted by $\mathcal{T}(x)$, and x is one of their *ancestors*.

Node keys are pairwise distinct, and must belong to a linearly ordered set. For each tree node y that descends from x, we have $\mathsf{key}(y) < \mathsf{key}(x)$ if y belongs to $\mathcal{T}(x_1)$, and $\mathsf{key}(y) > \mathsf{key}(x)$ if y belongs to $\mathcal{T}(x_2)$. Moreover, the *height* of x is the length (i.e., the number of parent-child edges) of the longest downward path starting from x and ending in a leaf; it is denoted by $\mathsf{h}(x)$. By convention, each null node has height -1.

A binary search tree is an AVL tree when the height difference $\mathsf{h}(x_1) - \mathsf{h}(x_2)$ belongs to the set $\{-1, 0, 1\}$ for all tree nodes x. This ensures that an AVL tree with n nodes is of height $\mathcal{O}(\log(n))$.

Here, we follow the presentation of [5] and assume that AVL tree structures are implemented by assigning a *rank* $\mathsf{r}(x)$ to each tree node x. This rank is our current estimation of the height $\mathsf{h}(x)$, which may be slightly invalid if, say, some leaf was inserted way below x, and we did not yet have the time to update information about the height of x. Hence, we should control the rank differences $\mathsf{r}(x) - \mathsf{r}(x_i)$ for each node x. Focusing on these differences, we say that x_i is an ℓ-child if $\mathsf{r}(x) - \mathsf{r}(x_i) = \ell$, and that x is an $\{\ell, \ell'\}$-node if its children are ℓ- and ℓ'-children.

In practice, these ranks are rarely stored directly in AVL implementations: instead, each node x contains either (i) the difference $\mathsf{r}(x_2) - \mathsf{r}(x_1)$ between the ranks of its children x_1 and x_2, or (ii) the difference $\mathsf{r}(p) - \mathsf{r}(x)$ between the rank of its parent p and its own rank. In what follows, we will disregard these implementation details, and pretend that we stored node ranks in clear; adapting our algorithm to let it fit the frameworks (i) or (ii) is easy.

Since binary search trees represented ordered sets, the three main operations they must support are searching, inserting, and deleting an element. Searching an element is easy: when searching a key k in a non-empty tree $\mathcal{T}(x)$, we compare k with $\mathsf{key}(x)$; then, we keep searching k in $\mathcal{T}(x_1)$ if $k < \mathsf{key}(x)$, or in $\mathcal{T}(x_2)$ if $k > \mathsf{key}(x)$, or we declare that k was found if $k = \mathsf{key}(x)$; whereas, when searching k in an empty tree, we must declare that k was nowhere to be found.

Inserting or deleting an element is substantially more difficult, because it requires altering the structure of the tree: this may change its height and even damage the balance of some tree nodes. As a result, the relation $\mathsf{r}(x) = \max\{\mathsf{r}(x_1), \mathsf{r}(x_2)\} + 1$ might fail for one node x, or one rank difference $\mathsf{r}(x_1) - \mathsf{r}(x_2)$ might no longer belong to the set $\{-1, 0, 1\}$. The purpose of bottom-up rebalancing algorithms is precisely to eliminate this anomaly, either directly, or by "propagating it upward", i.e., making it concern a strict ancestor of x, with rank larger than $\mathsf{r}(x)$.

2.2 Tree cuts and cut rebalancing

A *cut* of a tree $\mathcal{T}$ is a list $(x^1, x^2, \dots, x^\ell)$ of nodes of $\mathcal{T}$ that are pairwise incomparable for the ancestorship relation (i.e., the sub-trees $\mathcal{T}(x^i)$ are pairwise disjoint), ordered from left to right, and which is maximal for inclusion.

Let x denote the root of $\mathcal{T}$. When the nodes $x^1, x^2, \dots, x^\ell$ have ranks $\mathsf{r}(x) - \delta^1$, $\mathsf{r}(x) - \delta^2, \dots,$ $\mathsf{r}(x) - \delta^\ell$, the *profile* of the associated cut is defined as the integer tuple $\boldsymbol{\delta} = (\delta^1, \delta^2, \dots, \delta^\ell)$. Accordingly, we say that a node x has profile $\boldsymbol{\delta}$ when $\mathcal{T}(x)$ contains a tree cut of profile $\boldsymbol{\delta}$. When the order of the integers $\delta^1, \delta^2, \dots, \delta^\ell$ is irrelevant, we may also say that x has profile $\{\delta^1, \delta^2, \dots, \delta^\ell\}$, just writing a multiset of integers: this just means that there exists a permutation σ of $\{1, 2, \dots, \ell\}$ such that x has profile $(\delta^{\sigma(1)}, \delta^{\sigma(2)}, \dots, \delta^{\sigma(\ell)})$. In particular, an $\{\ell, \ell'\}$-node is simply a node with profile $\{\ell, \ell'\}$.

Below, we may modify the structure of a tree $\mathcal{T}$, thus generalising so-called *rotations*: starting from a cut $(x^1, x^2, \dots, x^\ell)$ of $\mathcal{T}$, let $a^1, a^2, \dots, a^{\ell-1}$ be the strict ancestors of the cut nodes x^i, listed from left to right. We assign a new pair of children to each node a^i, and decide that the new root should be some node a^j, as long as the graph we obtain is a binary search tree rooted at a^j; nodes a^j are said to be *affected* by the cut rebalancing. While doing so, (i) we must never modify the structure or rank of any of the sub-trees $\mathcal{T}(x^i)$, and (ii) node ranks of nodes a^i are reassigned to make sure that each node a^i is a $\{1, 1\}$- or a $\{1, 2\}$-node in the resulting tree. Such an operation is called a *cut rebalancing*, and is easy to describe diagrammatically, like in Figure 1 below; cut nodes and their ancestors will typically be denoted with capital letters, from left to right.

2.3 Textbook bottom-up algorithm

Inserting a key may create a *zero-edge*, i.e., an edge between a parent x and its child x_i such that $\mathsf{r}(x) = \mathsf{r}(x_i)$; in that case, the integer $\mathsf{r}(x) = \mathsf{r}(x_i)$ is called the *rank* of the zero-edge. A zero-edge is created when x is a leaf below which we wish to insert the key k, because some child x_i of x, which used to be null, becomes a leaf.

Similarly, deleting a key may create a $\{2, 2\}$- or $\{1, 3\}$-node, which we call a *four-node*; this happens if x is a $\{1, 2\}$-node one of whose children is a leaf containing the key k we wish to delete, because that leaf will be transformed into a null node.

Zero-edges are dealt with as follows, and as illustrated in Figure 1. Up to using a left-right symmetry, and without loss of generality, we assume that $\mathsf{r}(x_1) \geqslant \mathsf{r}(x_2)$, i.e., that both $\mathsf{r}(x)$ and $\mathsf{r}(x_1)$ are equal to the rank r of the zero-edge:

1. If $\mathsf{r}(x_{11}) = \mathsf{r}(x_{12}) = \mathsf{r} - 1$ and $\mathsf{r}(x_2) = \mathsf{r} - 2$ (row 1, left), the cut (x_{11}, x_{12}, x_2) has profile $(1, 1, 2)$. Let A, C and E denote the cut nodes x_{11}, x_{12} and x_2, and let B and D denote their ancestors x_1 and x. We rebalance $\mathcal{T}(x)$ by letting D become the parent of C and E, and B become the parent of A and D, thus obtaining a tree with root B. This cut rebalancing is a *simple rotation*.
2. If $\mathsf{r}(x_{11}) = \mathsf{r} - 1$ and $\mathsf{r}(x_{12}) = \mathsf{r}(x_2) = \mathsf{r} - 2$ (row 1, right), we perform the same rotation; however, the resulting ranks differ.
3. If $\mathsf{r}(x_{12}) = \mathsf{r} - 1$ and $\mathsf{r}(x_{11}) = \mathsf{r}(x_2) = \mathsf{r} - 2$ (row 2, left), the cut $(x_{11}, x_{121}, x_{122}, x_2)$ is to be rebalanced. Let A, C, E and G denote the cut nodes x_{11}, x_{121}, x_{122} and x_2, and let B, D and F denote their ancestors x_1, x_{12} and x. We rebalance $\mathcal{T}(x)$ by giving these seven nodes the shape of a complete binary tree: B becomes the parent of A and C, F becomes the parent of E and G, and D becomes the parent of B and F, as well as the root of the resulting tree. This is a *double rotation*.
4. If $\mathsf{r}(x_2) = 1$ (row 2, right), we rebalance $\mathcal{T}(x)$ by not changing its structure, but simply increasing the rank of its root x. This is a *promotion*.

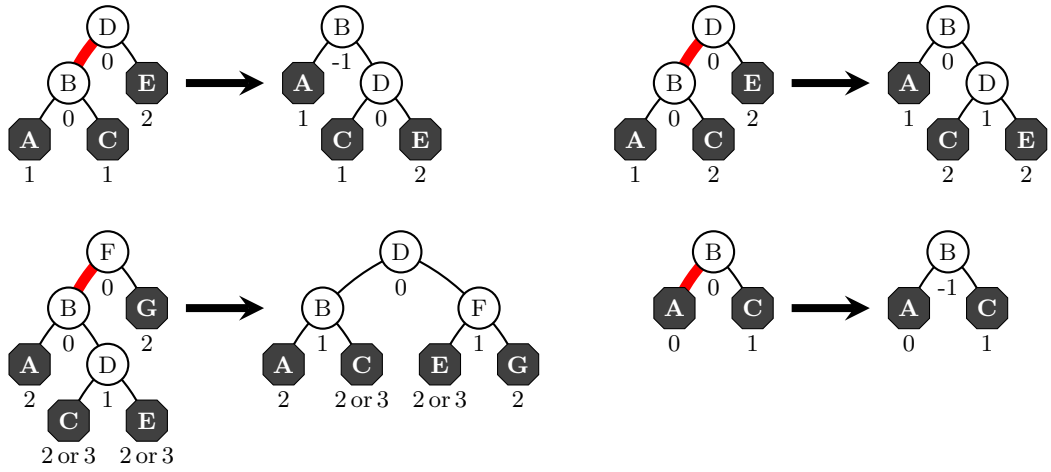


Figure 1 Eliminating or propagating upward a (thick, red-painted) zero-edge of rank r through cut rebalancings. Black octagons represent cut nodes, at which are rooted sub-trees whose structure and ranks will not change. White circles represent their ancestors. Once the rebalancing operation has taken place, the rank of a node is defined as the smallest integer larger than the ranks of its children. Below each node z is written the rank difference $\mathsf{r} - \mathsf{r}(z)$; when two rank differences δ and δ' are possible, we just write "δ or δ'". These notations will be used in all subsequent diagrams.

Doing so, whenever we face a zero-edge e of rank r, we will either eliminate it at once, or replace it by a new zero-edge e' of rank $\mathsf{r}+1$, placed just above where e' used to be. Consequently, if inserting a key creates a zero-edge, we will simply propagate that zero-edge upward until it disappears.

Likewise, four-nodes are dealt with as follows, and as illustrated in Figure 2. Up to using a left-right symmetry, we assume that $\mathsf{r}(x_1) \geqslant \mathsf{r}(x_2)$:

5. If $\mathsf{r}(x_{11}) = \mathsf{r}(x_{12}) = \mathsf{r}-2$ and $\mathsf{r}(x_2) = \mathsf{r}-3$ (row 1, left), we perform a simple rotation.
6. If $\mathsf{r}(x_{11}) = \mathsf{r}-2$ and $\mathsf{r}(x_{12}) = \mathsf{r}(x_2) = \mathsf{r}-3$ (row 1, right), we perform the same rotation; the resulting ranks differ.
7. If $\mathsf{r}(x_{12}) = \mathsf{r}-2$ and $\mathsf{r}(x_{11}) = \mathsf{r}(x_2) = \mathsf{r}-3$ (row 2, left), we perform a double rotation.
8. If $\mathsf{r}(x_2) = \mathsf{r}-2$ (row 2, right), we decrease the rank of its root x; this is a *demotion*.

Cut rebalancing operations 5 to 7 are the same as operations 1 to 3 for eliminating zero-edges, except that the ranks of x_1, x_2 and their descendants were all shifted down by one. In all eight cases, these operations result either in eliminating the four-node x or in making the (former) parent of x our new four-node.

3 Efficient bottom-up updates

3.1 Overview and complexity analysis

The textbook algorithm presented in Section 2.3 aims at eliminating or propagating upward zero-edges and four-nodes. Thus, each insertion or deletion update consists in (i) finding the location at which we want to create or delete a tree leaf, (ii) observe that we may have created a zero-edge or four-node, (iii) moving this anomaly upward, until (iv) we eliminate it once and for all. Step (i) requires no write operation on the tree structure, and steps (ii) and (iv) occur once per update. However, step (iii) may require arbitrarily many write operations per update.

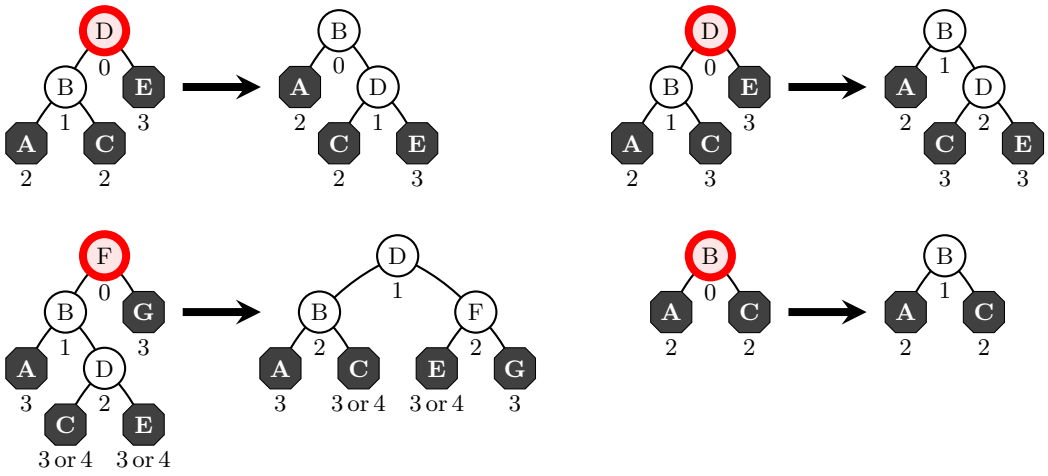


Figure 2 Eliminating or increasing the rank of a (thick, red-painted) four-node.

Hence, cut rebalancing operations of cases 1 to 8 are called *terminal* if they result in eliminating the anomaly (thus being a part of step (iv)), and *transient* if they just propagate it upward (thus being a part of step (iii)): transient operations are operations 1 and 4 when x used to be a 1-child, and operations 6 to 8 when x had a parent that used to be a $\{1,2\}$-node. We should prove that updating n times an initially empty AVL tree triggers $\mathcal{O}(n)$ transient operations.

With the textbook algorithm, the statement we wish to prove is invalid, as was shown in [2]. Consequently, we will tweak the operations performed in cases 1 to 8, aiming at making them terminal whenever possible.

The complexity proof of the resulting algorithm is based on *potential* techniques. In our case, we define the *potential* of a a tree $\mathcal{T}$ as the the number of $\{1,2\}$-nodes, plus twice the number of *full* nodes it contains, where full nodes are defined as follows.

In absence of zero-edges, a node x is called *full* when x and its two children x_1 and x_2 are $\{1,1\}$-nodes, and at least one of its grand-children x_{12} and x_{21} is a $\{1,1\}$-node. In other words, a node is full when it has profile $(2,3,3,2,2)$ or, symmetrically, $(2,2,3,3,2)$. When zero-edges are (temporarily) allowed, a node x may also be considered full if x is a $\{0,1\}$-node whose 1-child is a $\{1,1\}$-node; or if x is a $\{1,1\}$-node, x_1 and x_2 are $\{0,1\}$- and/or $\{1,1\}$-nodes, and one of x_{12} and x_{21} is a 0-child or a null, a $\{0,1\}$- or a $\{1,1\}$-node.

As we will prove:

1. each update or write operation modifies the neighbourhood (parenthood relations and/or rank) of a constant number of nodes, thus increasing the tree potential by at most a constant;
2. each transient operation aimed at eliminating a zero-edge of rank $\mathsf{r} \geqslant 4$ destroys one full node and creates one $\{1,2\}$-node, thus decreasing the tree potential;
3. each transient operation triggered by a deletion decreases the number of $\{1,2\}$-nodes, and does not increase the number of full nodes, also decreasing the tree potential.

Statement 1 will be immediate, and thus we will focus on proving statement 2 in Section 3.3 (this will be Lemma 3), and statement 3 in Section 3.4 (this will be Lemma 4). Once all three statements are proved, we will finally be able to derive the following result.

▶ **Theorem 1.** *The updating algorithm that we present in Sections 3.3 (for insertions) and 3.4 (for deletions) performs $\mathcal{O}(1)$ amortised write operations per update.*

Proof. Starting from an empty tree, let n insertion or deletion updates be performed. Let us keep track of the variations of tree potential while these updates are performed. There exists a constant K such that each update requires:

1. inserting or deleting a leaf, an operation that increases the potential by at most K;
2. eliminating at most four zero-edges of rank $\mathsf{r} \leqslant 3$, each of which increases the potential by at most K;
3. performing a certain number of transient operations, each of which decreases the potential by at least one;
4. performing at most one terminal operation, which increases the potential by at most K.

Hence, if T transient operations were performed in total, the potential increased by at most $6n\mathsf{K} - \mathsf{T}$. It started with the value 0, and ended with a non-negative value, which shows that $\mathsf{T} \leqslant 6n\mathsf{K}$. Therefore, in total, these n updates required performing no more than $\mathcal{O}(n) + \mathcal{O}(\mathsf{T}) \subseteq \mathcal{O}(n)$ write operations. ◀

3.2 Cut promotions and demotions

Some cut profiles, such as the ones that charaterise full nodes, allow rebalancing trees to either increase or decrease their rank. More precisely, if we are lucky enough, a tree $\mathcal{T}$ with rank r and profile $\boldsymbol{\delta}$ will always admit a cut, whose profile does not need to be $\boldsymbol{\delta}$, that can be rebalanced to obtain a tree $\mathcal{T}'$ of rank $\mathsf{r}' = \mathsf{r} + \varepsilon$, where $\varepsilon = \pm 1$ is an integer of our choice. This cut rebalancing operation is called a *cut promotion* when $\varepsilon = 1$, and a *cut demotion* when $\varepsilon = -1$.

Some profiles ensure that a cut promotion or demotion can be performed, as outlined by the following result.

▶ **Lemma 2.** *Trees with a full root can always be subject to a cut promotion; the root of the resulting tree is a $\{1,2\}$-node. By contrast, trees with profile $\{3,3,3,4,4\}$ can always be subject to a cut demotion; the root of the resulting tree is a $\{1,1\}$-node.*

Proof. The recipe given in Figure 3 (row 1) shows how to apply a cut promotion to a tree whose root is full. Hence, let us focus on applying a cut demotion on a tree $\mathcal{T}$ that admits a cut $(x^1, x^2, x^3, x^4, x^5)$ with profile $\{3,3,3,4,4\}$.

Replacing the three nodes x^i with rank $\mathsf{r}-3$ by their children gives an 8-node cut, whose nodes all have rank $\mathsf{r}-4$ or $\mathsf{r}-5$, at least five of these nodes having rank $\mathsf{r}-4$. These 8 nodes and their 7 ancestors can be reshaped to form a 15-node complete binary tree $\mathcal{T}'$ of height 3, whose nodes at height h must have rank $\mathsf{r}-5+h$ or $\mathsf{r}-4+h$. Moreover, among the four leaves (y^1, y^2, y^3, y^4), at least one has rank $\mathsf{r}-4$, which proves that the root x' of $\mathcal{T}'$ has a left child x'_1 of rank $\mathsf{r}-1$; similarly, its right child has rank $\mathsf{r}-2$, and x' is a $\{1,1\}$-node of rank $\mathsf{r}-1$. This is illustrated in Figure 3 (row 2). ◀

In general, when cut-rebalancing a k-node cut transforms a sub-tree $\mathcal{T}(x)$ into a tree $\mathcal{T}'$ with root x' and rank $\mathsf{r}(x)-1$, $\mathsf{r}(x)$ or $\mathsf{r}(x)+1$, the tree potential increases by no more than $2k+4$. Indeed, $k-1$ cut ancestors were affected, possibly becoming $\{1,2\}$-nodes or full nodes; the parent of x, whose child is now x', might also become a $\{1,2\}$-node or a full node; and the grand-parent and great-grand-parent of x might become full nodes. Hence, cut promotions increase the tree potential by 14 or less, and cut demotions increase this potential by 20 or less.

These two kinds of cut promotions and demotions will be used in Sections 3.3 and 3.4. Another kind of cut rebalancing operation, called *saving*, will be introduced in Section 4.3, only being useful for designing efficient top-down updating algorithms.

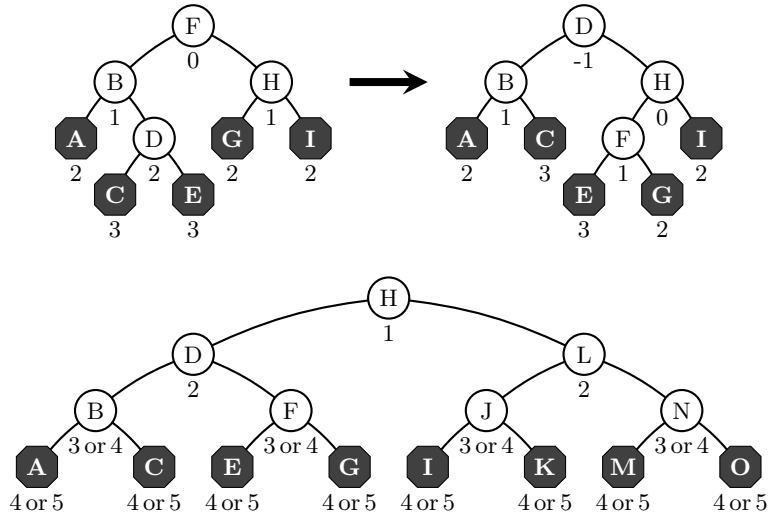


Figure 3 Row 1: Applying a cut promotion to a tree whose root is full. Row 2: 15-node complete binary tree whose root has profile $\{1, 1\}$ and rank $\mathsf{r} - 1$, which results from applying a cut demotion to a tree with profile $\{3, 3, 3, 4, 4\}$.

3.3 Insertion

Let us modify the textbook insertion algorithm to make some transient operations, which should occur in cases 1 and 4 of page 4, terminal operations. In particular, all transient operations in our variant are already transient in the textbook algorithm.

Case 1 is easy to take care of. Indeed, let us say that a node x is *weak* when x is either a leaf or a $\{1, 2\}$-node whose 1-child is also weak. An immediate induction proves that, when using the textbook algorithm, the lower end of a zero-edge is always weak. This proves, in particular, the well-known fact that case 1 is actually spurious, since it features a zero-edge whose lower end is a $\{1, 1\}$-node.

It remains to take care of case 4, which we subdivide into two sub-cases. First, we promote x like in the textbook algorithm, thus obtaining a tree $\mathcal{T}'$ of rank $\mathsf{r} + 1$. Then,

4. a) if x_2 is a $\{1, 1\}$-node, we do nothing more: we just performed a node promotion;

b) if x_2 is a $\{1, 2\}$-node, let us recall that x_1 is weak, which gives it the profile $\{2, 2, 3\}$, whereas x_2 has profile $\{1, 2\}$; hence, $\mathcal{T}'$ has profile $\{3, 3, 3, 4, 4\}$, and a cut demotion transforms $\mathcal{T}'$ into a tree $\mathcal{T}''$ of rank r again: we just performed a terminal operation.

▶ **Lemma 3.** *Each transient operation aimed at eliminating a zero-edge of rank* $\mathsf{r} \geqslant 4$ *destroys one full node and creates one* $\{1, 2\}$*-node, thus decreasing the tree potential.*

Proof. We just saw that the only transient case is case 4a, on the condition that x was a 1-child.

If the parent $x^{(1)}$ of x is full after x was promoted, it must now be a $\{0, 1\}$-node whose other child is a $\{1, 1\}$-child. Hence, before promoting x, its parent $x^{(1)}$ was already a $\{1, 1\}$-node whose children were a $\{0, 1\}$-and a $\{1, 1\}$-node, its child x having two children x_1 and x_2 that were a 0-child and a $\{1, 1\}$-node. This means that $x^{(1)}$ was already full.

Similarly, if the grand-parent $x^{(2)}$ of x is full after x was promoted, it must now be a $\{1, 1\}$-node whose children are a $\{1, 0\}$-node (the node $x^{(1)}$) and a $\{1, 1\}$-node (the sibling of $x^{(1)}$), one of it grand-children $x^{(2)}_{12}$ or $x^{(2)}_{21}$, say y, being a 0-child or a $\{1, 1\}$-node. Before promoting x, the node $x^{(1)}$ was already a $\{1, 1\}$-node, and either $y = x$, in which case y

was already a $\{0,1\}$-node, or $y \neq x$, in which case y was already a $\{1,1\}$-node. This means that $x^{(2)}$ was already full.

Finally, promoting x, in case it used to be a 1-child, does not make any other ancestor of x a full node. Hence, our transient operation transformed a full node into a $\{1,2\}$-node, and did not create any other full node or $\{1,2\}$-node: as a result, it decreased the tree potential by one. ◀

3.4 Deletion

Similarly, we wish to transform as many transient operations, which occur in cases 6 to 8 of the textbook textbook algorithm, into terminal operations.

The rotations and demotion performed in these cases transform a tree $\mathcal{T}(x)$ of rank r rooted at a four-node x into a tree $\mathcal{T}'$ of rank $\mathsf{r}-1$ whose root x' is a $\{1,1\}$-node. If x' is full, performing a cut promotion on $\mathcal{T}'$ results in another tree $\mathcal{T}''$ of rank r; similarly, if some child x'_i of x' is full, performing a cut promotion on $\mathcal{T}'(x'_i)$ and increasing the rank of x' by 1 also transforms $\mathcal{T}'$ into another tree $\mathcal{T}''$ of rank r; and if both children of x' are full, x' is also full, a case we already took care of. In all cases, this two-step transformation of $\mathcal{T}(x)$ into $\mathcal{T}''$ is a terminal operation.

If, however, neither x' nor its children is full, the rotation or demotion we performed in case 6, 7 or 8 did not create any full node. Indeed, replacing $\mathcal{T}(x)$ by a tree of smaller rank did not make any ancestor of x a full node. The only other nodes whose rank or induced sub-tree was changed are x' and its children, which we precisely assumed are not full either. In that case, our operation is transient if the parent of x, say y, was a $\{1,2\}$-node, and became a four-node.

▶ **Lemma 4.** *Each transient operation triggered by a deletion decreases the number of* $\{1,2\}$*-nodes, and does not increase the number of full nodes, also decreasing the tree potential.*

Proof. We just proved that transient operations are those where y was a $\{1,2\}$-node and neither its new child x' nor the children of x' became full. Hence, we did not create any full node, and it suffices to check, on a case-by-case basis, that the number of $\{1,2\}$-nodes decreased:

- **i)** in case 6, we destroyed two $\{1,2\}$-nodes ($\mathsf{B} = x_1$ and y);
- **ii)** in case 7, we destroyed at least two $\{1,2\}$-nodes ($\mathsf{B} = x_1$ and y) to create at most one $\{1,2\}$-node ($\mathsf{B} = x'_1$ or $\mathsf{F} = x'_2$, in case D used to be a $\{1,2\}$-node);
- **iii)** in case 8, we destroyed one $\{1,2\}$-node (y). ◀

4 Top-down updates

4.1 Overview and complexity analysis

Our top-down algorithms for inserting a key k in a tall enough AVL tree $\mathcal{T}$ and for deleting k from $\mathcal{T}$ follow the same general ideas, which were already successfully applied to weak AVL trees [5]. We will discover, in a top-down manner, the branch $x^0, x^1, \ldots$ below which k should be inserted or deleted, where x^0 is the root of $\mathcal{T}$ and each node x^{i+1} is a child of the previous node x^i. Using a $\mathcal{O}(1)$ look-ahead, and after discovering the first $\mathcal{O}(1)$ nodes on that branch, we must already decide what to do.

If we are lucky enough, we found a node x^i, for some integer $i \geqslant 1$, that is deemed *safe*: propagating a zero-edge (during insertions) or four-node (during deletions) that we would have planted along our branch far enough below x^i will stop when meeting x^i. When such a

node x^i exists among the top few nodes of the branch, we say that $\mathcal{T}$ was *friendly*; indeed, we should simply focus on updating the tree $\mathcal{T}(x^i)$, whose rank is smaller than $\mathcal{T}$ itself, and which will be transformed into a new tree of the same rank.

This fortunate case suggests focusing only on updating trees whose root is safe. Hence, if $\mathcal{T}$ has an unsafe root, a preliminary step will consist in transforming $\mathcal{T}$ into a tree whose root is safe, which we will then update; all subsequent (recursively called) updates will be performed on trees whose root is safe.

If, however, $\mathcal{T}$ has a safe root but is unfriendly, we will choose a deep enough node, say x^ℓ, and transform the sub-tree $\mathcal{T}(x^\ell)$ into a tree $\mathcal{T}'$ whose root is safe, while promoting it (during insertions) or demoting it (during deletions). This transformation will create a zero-edge or a four-node that will be propagated upward and whose propagation will stop when meeting the safe root x; afterward, it just remains to update $\mathcal{T}'$.

Once implemented, these ideas lead to a bottom-up updating algorithm, whose amortised complexity we study now.

▶ **Theorem 5.** *The updating algorithm whose skeleton we just presented, and whose details are presented in Sections 4.2 (for insertions) and 4.3 (for deletions) performs $\mathcal{O}(1)$ amortised write operations per update.*

Proof. Like for Theorem 1, we start from an empty tree, perform n insertion or deletion updates, and keep track of the variations of tree potential while these updates are performed. There is a large enough constant K such that each update requires:

1. optionally, making the tree root safe, which increases the tree potential by at most K;
2. identifying some safe nodes; this requires no write operation and does not change the tree potential;
3. operating on trees small height, i.e., whose height is smaller than a constant $h_{\min}$; this requires performing a constant number of write operations, and increases the tree potential by a at most K;
4. performing a certain number of batches, each of which consists in (i) cut-demoting a node to make it safe and make its ancestor a four-node; (ii) optionally, performing a cut rebalancing for ensuring that some node will stop propagating four-nodes; (iii) $\ell - \mathcal{O}(1)$ transient operations; and (iv) one "terminal" operation where the four-node stop being propagated.

In each batch, steps (i), (ii) and (iv) increase the tree potential by no more than a constant, and each transient operation decreases the tree potential. Thus, each batch decreases the tree potential by $\ell - \mathcal{O}(1)$, i.e., by no more than $\mathsf{L} - \ell$, where L is a constant; it also requires performing $\mathcal{O}(\ell+1)$ write operations. In particular, if $\ell \geqslant \mathsf{L}+1$, and if T batches of operations were performed in total, the potential increased by at most $n\mathsf{K} - \mathsf{T}$; of course, making sure that $\ell \geqslant \mathsf{L}+1$ means that we will need to determine a suitable constant L.

The tree potential started with the value 0, and ended with a non-negative value, which shows that $\mathsf{T} \leqslant 2n\mathsf{K}$. Therefore, in total, these n updates required performing no more than $\mathcal{O}(n) + \mathsf{T} \times \mathcal{O}(\mathsf{L}+1) \subseteq \mathcal{O}(n)$ write operations. ◀

Hence, it remains to:

1. define our notions of safe nodes, both for insertions and for deletions;
2. prove that trees rooted at an unsafe node can be cut promoted to make their root safe, during insertions;
3. prove that trees rooted at an unsafe node can be cut demoted to make their root safe, during deletions;

4. compute the constants hidden in the $\mathcal{O}(1)$ notations above, in order to effectively compute a constant ℓ for which batches decrease the tree potential.

Below, we will prove that choosing $\ell = 48$ during insertions, and $\ell = 126$ during deletions is enough for our purposes: each batch will decrease the tree potential by at least 1, and other operations will increase the tree potential by $\mathcal{O}(\ell)$ per update. These values of ℓ are grossly sub-optimal: for example, choosing $\ell = 15$ during insertions, and $\ell = 20$ during deletions would be enough; computing these better values, although not intrinsically difficult, involves lengthy case-by-case studies that we chose to avoid here.

4.2 Insertion

Below, we say that a node x is *insertion-safe* (or simply *safe*, when the context is clear) when (i) $\mathsf{r}(x) = 1$ and x is a $\{1, 2\}$-node, or (ii) $\mathsf{r}(x) \geqslant 2$ and x is not full. We showed in Section 3.2 that each tree rooted at a full node, i.e., each tree $\mathcal{T}$ of rank $\mathsf{r} \geqslant 2$ whose root is unsafe, can be subject to a cut promotion, resulting in a tree $\mathcal{T}'$ of rank $\mathsf{r} + 1$ whose root is a $\{1, 2\}$-node, thus being safe.

This makes following the recipe of Section 4.1 easy once the desired integer ℓ is chosen. First, if $\mathsf{r}(\mathcal{T}) \leqslant \ell + 2$, just use the bottom-up algorithm of Section 3.3, which may result in a tree $\mathcal{T}'$ of rank $\mathsf{r}(\mathcal{T}') = \mathsf{r}(\mathcal{T}) + 1$ only in case $\mathcal{T}$ was rooted at an unsafe node. Otherwise, $\mathsf{r}(\mathcal{T}) \geqslant \ell + 3$, and:

1. if $\mathcal{T}$ is rooted at an unsafe node, cut-promote it, thus obtaining a tree $\mathcal{T}'$ of rank $\mathsf{r}(\mathcal{T}') \geqslant \ell + 4$ whose root is safe;
2. if $\mathcal{T}$ is friendly, let $i \leqslant \ell$ be the least positive integer for which x^i is safe: we simply focus on inserting the key k in $\mathcal{T}(x^i)$;
3. if $\mathcal{T}$ is unfriendly, the first nodes $x^1, \dots$ on our branch are all unsafe; thus, they are $\{1, 1\}$-nodes, and the branch contains at least ℓ nodes, with the guarantee that $\mathsf{r}(x^\ell) \geqslant \mathsf{r}(\mathcal{T}) - (\ell + 1) \geqslant 2$, so that all nodes $x^1, \dots, x^\ell$ are full; hence we cut-promote $\mathcal{T}(x^\ell)$ and promote all nodes $x^1, \dots, x^{\ell-1}$, thereby creating a zero-edge between x^0 and x^1, which we eliminate by using the cut rebalancing prescribed in Section 3.3.

▶ **Lemma 6.** *Step 3 consists in a batch operation that decreases the tree potential by $\ell - 47$ or more.*

Proof. It suffices to decompose step 3 in atomic operations:

- **i)** a cut promotion, which increases the tree potential by 14 or less;
- **ii)** replacing each of the $\ell - 1$ full nodes $x^1, \dots, x^{\ell-1}$ by a $\{1, 2\}$-node, which decreases the tree potential by $\ell - 1$;
- **iii)** eliminating a zero-edge through a terminal cut rebalancing performed on a cut with 2 to 4 nodes, and optionally followed by a cut demotion; this increases the tree potential by 32 or less. ◀

4.3 Deletion

We will now characterise *deletion-safe* (or simply *safe*, when the context is clear) nodes: a node x should be safe when they can stop propagating four-nodes. There will be three families of safe nodes: (i) *surely safe* nodes, which stop propagating four-nodes even when using the textbook deletion algorithm, (ii) *barely safe* nodes, which are some of the nodes that can be transformed to create surely safe nodes in their close vicinity, and (iii) *simply safe* nodes, which stop propagating four-nodes when using the algorithm of Section 3.4.

Surely safe nodes are the nodes x of rank $\mathsf{r}(x) \geqslant 1$ with profile $(1,1)$, or rank $\mathsf{r}(x) \geqslant 2$ and profile $(2,2,2)$. Indeed, if some child x_i of x is to be replaced by a node of smaller rank, either (i) x had profile $(1,1)$, in which case its new profile is simply $\{1,2\}$, or (ii) x had profiles $\{1,2\}$ and $(2,2,2)$, in which case x_i was the 2-child of x (because the 1-child of x had profile $(1,1)$), and we fall in case 5 of the bottom-up algorithm.

Detecting other kinds of safe nodes is more challenging; in particular, whether a node x is safe may depend on which descendants of x are to be replaced by nodes of smaller rank. More precisely, let $x = x^\ell$ be a node that belongs to the branch $x^0, x^1, \ldots$ below which the key k will be deleted, and let $z = x^{\ell+4}$.

▶ **Lemma 7.** *Each tree $\mathcal{T}$ can be subject to a cut rebalancing, called* cut saving*, that yields a tree $\mathcal{T}'$ of rank $\mathsf{r}(\mathcal{T})$ or $\mathsf{r}(\mathcal{T})-1$ in which z has a surely safe ancestor. This cut saving increases the tree potential by no more than* 38*.*

Proof. Let x be the root of $\mathcal{T}$. If x is surely safe already, there is nothing to do. Hence, we assume that x has profiles $\{1,2\}$ and $\{2,2,3\}$. Replacing its descendants of rank $\mathsf{r}(x)-2$ by their children gives a cut $(\mathsf{A},\mathsf{C},\mathsf{E},\mathsf{G},\mathsf{I})$ with profile $\{3,3,3,3,3\}$, $\{3,3,3,3,4\}$ or $\{3,3,3,4,4\}$.

In the first case, the children of x must have profiles $(1,1)$ and $(2,2,2)$, and once again, there is nothing to do. In the third case, cut-demoting $\mathcal{T}(x)$ transforms it into a tree $\mathcal{T}'$ of rank $\mathsf{r}(x)-1$ whose root is a $\{1,1\}$-node.

It remains to treat the second case. Let y be the node of rank $\mathsf{r}(x)-4$ in the cut $(\mathsf{A},\mathsf{C},\mathsf{E},\mathsf{G},\mathsf{I})$, and let B, D, F and H be the cut ancestors. By symmetry, we assume that y coincides with A, C or E; if $y = \mathsf{E}$, that z descends from E, G or I; and if $y = \mathsf{E}$ and z descends from E, there are no more $\{1,2\}$-nodes among A and C than among G and I. Then, we distinguish ten cases; the tree $\mathcal{T}'$ will be of rank $\mathsf{r}(x)$ in cases 1 to 8, and $\mathsf{r}(x)-1$ in cases 9 and 10:

1. If (i) $y = \mathsf{A}$ or $y = \mathsf{C}$ and (iii) z descends from E, G or I, we rebalance $\mathcal{T}(x)$ as shown in Figure 4 (tree $\mathcal{T}_1$); this makes F a surely safe ancestor of z.
2. If (i) $y = \mathsf{A}$, (ii) C is a $\{1,1\}$-node and (iii) z descends from A or C, we rebalance $\mathcal{T}(x)$ as shown in Figure 4 (tree $\mathcal{T}_1$); this makes B a surely safe ancestor of z.
3. If (i) $y = \mathsf{A}$, (ii) E is a $\{1,1\}$-node (whereas C is not), and (iii) z descends from A or C, we rebalance $\mathcal{T}(x)$ as shown in Figure 4 (tree $\mathcal{T}_2$); this makes C a surely safe ancestor of z.
4. If (i) $y = \mathsf{C}$, (ii) A is a $\{1,1\}$-node, and (iii) z descends from A or C, we rebalance $\mathcal{T}(x)$ as shown in Figure 4 (tree $\mathcal{T}_1$); this makes B a surely safe ancestor of z.
5. If (i) $y = \mathsf{C}$, (ii) E is a $\{1,1\}$-node (whereas A is not), and (iii) z descends from A or C, we rebalance $\mathcal{T}(x)$ as shown in Figure 4 (tree $\mathcal{T}_4$); this makes B a surely safe ancestor of z.
6. If (i) $y = \mathsf{E}$ and (iii) z descends from G or I, we rebalance $\mathcal{T}(x)$ as shown in Figure 4 (tree $\mathcal{T}_5$); this makes H a surely safe ancestor of z.
7. If (i) $y = \mathsf{E}$, (ii) C is a $\{1,1\}$-node, and (iii) z descends from E, we rebalance $\mathcal{T}(x)$ as shown in Figure 4 (tree $\mathcal{T}_5$); this makes D a surely safe ancestor of z.
8. If (i) $y = \mathsf{E}$, (ii) A is a $\{1,1\}$-node (whereas C is not), and (iii) z descends from E, we rebalance $\mathcal{T}(x)$ as shown in Figure 4 (tree $\mathcal{T}_6$); this makes F a surely safe ancestor of z.
9. If (i) $y = \mathsf{A}$ or $y = \mathsf{C}$, (ii) all of A, C and E coincide with y or are $\{1,2\}$-nodes, and (iii) z descends from A or C, we first rebalance $\mathcal{T}(x)$ as shown in Figure 4 (tree $\mathcal{T}_3$). The sub-tree $\mathcal{T}_3(\mathsf{D})$ has profile $\{3,3,3,4,4\}$ and we cut-demote it, before demoting the node F itself; this yields a tree of rank $\mathsf{r}(x)$-1 whose root F is a $\{1,1\}$-node.
10. If (i) $y = \mathsf{E}$, (ii) all of A, C, G and I are $\{1,2\}$-nodes, and (iii) z descends from E, we first rebalance $\mathcal{T}(x)$ as shown in Figure 4 (tree $\mathcal{T}_5$). The sub-tree $\mathcal{T}_5(\mathsf{B})$ has profile $\{3,3,3,4,4\}$ and we cut-demote it, before demoting the node F itself; this yields a tree of rank $\mathsf{r}(x)$-1 whose root F is a $\{1,1\}$-node.

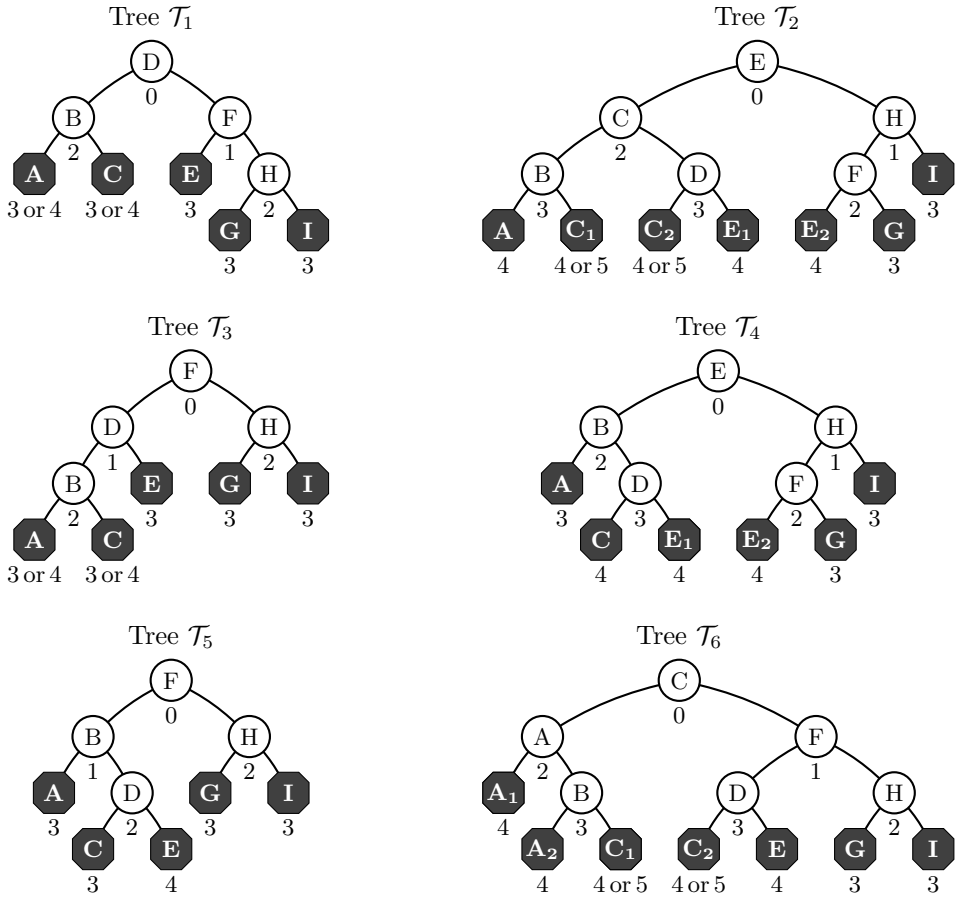


Figure 4 Trees the saving procedure may result in, or use as intermediate step (in cases 9 and 10).

All cases consist in performing a cut rebalancing operation on a cut with at most 7 nodes, optionally followed by a cut demotion. Hence, the saving operation increases the tree potential by no more than 38. ◀

We can now present two additional notions of safety. First, a node x is said to be *barely safe* when the cut saving transforms $\mathcal{T}(x)$ into a tree $\mathcal{T}'(x')$ of rank $\mathsf{r}(x)$. Second, as already mentioned, a node is said to be *simply safe* if its stops propagating four-nodes. Fortunately, deciding whether a node is simply safe is easy, as outlined by the following result.

▶ **Lemma 8.** *Being simply safe depends on local criteria only.*

Proof. Provided that the node $x = x^\ell$ is not surely safe, and if the node $t = x^{\ell+1}$ has just been replaced by a node of smaller rank, the textbook algorithm must now replace $\mathcal{T}(x)$ by a tree $\mathcal{T}'$ whose root x' is a $\{1,1\}$-node with rank $\mathsf{r}(x') = \mathsf{r}(x) - 1$. Furthermore, as can be seen in Figure 2, at least one of its children x'_1 and x'_2 will also be a $\{1,1\}$-node. Hence, we say that x is *m-strong* when the child x'_m is a $\{1,1\}$-node.

Being 1- or 2-strong depends on local criteria only. Indeed, the node x is 1-strong when:

- t coincides with x_1, and x has profile $(1,2)$, $(2,3,2)$, $(2,3,3,3)$ or $(2,3,4,3)$; or
- t coincides with x_2, and x has profile $(3,3,1)$, $(3,3,3,2)$ or $(3,3,4,2)$.

Hence, being 1-strong is a purely local criterion and, symmetrically, so is being 2-strong.

Then, knowing whether t is 1-strong and/or 2-strong allows checking whether x itself is safe, based on other local criteria. For instance, when t coincides with x_2, the node x is safe when it has profile $(1,1)$ or $(2,2,2)$, or in the following cases:

6. $\mathsf{r}(x_{11}) = \mathsf{r}(x_2) = \mathsf{r}-2$ and $\mathsf{r}(x_{12}) = \mathsf{r}-3$, and
  a) x has profile $(3,4,4,3,2)$, $(3,3,4,4,2)$ or $(2,4,5,5,2)$, or
  b) x has profile $(2,4,4,2)$ and x_2 is 1-strong;

7. $\mathsf{r}(x_{11}) = \mathsf{r}-3$ and $\mathsf{r}(x_{12}) = \mathsf{r}(x_2) = \mathsf{r}-2$, and
  a) x has profile $(3,4,4,3,2)$, $(3,3,4,4,2)$, $(4,5,5,4,4,3,2)$, $(4,5,5,4,4,4,2)$, $(4,4,5,5,4,3,2)$, $(4,4,5,5,4,4,2)$, $(3,3,4,5,5,2)$ or $(3,4,4,5,5,2)$, or
  b) x has profile $(3,3,4,4,2)$ or $(3,4,4,4,2)$ and x_2 is 1-strong;

8. $\mathsf{r}(x_1) = \mathsf{r}-2$ and $\mathsf{r}(x_2) = \mathsf{r}-1$, and
  a) x has profile $(3,4,4,1)$, or
  b) x has profile $(3,3,1)$ and x_2 is 1-strong.

These cases were listed by simply enumerating all conditions that could leaf to creating a full node while cut-rebalancing a tree in cases 6 to 8. Hence, some of them are redundant; for instance, in case 7a, if x has profile $(3,3,4,5,5,2)$, it must also have had profile $(3,3,4,4,2)$, which was sufficient for flagging it as safe. Symmetric characterisations exist when t coincides with the left child x_1 of the node x. ◀

Safe nodes and the above saving procedure allow us to implement the recipe succinctly described in Section 4.1. If $\mathcal{T}$ is of rank $\mathsf{r}(\mathcal{T}) \leqslant 2\ell+4$ and its root is barely safe, first use the saving procedure to let a surely safe node appear on the branch below which k will be deleted, and then use the bottom-up algorithm of Section 3.4; this will result in a tree $\mathcal{T}'$ of rank $\mathsf{r}(\mathcal{T}') = \mathsf{r}(\mathcal{T})$. Otherwise, if $\mathcal{T}$ is of rank $\mathsf{r}(\mathcal{T}) \leqslant 2\ell+4$ and its root is *not* barely safe, directly use the bottom-up algorithm of Section 3.4. It may result in a tree $\mathcal{T}'$ of rank $\mathsf{r}(\mathcal{T}') = \mathsf{r}(\mathcal{T}) - 1$ only when $\mathcal{T}$ was rooted at an unsafe node.

We focus now on trees $\mathcal{T}$ of rank $\mathsf{r}(\mathcal{T}) \geqslant 2\ell+5$:

1. If $\mathcal{T}$ is rooted at an unsafe node, the saving procedure transforms it into a tree $\mathcal{T}'$ of rank $\mathsf{r}(\mathcal{T}') \geqslant 2\ell+4$ whose root is safe.

2. If $\mathcal{T}$ is friendly, let $i \leqslant \ell$ be the least positive integer for which x^i is safe: we simply focus on inserting the key k in $\mathcal{T}(x^i)$.

3. If $\mathcal{T}$ is unfriendly, we first consider two sub-cases: if the root x^0 of $\mathcal{T}$ is surely or simply safe, we set $\alpha = x^0$; otherwise, applying the saving algorithm to $\mathcal{T}$ results, without altering the sub-tree $\mathcal{T}(x^4)$, in a tree $\mathcal{T}'$ of rank $\mathsf{r}(\mathcal{T})$ in which x^4 has a surely safe ancestor, the least of which we call α. Then, observing that $\mathsf{r}(x^\ell) \geqslant \mathsf{r}(x) - 2\ell \geqslant 4$, we apply the saving procedure to the sub-tree $\mathcal{T}(x^\ell)$, which makes $x^{\ell-1}$ a four-node. We propagate once this four-node by using the textbook deletion algorithm, i.e., we do not cut-promote full-nodes that might result from using the textbook algorithm; then, we keep propagating our four-node using the algorithm from Section 3.4, a propagation that will stop only when reaching the node α, at which point the four-node is eliminated.

▶ **Lemma 9.** *Step 3 consists in a batch operation that decreases the tree potential by $\ell - 125$ or more.*

Proof. Like for Lemma 4, it suffices to decompose step 3 in atomic operations:

i) optionally, using the saving procedure (cases 1 to 8) on $\mathcal{T}$, which increases the tree potential by no more than 38;

ii) cut-demoting $\mathcal{T}(x^\ell)$ or using the saving procedure (case 9 or 10) on $\mathcal{T}(x^\ell)$, which increases the tree potential by no more than 38;

iii) propagating once the four-node by using the textbook deletion algorithm on $\mathcal{T}(x^{\ell-1})$: this consists in rebalancing a cut with at most 4 nodes, thus increasing the tree potential by no more than 12;

iv) propagating the four-node by using the algorithm of Section 3.4 on nodes $x^{\ell-2}, \dots, x^4$, and stopping just before one meets the surely safe ancestor α of x^4; there are at least $\ell-5$ such propagations, hence the tree potential decreases by at least $\ell - 5$;

v) eliminating the four-node by using the algorithm of Section 3.4 on α; this consists in rebalancing a cut with at most 4 nodes, then possibly performing a cut promotion, and even a node promotion afterward; these three phases, two of which are optional, increase the tree potential by at most 12, 14, and 6, respectively. ◀

References

1 Georgii Adel'son-Velskii and Evgenii Landis. An algorithm for organization of information. In *Doklady Akademii Nauk*, volume 146(2), pages 263–266. Russian Academy of Sciences, 1962.

2 Mahdi Amani, Kevin Lai, and Robert Tarjan. Amortized rotation cost in AVL trees. *Information Processing Letters*, 116(5):327–330, 2016. doi:10.1016/J.IPL.2015.12.009.

3 Norbert Blum and Kurt Mehlhorn. On the average number of rebalancing operations in weight-balanced trees. *Theoretical Computer Science*, 11:303–320, 1980. doi:10.1016/0304-3975(80)90018-3.

4 Leonidas Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 8–21. IEEE Computer Society, 1978. doi:10.1109/SFCS.1978.3.

5 Bernhard Haeupler, Siddhartha Sen, and Robert E Tarjan. Rank-balanced trees. *ACM Transactions on Algorithms (TALG)*, 11(4):1–26, 2015. doi:10.1145/2689412.

6 Thomas Hibbard. Some combinatorial properties of certain trees with applications to searching and sorting. *Journal of the ACM*, 9(1):13–28, 1962. doi:10.1145/321105.321108.

7 Vincent Jugé. Grand-children weight-balanced binary search trees. In *19th International Symposium on Algorithms and Data Structures (WADS)*, volume 349 of *LIPIcs*, pages 14:1–14:20, 2025. doi:10.4230/LIPICS.WADS.2025.14.

8 Tony Lai and Derick Wood. A top-down updating algorithm for weight-balanced trees. *International Journal of Foundations of Computer Science*, 4(4):309–324, 1993. doi:10.1142/S0129054193000201.

9 Jürg Nievergelt and Edward Reingold. Binary search trees of bounded balance. In *4th Annual ACM Symposium on Theory of Computing (STOC)*, pages 137–142. ACM, 1972. doi:10.1145/800152.804906.

10 Hendrik Olivié. *A study of balanced binary trees and balanced one-two trees*. Universitaire Instelling Antwerpen (Belgium), 1980.

11 Hendrik Olivié. A new class of balanced search trees: half-balanced binary search tress. *RAIRO. Informatique théorique*, 16(1):51–71, 1982. doi:10.1051/ita/1982160100511.

12 Daniel Sleator and Robert Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985. doi:10.1145/3828.3835.

Online Hitting Sets for Disks of Bounded Radii

Minati De
Dept. of Mathematics, Indian Institute of Technology Delhi, New Delhi, India

Satyam Singh
Department of Computer Science, Aalto University, Espoo, Finland

Csaba D. Tóth
Department of Mathematics, California State University Northridge, Los Angeles, CA, USA
Department of Computer Science, Tufts University, Medford, MA, USA

Abstract

We present algorithms for the online minimum hitting set problem in geometric range spaces: Given a set P of n points in the plane and a sequence of geometric objects that arrive one-by-one, we need to maintain a hitting set at all times. For disks of radii in the interval $[1, M]$, we present an $O(\log M \log n)$-competitive algorithm. This result generalizes from disks to positive homothets of any convex body in the plane with scaling factors in the interval $[1, M]$. As a main technical tool, we reduce the problem to the online hitting set problem for a finite subset of integer points and bottomless rectangles. Specifically, for a given $N > 1$, we present an $O(\log N)$-competitive algorithm for the variant where P is a subset of an $N \times N$ section of the integer lattice, and the geometric objects are bottomless rectangles.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Online algorithms

Keywords and phrases Geometric Hitting Set, Online Algorithm, Homothets, Disks

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.50

Related Version *Full Version*: https://arxiv.org/abs/2412.04646

Funding *Minati De*: Research on this paper was supported by SERB MATRICS Grant MTR/2021/000584.
Satyam Singh: Research on this paper was supported by the Research Council of Finland, Grant 363444.
Csaba D. Tóth: Research on this paper was supported, in part, by the NSF award DMS-2154347.

1 Introduction

In the general form of the *Hitting Set* problem, we are given a point set P and a collection of subsets $\mathcal{C} = \{S_1, \ldots, S_m\}$, and we need to find a subset $H \subset P$ (**hitting set**) of minimal size such that every set $S_i \in \mathcal{C}$ contains some point in H. In the ***Online*** *Hitting Set* problem, the set P is known in advance, but the subsets $S_1, S_2, \ldots$ in $\mathcal{C}$ arrive one at a time (without advance knowledge). We need to maintain a hitting set $H_i \subseteq P$ for the first i sets $\{S_1, \ldots, S_i\}$ such that $H_i \subseteq H_{i+1}$ for all $i \geq 1$ (that is, we can add new points to the hitting set, but we cannot delete any point). The study of the *Online Hitting Set* problem (which is dual to the *Online Set Cover* problem) was initiated by Alon et al. [2]. They designed a deterministic algorithm with competitive ratio $O(\log |P| \log |\mathcal{C}|)$ and obtained almost matching lower bound of $\Omega\left(\frac{\log |P| \log |\mathcal{C}|}{\log\log |P| + \log\log |\mathcal{C}|}\right)$.

Geometric Hitting Set. In the **geometric** *Hitting Set* problem, we have $P \subseteq \mathbb{R}^d$ for some constant dimension d, and the sets in $\mathcal{C}$ are geometric objects of some type: for example, balls, unit balls, simplices, axis-aligned cubes, or hyper-rectangles. Depending on whether P is finite or infinite, there are different versions of the problem. In this paper, we consider P to be a finite set of points in $\mathbb{R}^2$.


© Minati De, Satyam Singh, and Csaba D. Tóth;
licensed under Creative Commons License CC-BY 4.0



Related Previous Work. When P is finite, Even and Smorodinsky [12] initiated the study of the geometric online *Hitting Set* problem for various geometric objects. They established an optimal competitive ratio of $\Theta(\log |P|)$ when the objects are intervals in $\mathbb{R}$, or half-planes or congruent disks in the plane. Later, Khan et al. [14] investigated this problem for a finite set of integer points $P \subseteq [0, N)^2 \cap \mathbb{Z}^2$ and a collection $\mathcal{C}$ of axis-aligned squares $S \subseteq [0, N)^2$ with integer coordinates for $N > 0$. They developed an $O(\log N)$-competitive algorithm for this variant. They also established a randomized lower bound of $\Omega(\log |P|)$, where $P \subset \mathbb{R}^2$ consists of finitely many points and $\mathcal{C}$ consists of translates of an axis-aligned square. Recently, De et al. [6] considered the variant when P is set of n points in $\mathbb{R}^2$ and $\mathcal{C}$ consists of homothetic copies of a regular k-gon (for $k \geq 4$) with scaling factors in the interval $[1, M]$, and designed an $O(k^2 \log M \log n)$-competitive randomized algorithm. Even though a disk can be approximated by a regular k-gon as $k \to \infty$, this does not imply any competitive algorithm for disks with radii in the interval $[1, M]$.

Our results and Technical Contribution. We study the *Online Hitting Set* problem when P is set of n points in $\mathbb{R}^2$. Table 1 summarizes the existing results and the results of the current paper.

Table 1 Summary of known and new results for the geometric *Online Hitting Set* problem where $|P| = n$ is finite. (#) indicates randomized results. Our results are listed in the last three lines.

Points	Objects	Lower Bound	Upper Bound
$P \subset \mathbb{R}$	Intervals in $\mathbb{R}$	$\Omega(\log n)$ [12]	$O(\log n)$ [12]
$P \subset \mathbb{R}^2$	Half-planes and translates of a disk in $\mathbb{R}^2$	$\Omega(\log n)$ [12]	$O(\log n)$ [12]
$P \subseteq [0, N)^2 \cap \mathbb{Z}^2$	Axis-aligned squares in $[0, N)^2 \cap \mathbb{Z}^2$ with integer coordinates	$\Omega(\log n)$ [14] (#)	$O(\log N)$ [14]
$P \subset \mathbb{R}$	Homothetic copies of a regular k-gon ($k \geq 4$) with scaling factors in the interval $[1, M]$	$\Omega(\log n)$ [14] (#)	$O(k^2 \log M \log n)$ [6] (#)
$P \subseteq [0, N)^2 \cap \mathbb{Z}^2$	Bottomless rectangles (for definition, see Section 2.1)	$\Omega(\log n)$ [12]	$O(\log N)$ [Theorem 1]
$P \subset \mathbb{R}^2$	Disks having radii in the interval $[1, M]$	$\Omega(\log n)$ [12]	$O(\log M \log n)$ [Theorem 12]
$P \subset \mathbb{R}^2$	Positive homothets of an arbitrary convex body in $\mathbb{R}^2$ with scaling factors in the interval $[1, M]$	$\Omega(\log n)$ [14] (#)	$O(\log M \log n)$ [Theorem 14]

We now present our contributions and briefly discuss the technical ideas involved.

Bottomless Rectangles in $[0, N]^2$. We present an $O(\log N)$-competitive deterministic algorithm for the geometric *Online Hitting Set* problem, where $P \subset [0, N)^2 \cap \mathbb{Z}^2$, and $\mathcal{C}$ is a sequence of bottomless rectangles of the form $[a, b) \times [0, c)$ arriving one by one (Theorem 1 in Section 2). When a bottomless rectangle $[a, b) \times [0, c)$ arrives, our algorithm chooses hitting points guided by the **canonical partition** of the interval $[a, b]$ (see Section 2 for a definition). For each point p in an offline optimum, this structured canonical partition ensures that $O(\log N)$ points are sufficient to hit all the incoming rectangles in $[0, N]^2 \cap \mathbb{Z}^2$ that are hit by p. We prove that our algorithm is $O(\log N)$-competitive for a broader class of objects– sets $S \subset [a, b) \times \mathbb{R}$ with **lowest-point property** (see Section 2.2 for a definition).

Disks with Radii in $[1, M]$. Our main result is a deterministic $O(\log M \log n)$-competitive *Online Hitting Set* algorithm for an arbitrary set P of n points in the plane, and a sequence of disks of radii in the interval $[1, M]$ (Theorem 12 in Section 4). Previously, an $O(\log n)$-competitive algorithm was known only for congruent disks [12]. In particular, our result is the first $O(\log n)$-competitive algorithm that works for disks of radii in the interval $[1, 1+\varepsilon]$ for any constant $\varepsilon > 0$ (Corollary 13).

However, a finite set of disks in the plane do not necessarily have the lowest-point property. We reduce the problem to objects with the lowest-point property in two steps. First, we consider a restricted version, the **line-separated setting** (Section 3), where the centers of disks in $\mathcal{C}$ lie on one side of a line (w.l.o.g., the x- or y-axis), while P lies on the other side. We use the concept of **disk hull** for a point set (introduced by Dumitrescu et al. [10]), which generalizes the notion of convex hulls and α-hulls. Among other important properties, the boundary of the disk hull is monotone w.r.t. the separating line. Using these properties, we reduce the *Hitting Set* problem in the line-separated setting to objects with the lowest-point property, and obtain an $O(\log n)$-competitive algorithm in the line-separated setting (Theorem 9 in Section 3).

In general, there is no restriction on the location of the points in P and the centers of disks. We reduce the general problem to the line-separated setting as follows: We partition the disks of radii in the interval $[1, M]$ into $O(\log M)$ layers, ensuring that the ratio of radii of disks in each layer is bounded by at most 2. For each layer, our algorithm maintains a tiling of the plane into axis-aligned squares such that (a) any disk of a given layer contains the entire tile that contains the disk center, and (b) each disk intersects only $O(1)$ tiles. Our algorithm simultaneously runs several invocations of the line-separating algorithm (one for each directed grid line). When a disk arrives, our algorithm inserts it into all relevant invocations of the line-separating algorithms; we show that only $O(1)$ invocations are relevant. In the competitive analysis, we show that for each point p in an offline optimum solution, our algorithm uses $O(\log n)$ hitting points for the disks in each layer that contain p. Since there are $O(\log M)$ layers, our algorithm is $O(\log M \log n)$-competitive.

Homothets of a Convex Body with Diameters in $[1, M]$. We generalize our main result from disks to positive homothets of any convex body in the plane, where the radii in the interval $[1, M]$ are replaced by scaling factors in the interval $[1, M]$ (Theorem 14 in Section 5). Our online algorithm is based on a two-stage approach, similar to the case of disks, and it is $O(\log M \log n)$-competitive. The key technical difficulty arises from the geometric differences between a disk and a general convex body. It is easy to extend the concept of a disk hull to hulls for homothetic convex bodies. However, unlike for disks, the boundary of the hull is not necessarily x- or y-monotone: We show that it is monotone w.r.t. some carefully chosen directions. To generalize a layered decomposition of axis-parallel lines, we need **two** directions in which the hull is monotone, the two directions must be far apart (in the space of directions), to create a tiling with properties (a) and (b) above. We call a pair of directions satisfying these requirements a **good pair** of directions. We use a careful geometric argument, which heavily relies on convexity, a suitable affine transformation, and the variational method (i.e., the intermediate value theorem) to prove that every convex body in the plane admits a good pair of directions.

1.1 Further Related Work

When the point set P is infinite, one may further distinguish between the **continuous** setting where $P = \mathbb{R}^d$ (also known as the **piercing problem**) and the **discrete** setting where P is a discrete subset of $\mathbb{R}^d$ (for example, $P = \mathbb{Z}^d$).

Continuous Setting. In the geometric setting, the duality between the *Hitting Set* problem and the *Set Cover* problem only holds when the objects are translates of a convex body [5, Theorem 2]. Hence the results obtained for the *Set Cover* problem for translates of a convex body also hold for the *Hitting Set* problem. Charikar et al. [3] studied the *Online Set Cover* problem for translates of a ball. They proposed an algorithm with a competitive ratio $O(2^d d \log d)$. They also proved $\Omega(\log d / \log\log\log d)$ as the deterministic lower bound of the competitive ratio for this problem. Dumitrescu et al. [9] improved the bounds on the competitive ratio for translates of a ball, establishing an upper bound of $O(1.321^d)$ and a lower bound of $\Omega(d+1)$. For translates of a centrally symmetric convex body, they proved that the competitive ratio of every deterministic algorithm is at least $I(s)$, where $I(s)$ is the illumination number of the object s[1]. For translates of an axis-aligned hypercube in $\mathbb{R}^d$, Dumitrescu and Tóth [11] proved that the competitive ratio of any deterministic algorithm for *Online Set Cover* is at least 2^d. Later, De et al. [5] studied the *Online Hitting Set* problem for α-fat objects in $\mathbb{R}^d$ with diameters in $[1, M]$ and designed a deterministic algorithm with competitive ratio $O\left((2+\frac{2}{\alpha})^d \log M\right)$. For hitting axis-aligned homothetic hypercubes with side lengths in $[1, M]$, they gave a deterministic algorithm with competitive ratio at most $3^d\lceil \log_2 M\rceil + 2^d$. They also proved a $\Omega(d \log M + 2^d)$ lower bound for the problem of hitting homothetic hypercubes in $\mathbb{R}^d$, with side lengths in the interval $[1, M]$.

Discrete Setting. De and Singh [7, 8] studied a variant of this problem where $P = \mathbb{Z}^d$ and $\mathcal{C}$ consists of translates of a ball or an axis-aligned hypercube in $\mathbb{R}^d$. For translates of an axis-aligned hypercube, they showed that there is a randomized algorithm with an expected competitive ratio of $O(d^2)$ and also proved that every deterministic algorithm has a competitive ratio of at least $d+1$. For translates of a ball in $\mathbb{R}^d$, they proposed a deterministic algorithm having a competitive ratio of $O(d^4)$ and proved that every deterministic algorithm has a competitive ratio of at least $d+1$, for $d \leq 3$. Recently, Alefkhani et al. [1] considered the variant where $P = (0, N)^d \cap \mathbb{Z}^d$ and $\mathcal{C}$ is a family of α-fat objects in $(0, N)^d$, for some constant $\alpha > 0$. They proposed a deterministic algorithm with a competitive ratio of at most $(\frac{4}{\alpha}+1)^{2d} \log N$, and proved that the competitive ratio of every deterministic algorithm is $\Omega\left(\frac{\log N}{1+\log \alpha}\right)$. Very recently, De et al. [6] improved both the upper and lower bounds of Alefkhani et al. [1]. They considered the case where $P = \mathbb{Z}^d$ and $\mathcal{C}$ is a family of α-fat objects having diameters in $[1, M]$, for some constant $\alpha > 0$. They proposed a deterministic algorithm with competitive ratio $O((\frac{2}{\alpha})^d \log M)$, and established that the competitive ratio of any randomized algorithm is $\Omega(d \log M)$.

2 Bottomless Rectangles and Integer Points

We present an $O(\log N)$-competitive algorithm for the *Online Hitting Set* problem where P is a subset of an $N \times N$ section of the integer lattice, and the objects are *bottomless rectangles* (Section 2.1); and then generalize the algorithm for the same point set but with objects that have the *lowest-point property* (Section 2.2).

[1] The **illumination number** of an object s, denoted by $I(s)$, is the minimum number of smaller homothetic copies of s (λs, where $\lambda \in (0, 1)$) whose union contains s.

2.1 Bottomless Rectangles

In this section we present an $O(\log N)$-competitive algorithm for the *Online Hitting Set* problem where P is a subset of the integer lattice with nonnegative coordinates less than N, that is, $P \subset [0, N)^2 \cap \mathbb{Z}^2$; and the objects are bottomless rectangles. **Bottomless rectangles** are of the form $r_i = [a_i, b_i) \times [0, c_i)$, where $0 \le a_i < b_i \le N$ and $0 \le c_i \le N$. Note that there are only $O(N^3)$ combinatorially different rectangles w.r.t. P, so the general result by Alon et al. [2] gives an algorithm for the online hitting set with competitive ratio $O(\log^2 N)$. In this section, we present an $O(\log N)$-competitive algorithm, which is the best possible (a matching lower bound follows from the lower bound for the *Online Hitting Set* problem for intervals in one-dimension [12]).

Preliminaries. We need some preparation before we can present the algorithm. We may assume w.l.o.g. that N is a power of 2, and every bottomless rectangle $r_i = [a_i, b_i) \times [0, c_i)$ is given with integer parameters a_i, b_i, and c_i. An interval I is **canonical** if it is of the form $I = \big[q2^j, (q+1)2^j\big)$ for some integers $q, j \ge 0$. For a canonical interval $I = \big[q2^j, (q+1)2^j\big)$, we also define the **left neighbor** $L(I) = \big[(q-1)2^j, q2^j\big)$ and the **right neighbor** $R(I) = \big[(q+1)2^j, (q+2)2^j\big)$. For every canonical interval I, if $(I \times [0, N)) \cap P \neq \emptyset$, then let $p(I)$ denote a **lowest-point** in $(I \times [0, N)) \cap P \neq \emptyset$ (that is, a point with minimum y-coordinate; ties are broken arbitrarily). If $(I \times [0, N)) \cap P = \emptyset$, then $p(I)$ is undefined.

For every interval $[a, b)$ with nonnegative integer endpoints, we define a **canonical partition**, i.e., a partition of $[a, b)$ into canonical intervals. This partition is standard – we walk through some of the technical details because we need them for our algorithm and its analysis. Let $j \ge 0$ be the largest integer such that $q2^j \in (a, b)$, for some $q \in \mathbb{Z}$. (Note that $q \in \mathbb{Z}$ is unique. Indeed, suppose that q is not unique, say $q2^j, (q+1)2^j \in (a, b)$. Since q or $q+1$ is even, then $q/2$ or $(q+1)/2$ is an integer. Now, we have $\frac{q}{2}2^{j+1}$ or $\frac{q+1}{2}\,2^{j+1} \in (a, b)$, which contradicts the maximality of j.) We call the integer $s_{[a,b)} := q2^j$ the **splitting point** of $[a, b)$. We can partition a given interval $[a, b)$ into canonical intervals as follows. If $[a, b)$ is not canonical, find its splitting point $s = s_{[a,b)}$, partition it into two intervals $[a, b) = [a, s) \cup [s, b)$, and recurse on $[a, s)$ and $[s, b)$. For example, the splitting point of interval $[5, 11)$ is 8, and its canonical partition is $[5, 11) = [5, 6) \cup [6, 8) \cup [8, 10) \cup [10, 11)$; see Figure 1a) for an illustration.

Note also that in the canonical partition of $[a, s)$ (resp., $[s, b)$), there is at most one interval of each size, where the possible sizes are powers of 2 between 1 and $s - a$ (resp., $b - s$). Specifically, if I is in the canonical partition of $[a, s)$, then its left neighbor $L(I)$ is not contained in $[a, b)$, consequently $a \in \overline{L(I)}$, where $\overline{L(I)}$ is the closure of $L(I)$. Similarly, if I is in the canonical partition of $[s, b)$, then $b \in R(I)$.

Online algorithm ALG for bottomless rectangles. We can now present our online algorithm. We maintain a hitting set $H_i \subseteq P$, which is initially empty: $H_0 = \emptyset$. When the i-th bottomless rectangle $r_i = [a_i, b_i) \times [0, c_i)$ arrives, initialize $H_i := H_{i-1}$. If $r_i \cap H_i \neq \emptyset$, then do not add any new points to H_i. Otherwise, we may assume that $r_i \cap H_i = \emptyset$. Compute the splitting point s_i of $[a_i, b_i)$, and the canonical partitions $\mathcal{A}_i$ and $\mathcal{B}_i$ of $[a_i, s_i)$ and $[s_i, b_i)$, respectively. If $([a_i, s_i) \times [0, c_i)) \cap P \neq \emptyset$, then find the largest canonical interval $I \in \mathcal{A}_i$ such that $p(I) \in r_i$, and put $H_i := H_i \cup \{p(I)\}$. Similarly, if $([s_i, b_i) \times [0, c_i)) \cap P \neq \emptyset$, then find the largest interval $I \in \mathcal{B}_i$ such that $p(I) \in r_i$, and put $H_i := H_i \cup \{p(I)\}$. Overall, we add at most two new points to H_i in step i.

Competitive analysis. We now prove that ALG is $O(\log N)$-competitive.

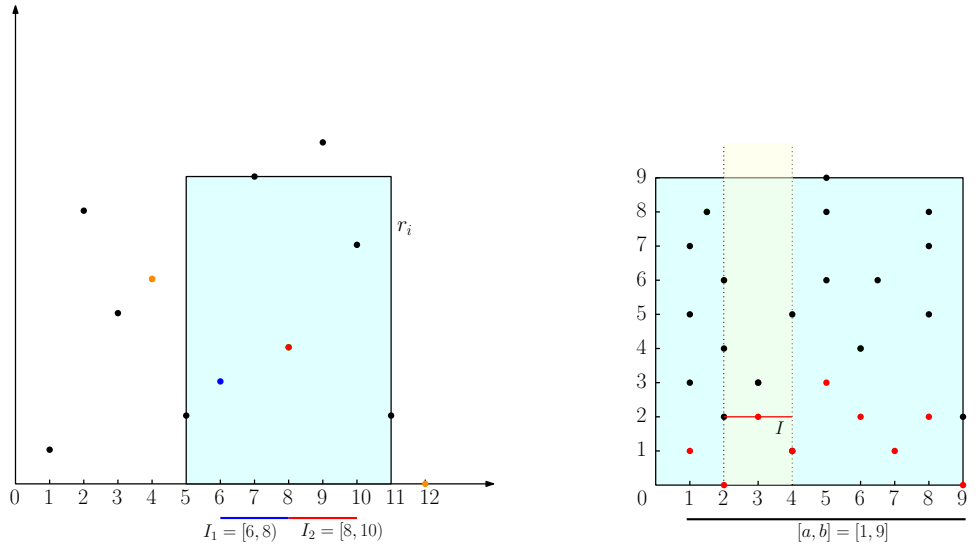


Figure 1 (a) When the ith bottomless rectangle $r_i = [5, 11) \times [0, c_i)$ arrives, suppose that the hitting set H_i contains the orange points, and $r_i \cap H_i = \emptyset$. The splitting point of $[5, 11)$ is 8, with canonical partitions $\mathcal{A}_i = [5, 6) \cup [6, 8)$ and $\mathcal{B}_i = [8, 10) \cup [10, 11)$, respectively. Here, $I_1 = [6, 8) \subset \mathcal{A}_i$ and $I_2 = [8, 10) \subset \mathcal{B}_i$ are the largest canonical intervals in $\mathcal{A}_i$ and $\mathcal{B}_i$, respectively. The blue (resp., red) point is the lowest-point in $I_1 \times [0, N) \cap P$ (resp., $I_2 \times [0, N) \cap P$). We add both points to H_i. (b) The black and red colored points denote the set P, while the red colored points denote the set S. The yellow strip denotes $I \times \mathbb{R}$.

▶ **Theorem 1.** *For the Online Hitting Set problem for a point set $P \subseteq [0, N)^2 \cap \mathbb{Z}^2$ and a sequence of bottomless rectangles, the online algorithm ALG has competitive ratio $O(\log N)$.*

Proof. Let $\mathcal{C}$ be a sequence of bottomless rectangles. Let H and OPT be the hitting set returned by the online algorithm ALG and an (offline) minimum hitting set of $\mathcal{C}$, respectively. For a point $p \in$ OPT, let $\mathcal{C}_p$ be the subsequence of bottomless rectangles that contain p. It is enough to show that for every $p \in$ OPT, our algorithm adds $O(\log N)$ points to H in response to the objects in $\mathcal{C}_p$.

Let $p \in$ OPT, with coordinates $p = (p_x, p_y)$; and let $r_1, \ldots, r_m$ be a sequence of bottomless rectangles in $\mathcal{C}_p$ for which our algorithm adds new points to the hitting set. We show that $m = O(\log N)$. We can distinguish between two types of rectangles $r_i = [a_i, b_i) \times [0, c_i)$ depending on whether the x-coordinate p_x of p is on the left or right of the splitting point s_i of $[a_i, b_i)$: namely, $p_x < s_i$ or $s_i \leq p_x$. We analyze the two types separately (the two cases are analogous).

Assume w.l.o.g. that $p_x < s_i$ for $i = 1, \ldots, m$. This means that $p \in [a_i, s_i) \times [0, c_i)$, and so ALG adds the hitting point $p(I)$ for exactly one interval $I \in \mathcal{A}_i$. Suppose that the algorithm adds the hitting point $p(I)$ for $I \in \mathcal{A}_i$. Then I is the largest (hence rightmost) canonical interval in $\mathcal{A}_i$ such that $(I \times [0, c_i)) \cap P \neq \emptyset$. Recall that $a_i \in \overline{L(I)}$, where $L(I)$ is the left neighbor of the canonical interval I. This implies that $p_x \in L(I) \cup I$. That is, either I or its left neighbor $L(I)$ contains p_x. Note that p_x is contained in $\log N$ canonical intervals (one for each possible size), and each of these canonical intervals has a unique right neighbor. Consequently, I is one of at most $2 \log N$ canonical intervals under the assumption that $p_x < s_i$ for all $i = 1, \ldots, m$. This proves that $m \leq 4 \log N$. ◀

2.2 Objects with the lowest-point property

In this section, we generalize Theorem 1 to a broader class of objects. Similarly to Section 2.1, let $P \subseteq [0, N)^2 \cap \mathbb{Z}^2$. For a set $S \subseteq P$, the span of S, denoted $\text{span}(S)$ is the smallest interval $[a, b)$ with integer endpoints $a, b \in \mathbb{Z}$ such that $S \subset [a, b) \times \mathbb{R}$. An object $S \subseteq P$ has the **lowest-point property** if for every point $s = (s_x, s_y)$ in S and every interval $I \subset \text{span}(S)$ that contains s_x, the object S contains all points in $P \cap (I \times \mathbb{R})$ with the minimum y-coordinates. For an illustration of set S see Figure 1b. Note, in particular, that every bottomless rectangle $r_i = [a_i, b_i) \times [0, c_i)$ has the lowest-point property: Indeed, if $s_x \in I \subset [a_i, b_i)$, then $I \times [0, s_y] \subset r_i$.

Our online hitting set algorithm and its analysis readily generalize when the objects have the lowest-point property. Let $\mathcal{C} = (S_1, \ldots, S_m)$ be a sequence of objects with the lowest-point property.

Online algorithm ALG_0 for objects with the lowest-point property. We maintain a hitting set $H_i \subseteq P$, which is initially empty: $H_0 = \emptyset$. When set S_i arrives, initialize $H_i := H_{i-1}$. If $S_i \cap H_i \neq \emptyset$, then do not add any new points to H_i. Suppose that $S_i \cap H_i = \emptyset$. Let $[a_i, b_i) = \text{span}(S_i)$. Compute the splitting point s_i of $[a_i, b_i)$, and the canonical partitions $\mathcal{A}_i$ and $\mathcal{B}_i$ of $[a_i, s_i)$ and $[s_i, b_i)$, respectively. If $([a_i, s_i) \times \mathbb{R}) \cap S_i \cap P \neq \emptyset$, then find the largest interval $I \in \mathcal{A}_i$ such that $p(I) \in S_i$, and put $H_i := H_i \cup \{p(I)\}$. Similarly, if $([s_i, b_i) \times \mathbb{R}) \cap S_i \cap P \neq \emptyset$, then find the largest interval $I \in \mathcal{B}_i$ such that $p(I) \in S_i$, and put $H_i := H_i \cup \{p(I)\}$. Overall, we add at most two new points to H_i in step i.

Correctness and competitive analysis. When ALG_0 adds a points $p(I)$ to H_i in step i, the lowest-point property ensures that $p(I) \in S_i$. Therefore, ALG_0 maintains that H_i is a hitting set for $\{S_1, \ldots, S_i\}$, proving the correctness of ALG_0. We now show that ALG_0 is $O(\log N)$-competitive.

▶ **Theorem 2.** *For the Online Hitting Set problem for a point set $P \subset [0, N]^2 \cap \mathbb{Z}^2$ and a sequence $\mathcal{C} = (S_1, \ldots, S_m)$ of objects with the lowest-point property, algorithm ALG_0 has competitive ratio $O(\log N)$.*

Proof. Let $\mathcal{C}$ be a sequence of objects with the lowest-point property. Let H and OPT be the hitting set returned by the online algorithm ALG_0 and an (offline) minimum hitting set of $\mathcal{C}$, respectively. For each $p \in \mathsf{OPT}$, let $\mathcal{C}_p$ the subsequence of sets in $\mathcal{C}$ that contain p. It is enough to show that for every $p \in \mathsf{OPT}$, our algorithm adds $O(\log N)$ points to H in response to the objects in $\mathcal{C}_p$.

Let $p \in \mathsf{OPT}$, with coordinates $p = (p_x, p_y)$; and let $S_1, \ldots, S_m$ be a sequence of sets in $\mathcal{C}_p$ for which our algorithm adds new points to the hitting set. We show that $m = O(\log N)$. We can distinguish between two types of sets S_i depending on whether the x-coordinate p_x of p is to the left or right of the splitting point s_i: namely, $p_x < s_i$ or $s_i \leq p_x$. We analyze the two types separately (the two cases are analogous).

Assume w.l.o.g. that $p_x < s_i$ for $i = 1, \ldots, m$. This means that $p \in [a_i, s_i) \times \mathbb{R}$, and so ALG adds the hitting point $p(I)$ for exactly one interval $I \in \mathcal{A}_i$. Suppose that the algorithm adds the hitting point $p(I)$ for $I \in \mathcal{A}_i$. Then I is the largest (hence rightmost) canonical interval in $\mathcal{A}_i$ such that $(I \times \mathbb{R}) \cap P \neq \emptyset$. Recall that $a_i \in \overline{L(I)}$, where $L(I)$ is the left neighbor of the canonical interval I. This implies that $p_x \in L(I) \cup I$. That is, either I or its left neighbor $L(I)$ contains p_x. Note that p_x is contained in $\log N$ canonical intervals (one for each possible size), and each of these canonical intervals has a unique right neighbor. Consequently, I is one of at most $2 \log N$ canonical intervals under the assumption that $p_x < s_i$ for all $i = 1, \ldots, m$. This proves that $m \leq 4 \log N$. ◀

3 Disks in the Plane: Separated Setting

In this section, we consider the *Online Hitting Set* problem in the plane, where P is a finite set above the x-axis (given in advance); and $\mathcal{C}$ consists of disks of arbitrary radii with centers located on or below the x-axis (arriving one-by-one). Note that the disks in $\mathcal{C}$ do not necessarily have the lowest-point property; see Figure 2.

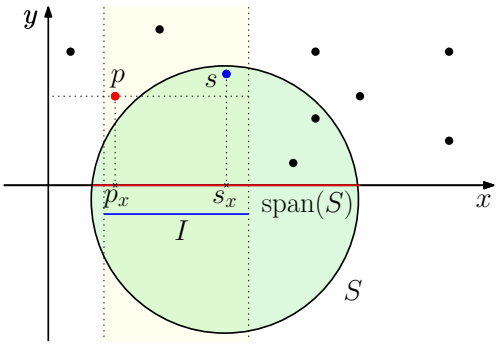


Figure 2 Disk S with center below the x-axis does not have the lowest-point property: We have $s \in S$, and the interval $I \subset \text{span}(S)$ contains s_x, but S does not contain the point $p \in P$ with minimum y-coordinate in the strip $I \times \mathbb{R}$.

Disk hulls for a point set w.r.t. disks and its properties. The unit disk hull of a point set was introduced by Dumitrescu et al. [10] as an analogue of the convex hull. Recall that the convex hull $\text{conv}(P)$ of a point set $P \subset \mathbb{R}^2$ is the smallest convex set in the plane that contains P. Equivalently, it is the intersection of all closed half-planes that contain P; it can be computed by the classical "rotating calipers" algorithm, where we continuously rotate a line ℓ around P while P remains in one closed half-plane bounded by ℓ. Intuitively, we obtain the unit disk hull of P by rolling a unit disk, with center on or below the x-axis, around P. We generalize this notion to disks of any fixed radius $t > 0$.

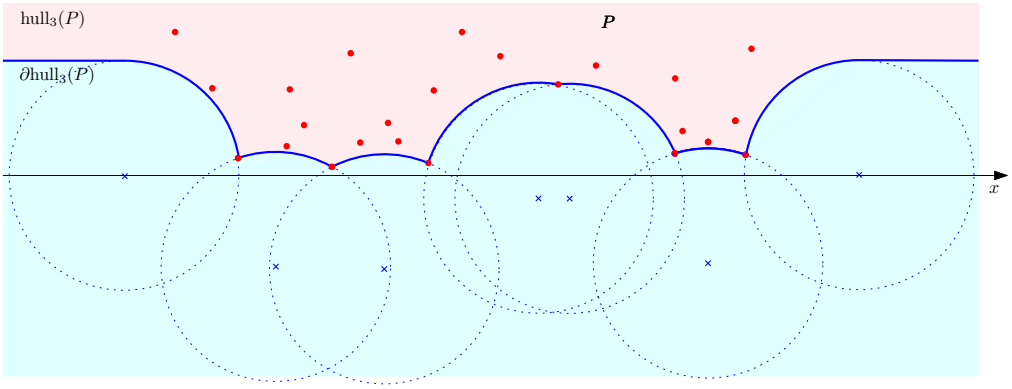


Figure 3 A point set P (red) and region $\text{hull}_3(P)$ (pink). The boundary $\partial\text{hull}_3(P)$ is composed of horizontal lines and circular arcs.

▶ **Definition 3.** *Let $P \subset \mathbb{R}^2$ be a finite set of points above the x-axis and let $t > 0$. Let $\mathcal{D}_t$ be the set of all disks of radius t with centers on or below the x-axis. Let $M_t(P)$ be the union of all disks $D \in \mathcal{D}_t$ such that $P \cap \text{int}(D) = \emptyset$. Now, we define the* ***t-hull*** *of P as $\text{hull}_t(P) = \mathbb{R}^2 \setminus \text{int}(M_t(P))$. The boundary of $\text{hull}_t(P)$ is denoted by $\partial\text{hull}_t(P)$; for an illustration see Figure 3.*

Dumitrescu et al. [10, Lemma 4] proved that $\partial\text{hull}_t(P)$ is x-monotone[2] for any $t > 0$, and established other properties, which were used by Conroy and Tóth [4], as well.

▶ **Lemma 4** (Dumitrescu et al. [10]). *For a finite set $P \subset \mathbb{R}^2$ above the x-axis and $t > 0$, the following holds:*

1. *$\partial\text{hull}_t(P)$ lies above the x-axis;*
2. *every vertical line intersects $\partial\text{hull}_t(P)$ in one point, thus $\partial\text{hull}_t(P)$ is an x-monotone curve;*
3. *for every disk $D \in \mathcal{D}_t$, the intersection $D \cap (\partial\text{hull}_t(P))$ is connected (possibly empty);*
4. *for every disk $D \in \mathcal{D}_t$, if $P \cap D \neq \emptyset$, then $P \cap D$ contains a point in $\partial\text{hull}_t(P)$.*

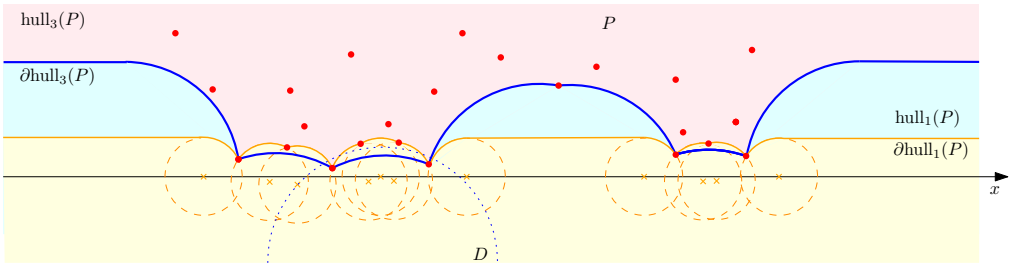


Figure 4 A point set P (red), hull$_3(P)$ (pink), and hull$_1(P)$ (light blue or pink). A disk $D \in \mathcal{D}_3$ of radius 3 (dotted blue), where the intersection $D \cap (\partial\text{hull}_1(P))$ has two components.

Since we consider the case of disks with bounded radii, for our purposes, we need to compare two disk hulls for the same point set P w.r.t. different radii; see Figure 4. We start with an easy observation.

▶ **Lemma 5.** *Let γ_1 and γ_2 be circular arcs lying entirely above the x-axis, such that γ_1 and γ_2 are arcs of circles C_1 and C_2, resp., of radii r_1 and r_2, with centers on or below the x-axis.*

1. *Then both γ_1 and γ_2 are x-monotone and concave curves.*
2. *Furthermore, if points $p_1, p_2 \in \mathbb{R}^2$ are contained in both γ_1 and γ_2, and $r_1 < r_2$, then γ_1 lies above γ_2 (i.e., for every vertical line L that separates p_1 and p_2, point $\gamma_1 \cap L$ lies above point $\gamma_2 \cap L$).*

Proof. (1) For every $i \in \{1, 2\}$, the center of C_i is below the x-axis, and so the leftmost and rightmost points of C_1 are also below the x-axis. The leftmost and rightmost points partition C_i into two halfcircles, one above the center and one below the center. Both halfcircles are x-monotone: The lower halfcircle is convex curve and the upper halfcircle is concave. Since γ_i lies entirely above the x-axis, it is contained in the upper halfcirlce, which is x-monotone and concave.
(2) The locus of centers of circles that contain both p_1 and p_2 is the orthogonal bisector of the line segment p_1p_2, that we denote by $(p_1p_2)^\perp$. Note that p_1p_2 is not vertical (or else $(p_1p_2)^\perp$ would be a horizontal line above the x-axis, and the centers of C_1 and C_2 would also be above the x-axis). As the center a circle containing p_1 and p_2 continuously moves from the center of C_1 down to $y = -\infty$, the circular arc between p_1 and p_2 deforms continuously from γ_1 to the line segment p_1p_2. Since γ_1 concave, it lies above the segment p_1p_2. Since $r_1 < r_2$, the arc γ_2 lies between the arc γ_1 and the segment p_1 and p_2. Consequently, then γ_2 lies below γ_1, as claimed. ◀

[2] A curve in the plane is x-monotone if every vertical line intersects it at most once.

▶ **Lemma 6.** *For every finite set $P \subset \mathbb{R}^2$ above the x-axis, the following holds:*

1. *If $0 < s < t$, then for every disk $D \in \mathcal{D}_s$ of radius s, the intersection $D \cap (\partial\text{hull}_t(P))$ is connected (possibly empty).*
2. *Suppose that $p \in P$ lies on the curve $\partial\text{hull}_t(P)$ for some $t > 0$. Then there is a radius $r_p \in (0, t)$ such that p is also on $\partial\text{hull}_s(P)$ for all $s \in [r_p, t]$, but p is below $\partial\text{hull}_s(P)$ for all $s \in [0, r_p)$.*

Proof. (1) Let $D \in \mathcal{D}_s$. Suppose, to the contrary, that the intersection $D \cap (\partial\text{hull}_t(P))$ has two or more components. By Lemma 4(2), the x-coordinates of the components form disjoint intervals, and the components have a natural left-to-right ordering. Let q_1 be the rightmost point in the first component, and let q_2 be the leftmost point in the second component. Clearly $q_1, q_2 \in \partial D$. Let q' be an arbitrary point in $\partial\text{hull}(A)$ between q_1 and q_2. Then q' lies on the boundary of some disk D' of radius t whose center is below the x-axis, and whose interior is disjoint from P. In particular, neither q_1 nor q_2 is in the interior of D'. Since the center of D' is below the x-axis, $\partial D'$ contains two interior-disjoint circular arcs between q and the x-axis; and both arcs must cross ∂D. We have found two intersection points $p_1, p_2 \in \partial D \cap \partial D'$ above the x-axis. Furthermore, between p_1 and p_2, the circular arc ∂D lies above the circular arc $\partial D'$, contradicting Lemma 5(2). This completes the proof of Property 1.
(2) Consider a point $p \in P$ that lies on the curve $\partial\text{hull}_t(P)$ for some $t > 0$. Then there exists a disk $D \in \mathcal{D}_t$ of radius t centered at some point c below the x-axis such that $p \in \partial D$. Let c_1 be the intersection point of the x-axis the line cp, and c_2 the orthogonal projection of p to the x-axis. We describe two continuous motions, where the disk D continuously changes while p is in the circle ∂D and there is no point in P in the interior of D: First, a central dilation from center p continuously moves D to a disk D_1 centered at c_1. Second, the center of D moves from c_1 towards c_2 continuously until its center reaches c_2 or a point c_3 where ∂D contains both p and another point $p' \in P$. Let r_p be radius of D at that time. The continuous motion shows that $p \in \partial\text{hull}_s(P)$ for all $s \in [r_p, t]$, but it is not in $\partial\text{hull}_s(P)$ for all $s < r_p$. ◀

Note that Lemma 6(1) is not symmetric for $s < t$: For a disk $D \in \mathcal{D}_t$ of radius t, the intersection $D \cap (\partial\text{hull}_s(P))$ is not necessarily connected; see Figure 4 for an example.

Reduction. We can reduce the *Online Hitting Set* problem for a finite set $P \subset \mathbb{R}^2$ and disks of bounded radii in the separated setting, to the *Online Hitting Set* problem for a finite subset of integer points and objects with the lowest-point property. We achieve the reduction in two steps:
(1) We choose a subset $Q \subseteq P$ of points that are relevant for a hitting set (Lemma 7); and
(2) we map the points in P into a set of integer points $P' \subset [0, n]^2 \cap \mathbb{Z}^2$ (Lemma 8).

For a finite point set P in the plane above the x-axis, let $Q = Q(P)$ be the set of points $p \in P$ such that $p \in \partial\text{hull}_t(P)$ for some $t > 0$.

▶ **Lemma 7.** *For a finite point set P in the plane above the x-axis, $Q = Q(P)$ has the following property: For every disk D centered below the x-axis, if $D \cap P \neq \emptyset$, then $D \cap Q \neq \emptyset$.*

Proof. Let D be a disk of radius $t > 0$ centered below the x-axis. By Lemma 4(4), $D \cap P$ contains a point in $\partial\text{hull}_t(P)$. By the definition of Q, this point is in Q. ◀

We may assume that the points in P have distinct x-coordinates (if two or more points in P have the same x-coordinate, w.l.o.g. a minimum hitting set would contain only the point with the smallest y-coordinate). Sort P by increasing x-coordinates such that $P = \{p_0, \ldots, p_{n-1}\}$.

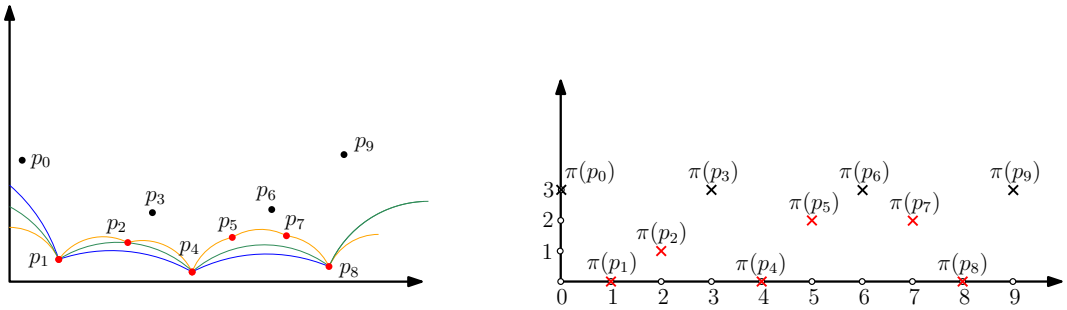


Figure 5 Description of the function π. Left: $\partial\text{hull}_1(P)$ is orange, $\partial\text{hull}_2(P)$ is green, and $\partial\text{hull}_3(P)$ is blue. Right: The grid points $\pi(p_0), \ldots, \pi(p_9)$ corresponding to $p_0, \ldots, p_9$.

For every point $q \in Q$, let $t(q) > 0$ be the maximum radius such that $q \in \partial\text{hull}_{t(q)}(P)$. Consider the set of radii $T = \{t(q) : q \in Q\}$. Sort the radii in T in decreasing order as $t_0 > t_1 > \ldots > t_{|T|-1}$. We can now define the function $\pi : P \to [0, n)^2 \cap \mathbb{Z}^2$. For every $p_i \in Q$, let $\pi(p_i) = (i, j)$ if and only if $t(p_i) = t_j$, that is, the first coordinate of $\pi(p_i)$ corresponds to the index i of p_i (the x-order of all points in Q), and the second coordinate of $\pi(p_i)$ corresponds to index j of the radius $t_j = t(p_i)$. For every $p_i \in P \setminus Q$, let $\pi(p_i) = (i, |T|)$; see Figure 5 for an illustration. Finally, let $P' = \pi(P) = \{\pi(p_i) : p_i \in P\}$ and $Q' = \pi(Q) = \{\pi(p_i) : p_i \in Q\}$. Note that the points in $P' \setminus Q'$ lie above all points in Q'. Since π is injective, then it is a bijection between P and P'. Note also that $|T| \leq |Q| \leq |P| = n$, consequently $P' \subset [0, n]^2 \cap \mathbb{Z}^2$.

▶ **Lemma 8.** *For a set P of n points in the plane above the x-axis and for every disk D centered below the x-axis, the set $\pi(D \cap P)$ has the lowest-point property.*

Proof. Let D be a disk centered below the x-axis. We rephrase the lowest-point property in terms of $D \cap P$. Recall that the points in P are sorted by x-coordinates. Suppose that $s = (s_x, s_y)$ is in $D \cap P$ and $s_x \in I \subset \text{span}(D \cap P)$. Consider the point sets $P(I) := \{p = (p_x, p_y) \in D \cap P : p_x \in I\}$. By Lemma 7, we know that $D \cap Q \neq \emptyset$; let t be the largest radius in T such that $Q(I) \cap \partial\text{hull}_t(P) \neq \emptyset$. We need to show that D contains all points in $P(I) \cap \partial\text{hull}_t(P)$.

Let q_{left} and $q_{\text{right}} \in P(I)$, resp., be the leftmost and rightmost points in $P(I) \cap Q$; and let L_{left} and L_{right} be the vertical lines through q_{left} and q_{right}. By the definition of Q, we have $q_{\text{left}} \in \partial\text{hull}_{t(q_{\text{left}})}(P)$ and $q_{\text{right}} \in \partial\text{hull}_{t(q_{\text{right}})}(P)$, and $t \geq \max\{t(q_{\text{left}}), t(q_{\text{right}})\}$ by the definition of t. Consequently, the intersection point $\ell := L_{\text{left}} \cap \partial\text{hull}_t(P)$ lies at or below q_{left}, the intersection point $r := L_{\text{right}} \cap \partial\text{hull}_t(P)$ lies at or below q_{right}. Since $q_{\text{left}}, q_{\text{right}} \in D$, then D contains both ℓ and r. We know that $\partial\text{hull}_t(P)$ is an x-monotone curve by Lemma 4(2), and $D \cap \partial\text{hull}_t(P)$ is connected by Lemma 6(2). Since D contains both ℓ and r, then D contains the sub-curve of $\partial\text{hull}_t(P)$ between r and ℓ. Since all points in $P(I)$ are between the vertical lines L_{left} and L_{right}, then D contains all points in $P(I) \cap \partial\text{hull}_t(P)$, as required. ◀

Online algorithm for disks in the separated setting. We can now complete the reduction.

▶ **Theorem 9.** *For the Online Hitting Set problem for a set $P \subset \mathbb{R}^2$ of n points above the x-axis and disks centered below the x-axis, there is an $O(\log n)$-competitive algorithm.*

Proof. We are given a set $P \subset \mathbb{R}^2$ of n points above the x-axis, and we receive a sequence $\mathcal{C} = (D_1, \ldots, D_m)$ of disks centered on or below the x-axis in an online fashion. Let $\mathsf{OPT} \subseteq P$ be a minimum hitting set for $\mathcal{C}$.

Initially, we compute the set $P' \subset [0,n]^2 \cap \mathbb{Z}^2$ as defined above Lemma 8. When a disk D_i arrives, we compute the set $S_i = \pi(D_i \cap P)$, which has the lowest-point property by Lemma 8. The bijection π maps OPT to a set $\mathsf{OPT}' = \pi(\mathsf{OPT}) \subseteq P'$, where $|\mathsf{OPT}| = |\mathsf{OPT}'|$. Here, OPT' is a hitting set for the sets $\mathcal{C}' = (S_1, \ldots, S_m)$.

We run the online algorithm ALG_0 described in Section 2.2 for the point set P' and the sequence $\mathcal{C}'$ of sets. By Theorem 1, ALG returns a hitting set $H' \subseteq P'$ of size $|\mathsf{OPT}'| \cdot O(\log n)$. By Lemma 7, $H = \pi^{-1}(H') \subset P$ is a hitting set for $\mathcal{C}$, and its size is bounded by $|H| = |H'| \leq |\mathsf{OPT}'| \cdot O(\log n) = |\mathsf{OPT}| \cdot O(\log n)$, as required. ◀

4 Disks of Bounded Radii: General Setting

In this section, we consider the *Online Hitting Set* problem, where P is a finite set (given in advance) in the plane; and the objects are disks with radii in the interval $[1, M)$, where $M > 1$ is a constant.

Distinguishing layers of disks, according to their radii. We partition the disks of radii in the interval $[1, M)$ into $\lfloor \log M \rfloor + 1$ layers as follows: for each $j \in \{0, 1, \ldots, \lfloor \log M \rfloor\}$, let **layer** L_j be the set of disks of radii in the interval $[2^j, 2^{j+1})$. The index of each layer L_j is denoted by j.

Tiling of the plane for each layer index j. For every $j \in \{0, 1, \ldots, \lfloor \log M \rfloor\}$, let $\Lambda_j = \{\alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 : (\alpha_1, \alpha_2) \in \mathbb{Z}^2\}$ be a two-dimensional lattice spanned by vectors $\mathbf{v}_1 = 2^{j-1/2}\mathbf{e}_1$ and $\mathbf{v}_2 = 2^{j-1/2}\mathbf{e}_2$, where $\mathbf{e}_1 = (1,0)$ and $\mathbf{e}_2 = (0,1)$ are the standard basis vectors. Let $\tau_j = \left[0, 2^{j-1/2}\right]^2$ be a square of side length $2^{j-1/2}$ with lower-left corner at the origin. Translates of τ_j (**tiles**), with translation vectors in the lattice Λ_j, form the **tiling** $\mathcal{T}_j$. Let $\mathcal{L}_j$ denote the set of axis-parallel lines spanned by the sides of the tiles in $\mathcal{T}_j$.

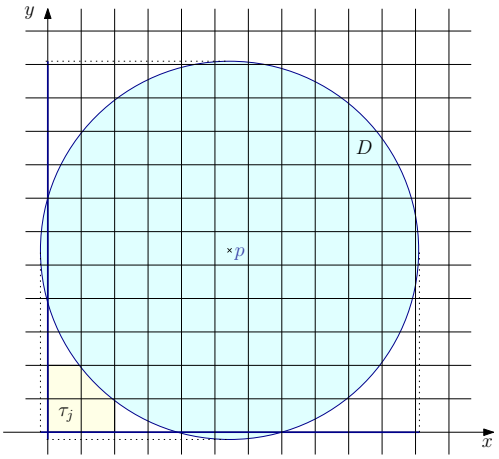


Figure 6 A section of the tiling $\mathcal{T}_j$, the tile τ_j of side length $2^{j-1/2}$, and a disk D of radius 2^{j+2}.

We observe two key properties of the construction of layers and the tilings.

▶ **Observation 10.** *For every $j \in \{0, 1, \ldots, \lfloor \log M \rfloor\}$, if $\sigma \in L_j$ and the center of σ is in a tile $\tau \in \mathcal{T}_j$, then $\tau \subset \sigma$; see Figure 6.*

Proof. Since $\sigma \in L_j$, the radius of the disk σ is in at least 2^j. The tile τ is a translate of $\tau_j = [0, 2^{j-1/2}]^2$, and so its diameter is $\sqrt{2} \cdot 2^{j-1/2} = 2^j$. If the center c of σ is in S, then every $p \in \tau$ is within distance 2^j from c, which implies that $\tau \subset \sigma$. ◀

▶ **Observation 11.** *For every $j \in \{0, 1, \ldots, \lfloor \log M \rfloor\}$, every disk D of radius at most 2^{j+2} intersects at most 24 lines in $\mathcal{L}_j$: at most 12 horizontal and 12 vertical lines.*

Proof. Let D be a disk of radius 2^{j+2}; see Figure 6. The orthogonal projection of D to the x-axis (resp., y-axis) is an interval of length at most 2^{j+3}. Since the distance between any two consecutive vertical (resp., horizontal) lines in $\mathcal{L}_j$ is $2^{j-1/2}$, then D intersects at most $\lceil 2^{j+3}/2^{j-1/2} \rceil = \lceil 2^{7/2} \rceil = 12$ horizontal and at most 12 vertical lines in $\mathcal{L}_j$. ◀

Subproblem for a directed line L. For a directed line L, we denote by L^- and L^+ the closed half-plane on the left and right of L, respectively. Given a directed line L and the input $(P, \mathcal{C})$ of the *Online Hitting Set* problem, where P is a set of points, and $\mathcal{C}$ is a sequence of disks in the plane, we define a subproblem $(P_L, \mathcal{C}_L)$ as follows: Let $P_L = P \cap L^-$, and let $\mathcal{C}_L$ be the subsequence of disks $\sigma_i \in \mathcal{C}$ such that the center of σ_i is in L^+ and σ_i contains at least one point in P_L. Now for each subproblem $(P_L, \mathcal{C}_L)$, we can run the online algorithm ALG_0 described in Theorem 9, which was developed for the separated setting in Section 3. Let $\mathsf{ALG}_0(L)$ denote the online algorithm, where we run the online algorithm ALG_0 on the subproblem $(P_L, \mathcal{C}_L)$.

Online algorithm. We can now present our online algorithm ALG. In the current algorithm, we use the online algorithm $\mathsf{ALG}_0(L)$ as a subroutine. For each $j \in \mathbb{N} \cup \{0\}$, let layer L_j be the set of disks of radii in the interval $[2^j, 2^{j+1})$. The algorithm maintains a hitting set $H \subseteq P$ for the disks presented so far. Upon the arrival of a new disk σ with radius r, if it is already hit by a point in H, then do nothing. Otherwise, proceed as follows.

- First, find the layer L_j, where $j = \lfloor \log r \rfloor$, in which σ belongs.
- Find the tile $\tau \in \mathcal{T}_j$ that contains the center of σ.
  - If $P \cap \tau \neq \emptyset$, then choose an arbitrary point $p \in P \cap \tau$ and add it to H.
  - Otherwise, for every line $L \in \mathcal{L}_j$ that intersects σ, direct L such that L^+ contains the center of σ, feed the disk σ to the online algorithm $\mathsf{ALG}_0(L)$, and add any new hitting point chosen by $\mathsf{ALG}_0(L)$ to H.

Competitive analysis. We now prove that ALG is $O(\log M \log n)$-competitive.

▶ **Theorem 12.** *For the Online Hitting Set problem for a set P of n points in the plane and a sequence $\mathcal{C} = (\sigma_1, \ldots, \sigma_m)$ of disks of radii in the interval $[1, M]$, the online algorithm ALG has competitive ratio $O(\log M \log n)$.*

Proof. Let $\mathcal{C}$ be a sequence of disks. For each $j \in \{0, 1, \ldots, \lfloor \log M \rfloor\}$, let $\mathcal{C}^j$ be the collection of disks in $\mathcal{C}$ with radii in the interval $\left[2^j, 2^{j+1}\right)$. Let H and OPT, resp., be the hitting set returned by the online algorithm ALG and an (offline) minimum hitting set for $\mathcal{C}$. For every point $p \in \mathsf{OPT}$, let $\mathcal{C}_p$ be the set of disks in $\mathcal{C}$ containing p. For each $j \in \{0, 1, \ldots, \lfloor \log M \rfloor\}$, let $\mathcal{C}_p^j$ be the set of disks in $\mathcal{C}^j$ containing p, i.e., $\mathcal{C}_p^j = \mathcal{C}^j \cap \mathcal{C}_p$. Let $H_p^j \subseteq H$ be the set of points that ALG adds to H in response to hit objects in $\mathcal{C}_p^j$. It is enough to show that for every $j \in \{0, 1, \ldots, \lfloor \log M \rfloor\}$ and $p \in \mathsf{OPT}$, we have $|H_p^j| \leq O(\log n)$.

Let τ be the tile in $\mathcal{T}_j$ that contains p, and let ${\mathcal{C}'}_p^j \subseteq \mathcal{C}_p^j$ be the subset of disks whose centers are located in τ. To hit the first disk $\sigma \in {\mathcal{C}'}_p^j$, our algorithm adds a point from $P \cap \tau$ to H. By Observation 10, any point in $P \cap \tau$ hits σ, as well as any subsequent disks in ${\mathcal{C}'}_p^j$. Our algorithm adds at most 1 point to H to hit all the disks in ${\mathcal{C}'}_p^j$.

It remains to bound the number of points our algorithm adds for disks in $\mathcal{C}_p^j \setminus \mathcal{C}'^j_p$. Notice that a disk D_0 centered at p of radius 2^{j+1} contains all the centers of the disks in $\mathcal{C}_p^j \setminus \mathcal{C}'^j_p$. By the triangle inequality, a disk D centered at p of radius 2^{j+2} contains all disks in $\mathcal{C}_p^j \setminus \mathcal{C}'^j_p$. For any disk $\sigma \in \mathcal{C}_p^j \setminus \mathcal{C}'^j_p$, our algorithm uses algorithm $\mathsf{ALG}_0(L)$ for a line $L \in \mathcal{L}_j$, directed such that L^+ contains the center of σ. According to Observation 11, the disk D intersects at most 24 lines in $\mathcal{L}_j$. However, depending on the location of the center of σ, each line may be used in either direction for $\mathsf{ALG}_0(L)$. As a result, for all disks in $\mathcal{C}_p^j \setminus \mathcal{C}'^j_p$, algorithm $\mathsf{ALG}_0(L)$ is called with at most 48 directed lines L.

For each directed line L, the online algorithm $\mathsf{ALG}_0(L)$ maintains a hitting set $H(L)$ for the disks fed into this algorithm. For the point p, let $H_p^j(L)$ denote the set of points that algorithm $\mathsf{ALG}_0(L)$ adds to $H(L)$ in response to a disk in $\mathcal{C}_p^j \setminus \mathcal{C}'^j_p$ that it receives as input. By Theorem 9, we have $|H_p^j(L)| \leq O(\log |\mathcal{C}_p^j \setminus \mathcal{C}'^j_p|) \leq O(\log n)$ for every directed line L. This yields $|H_p^j| \leq 1 + 48 \cdot O(\log n) = O(\log n)$, as required.

By construction, we have $H = \bigcup_{j=0}^{\lfloor \log M \rfloor} \bigcup_{p \in \mathsf{OPT}} H_p^j$. We have shown that $|H_p^j| = O(\log n)$, for all $j \in \{0, 1, \ldots, \lfloor \log M \rfloor\}$ and $p \in \mathsf{OPT}$. Consequently, we obtain $|H| \leq \sum_{j=0}^{\lfloor \log M \rfloor} \sum_{p \in \mathsf{OPT}} O(\log n) = (\lfloor \log M \rfloor + 1)\, |\mathsf{OPT}|\, O(\log n) = O(\log M \log n)|\mathsf{OPT}|$. ◀

For disks of radii in $[1, 1+\varepsilon]$ where $\varepsilon > 0$ is constant, Theorem 12 implies the following.

▶ **Corollary 13.** *For the Online Hitting Set problem for a set P of n points in the plane and a sequence $\mathcal{C} = (\sigma_1, \ldots, \sigma_m)$ of disks of radii in the interval $[1, 1+\varepsilon]$, where $\varepsilon > 0$ is a constant, the online algorithm ALG is $O(\log n)$-competitive.*

5 Generalization to Positive Homothets of a Convex Body

In this section, we generalize Theorem 12 for positive homothets of an arbitrary convex body C in the plane. A set $C \subset \mathbb{R}^2$ is a **convex body** if it is convex and has a nonempty interior; and it is **centrally symmetric** (w.r.t. the origin) if $C = -C$, where $-C = \{-p : p \in C\}$.

The key components of our $O(\log n)$-competitive algorithm for disks of comparable sizes were an $O(\log n)$-competitive online algorithm in the line-separated setting and a grid tiling that allowed a reduction to the line-separated setting. Specifically, Observation 10 and Observation 11 formulate the two essential properties of a tiling: If a center of disk σ lies in a tile τ, then $\tau \subset \sigma$ (Observation 10); and every disk intersects $O(1)$ grid lines (Observation 11).

We state the main result of this section here. For a detailed explanation and the complete proof, please refer to the full version of the paper.

▶ **Theorem 14 (⋆).** *Given any convex body $\sigma \subset \mathbb{R}^2$ and a parameter $M \geq 1$, there is an online algorithm with a competitive ratio of $O(\log M \log n)$ for the Online Hitting Set problem for a set P of n points in the plane and a sequence $\mathcal{C} = (\sigma_1, \ldots, \sigma_m)$ of positive homothets $\sigma_i = a_i\sigma + b_i$, where $a_i \in [1, M]$.*

For positive homothets of a convex object with scaling factor in $[1, 1+\varepsilon]$, where $\varepsilon > 0$ is a constant, Theorem 14, implies the following.

▶ **Corollary 15.** *Given any convex body $\sigma \subset \mathbb{R}^2$ and constant $\varepsilon > 0$, there is an online algorithm of competitive ratio $O(\log n)$ for the Online Hitting Set problem for a set P of n points in the plane and a sequence $\mathcal{C} = (\sigma_1, \ldots, \sigma_m)$ of positive homothets $\sigma_i = a_i\sigma + b_i$, where $a_i \in [1, 1+\varepsilon]$.*

A good pair of lines. The key technical tool for the proof of Theorem 14 is Definition 16. Given a convex body C, we first consider an inscribed triangle of the maximum area (see [13]). We then apply an area-preserving (unary) affine transformation to transform C so that this inscribed triangle of the maximum area becomes an equilateral triangle. (This is similar to mapping the minimum enclosing ellipse of C into a circle, or assuming that C is "fat" after a suitable affine transformation.) We may further assume, by scaling, that the inscribed circle of this triangle has a unit diameter.

▶ **Definition 16.** *Let C be a convex body in the plane such that an inscribed triangle of the maximum area is an equilateral triangle T_{in}, and the circle inscribed in T_{in} is a circle of a unit diameter. A pair of lines $\{\ell_1, \ell_2\}$ is a* ***good pair for C*** *if they satisfy the following properties:*

1. *The angle between the two lines is bounded from below by $\angle(\ell_1, \ell_2) \geq \pi/15$.*
2. *For $i \in \{1, 2\}$, there exist points $p_i, q_i \in \partial C$ such that the two lines tangent to C parallel to ℓ_i contain p_i and q_i, respectively; furthermore, C contains the disk $B(x, \frac{1}{50})$ of diameter $\frac{1}{25}$ centered at the intersection point $x = p_1q_1 \cap p_2q_2$.*

In the full version, we prove that every convex body C specified in Definition 16 admits a good pair of lines, which can be computed in polynomial time if C is a convex polygon. No attempts were made to optimize the constants $\pi/15$ and $\frac{1}{50}$ in Definition 16.

6 Conclusions and Open Problems

We revisited the *Online Hitting Set* problem for a set of n points in the plane and geometric objects that arrive in an online fashion, such as disks or homothets of a convex body of comparable sizes, or bottomless rectangles in the plane. In all these cases, we designed $O(\log n)$-competitive online algorithms, which is the best possible. It remains an open problem whether our results generalize to 3- or higher dimensions. In fact, no $O(\log n)$-competitive algorithm is currently known for simple geometric objects in 3-space, for example, a set of n points and a sequence of unit balls in $\mathbb{R}^3$; or a set of n points $P \subset [0, n)^3 \cap \mathbb{Z}^3$ and a sequence of axis-aligned cubes in $\mathbb{R}^3$.

Our results provide further evidence that there may exist $O(\log n)$-competitive algorithms for the *Online Hitting Set* problem for n points in $\mathbb{R}^d$ and any sequence of objects $\mathcal{C}$ of bounded VC-dimension – an open problem raised by Even and Smorodinsky [12]; see also [14]. This problem remains open: The current best lower and upper bounds are $\Omega(\log n)$ and $O(\log^2 n)$ [2]. No better bounds are known even in some of the most common geometric range spaces, for example, when P is a subset of the grid $[0, n)^2 \cap \mathbb{Z}^2$ and $\mathcal{C}$ is a sequence of axis-aligned rectangles in the plane; or when P is a set of n points in the plane, and $\mathcal{C}$ is a sequence of disks of arbitrary radii.

References

1 Shanli Alefkhani, Nima Khodaveisi, and Mathieu Mari. Online hitting set of d-dimensional fat objects. In *Proc. 21st International Workshop on Approximation and Online Algorithms (WAOA)*, volume 14297 of *LNCS*, pages 134–144. Springer, 2023. doi:10.1007/978-3-031-49815-2_10.

2 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009. doi:10.1137/060661946.

3 Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. doi:10.1137/S0097539702418498.

4 Jonathan B. Conroy and Csaba D. Tóth. Hop-spanners for geometric intersection graphs. *Journal of Computational Geometry*, 14(2):26–64, 2023. doi:10.20382/jocg.v14i2a3.

5 Minati De, Saksham Jain, Sarat Varma Kallepalli, and Satyam Singh. Online geometric covering and piercing. *Algorithmica*, 86:2739–2765, 2024. doi:10.1007/s00453-024-01244-1.

6 Minati De, Ratnadip Mandal, and Satyam Singh. New lower bound and algorithm for online geometric hitting set problem. In *Computing and Combinatorics - 31st International Computing and Combinatorics Conference, COCOON 2025, Proceedings, Part I*, volume 15983 of *Lecture Notes in Computer Science*, pages 234–247. Springer, 2025. doi:10.1007/978-981-95-0215-8_18.

7 Minati De and Satyam Singh. Hitting geometric objects online via points in $\mathbb{Z}^d$. In *Proc. 28th Conf. Computing and Combinatorics (COCOON)*, volume 13595 of *LNCS*, pages 537–548. Springer, 2022. doi:10.1007/978-3-031-22105-7_48.

8 Minati De and Satyam Singh. Online hitting of unit balls and hypercubes in $\mathbb{R}^d$ using points from $\mathbb{Z}^d$. *Theor. Comput. Sci.*, 992:114452, 2024. doi:10.1016/J.TCS.2024.114452.

9 Adrian Dumitrescu, Anirban Ghosh, and Csaba D. Tóth. Online unit covering in Euclidean space. *Theor. Comput. Sci.*, 809:218–230, 2020. doi:10.1016/J.TCS.2019.12.010.

10 Adrian Dumitrescu, Anirban Ghosh, and Csaba D. Tóth. Sparse hop spanners for unit disk graphs. *Comput. Geom.*, 100:101808, 2022. doi:10.1016/J.COMGEO.2021.101808.

11 Adrian Dumitrescu and Csaba D. Tóth. Online unit clustering and unit covering in higher dimensions. *Algorithmica*, 84(5):1213–1231, 2022. doi:10.1007/S00453-021-00916-6.

12 Guy Even and Shakhar Smorodinsky. Hitting sets online and unique-max coloring. *Discret. Appl. Math.*, 178:71–82, 2014. doi:10.1016/J.DAM.2014.06.019.

13 Ivor van der Hoog, Vahideh Keikha, Maarten Löffler, Ali Mohades, and Jérôme Urhausen. Maximum-area triangle in a convex polygon, revisited. *Inf. Process. Lett.*, 161:105943, 2020. doi:10.1016/J.IPL.2020.105943.

14 Arindam Khan, Aditya Lonkar, Saladi Rahul, Aditya Subramanian, and Andreas Wiese. Online and dynamic algorithms for geometric set cover and hitting set. In *Proc. 39th Symposium on Computational Geometry (SoCG)*, volume 258 of *LIPIcs*, pages 46:1–46:17. Schloss Dagstuhl, 2023. doi:10.4230/LIPICS.SOCG.2023.46.

Quantum Approximate k-Minimum Finding

Minbo Gao
Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China

Zhengfeng Ji
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Qisheng Wang
School of Informatics, University of Edinburgh, UK

Abstract

Quantum k-minimum finding is a fundamental subroutine with numerous applications in combinatorial problems and machine learning. Previous approaches typically assume oracle access to exact function values, making it challenging to integrate this subroutine with other quantum algorithms. In this paper, we propose an (almost) optimal quantum k-minimum finding algorithm that works with approximate values for all $k \geq 1$, extending a result of van Apeldoorn, Gilyén, Gribling, and de Wolf (FOCS 2017) for $k = 1$. As practical applications, we present efficient quantum algorithms for identifying the k smallest expectation values among multiple observables and for determining the k lowest ground state energies of a Hamiltonian with a known eigenbasis.

2012 ACM Subject Classification Theory of computation → Quantum query complexity; Theory of computation → Design and analysis of algorithms

Keywords and phrases Quantum Computing, Quantum Algorithms, Quantum Minimum Finding

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.51

Related Version *Full Version*: `https://arxiv.org/abs/2412.16586` [16]

Funding MG and ZJ were supported by National Key Research and Development Program of China (Grant No. 2023YFA1009403).
Zhengfeng Ji: ZJ was supported by National Natural Science Foundation of China (Grant No. 12347104 and No. 61832015), Beijing Natural Science Foundation (Grant No. Z220002), and a startup funding from Tsinghua University.
Qisheng Wang: QW was supported by the Engineering and Physical Sciences Research Council under Grant EP/X026167/1.

Acknowledgements The authors thank Mingsheng Ying and François Le Gall for helpful suggestions and discussions. Minbo Gao would like to thank François Le Gall for supporting his visit to Nagoya University where part of the work was completed.

1 Introduction

Quantum minimum finding was initially proposed in [14], with a direct but clever use of Grover search [19]. They showed that given quantum query access to n unordered elements (with exact values), one can find the smallest one with an optimal quantum query complexity $\Theta(\sqrt{n})$, which quadratically outperforms any classical algorithms. As a generalization, quantum k-minimum finding was later developed in [13], which finds the k smallest elements from n elements with an optimal quantum query complexity $\Theta(\sqrt{nk})$. Quantum k-minimum finding for exact values turns out to have various applications in different fields, e.g., minimum spanning tree and strong connectivity in graph theory [13], clustering [3, 27] and k-nearest neighbor [43] in machine learning (see also [37, 7]), and pattern matching [34] and other string problems [42, 4, 24] in stringology.


© Minbo Gao, Zhengfeng Ji, and Qisheng Wang;
licensed under Creative Commons License CC-BY 4.0



The traditional quantum (k-)minimum finding works when the elements are comparable by their exact values. However, it does not apply to the numerical case, where the values of the elements can be computed by (stochastic) numerical methods. Notably, this case often appears in practice, especially when the values of the elements are estimated by quantum subroutines such as quantum phase estimation [25, 31] and amplitude estimation [9]. A typical example is the quantum SDP (semi-definite program) solver proposed in [40, 8, 39], where the ground energy of a Hamiltonian is found by a quantum minimum finding over approximate quantum phases. To facilitate the discussion of the numerical scenarios and the composability of quantum k-minimum finding, we introduce an abstract type of quantum oracle for numerical inputs, called *approximate oracle*. In an approximate oracle, each query returns a *superposition* of approximate values of the queried element to high precision and with high success probability (see Definition 5 for the formal definition). This is a very general definition which meets the requirement for all known situations where quantum k-minimum finding is used as a subroutine. A key question is then

How to solve quantum k-minimum finding problem with an approximate oracle?

This question is not only of theoretical interest, but also holds significant practical value. The solution turns out to be more subtle as one may initially expect. To address this problem clearly, we propose two (weak and strong) definitions of approximate minimum index sets as characterizations of the required output in k-minimum finding algorithms, naturally generalizing the requirement of the approximate minimum finding problem. We give almost optimal quantum algorithms for computing these sets with $\widetilde{O}(\sqrt{nk})$ queries to the approximate oracle. Finally, we showcase two applications of our algorithms: identifying the k smallest expectation values and determining the k lowest ground state energies of a Hamiltonian.

1.1 Approximate Oracles and Approximate Minimum Index Sets

Before presenting our main results, we define the concept of approximate oracle, explain why the standard algorithm for quantum k-minimum finding fail in this case, and define the concept of approximate minimum index set as an appropriate output.

1.1.1 Approximate Oracles

In the standard setting of quantum (k-)minimum finding (see [14, 13]), an *exact oracle* for the input data is assumed, which always returns the *exact function value* for each query. The assumption of an exact oracle is reasonable in the case where the input data is previously known or is produced by deterministic algorithms. However, it fails to describe the situation where the input data is generated by randomized or quantum numerical algorithms.

Generally speaking, for a randomized numerical algorithm that estimates a real number x, its output is a random variable X, such that the distance between X and x is no more than ε with probability at least $1-\delta$. We will call ε the precision parameter and δ the error parameter from now on. If the values are estimated using quantum algorithms, a general situation is that the estimated values are stored in a quantum state in superposition. Hence, it is natural to consider the following definition of *approximate oracles*. An (ε, δ)-approximate oracle for the input x_i's is a quantum unitary operator that with probability at least $1-\delta$, outputs a quantum state which is a *superposition* of estimated values for x_i within ε distance upon querying i. The formal definition of approximate oracle can be found in Definition 5. This input model works well with two widely-used quantum procedures for quantum numerical computations, namely phase estimation and amplitude estimation.

There are other input unitary oracle models, such as the bounded-error oracle [21, 42], and untrustworthy oracle [32]. We summarize their properties in the following table, with a detailed discussion in Section 3.1.

Table 1 Different Unitary Oracle Models for Minimum Finding.

	Allows Error	Considers Precision	Complexity of Min Finding
Exact oracle [13]	×	×	$O(\sqrt{n})$
Bounded-error oracle [42]	✓	×	$\widetilde{O}(\sqrt{n})$
Untrustworthy oracle [32]	✓	✓	$\widetilde{O}(\sqrt{n(1+\Delta)})^a$
Approximate oracle	✓	✓	$\widetilde{O}(\sqrt{n})$

[a] Δ is the upper bound on the number of elements within any interval of unit length.

1.1.2 Approximate Minimum Index Set

When discussing the k-minimum finding problem, it is important to define what conditions of the output, which we call the approximate minimum index set, need to satisfy. For reason to be clear later, we will consider both the weak and strong approximate minimum index sets.

As a warm-up, let us consider what we can learn from the randomized numerical algorithms. Suppose that there are n elements $x_1, x_2, \ldots, x_n$, and for each $j \in [n]$, a random result X_j is generated by a randomized numerical algorithm, that is ε-close to the actual value x_j with probability at least $1 - \delta$. With this as the input oracle, it may be impossible for us to find out the k smallest elements in extreme cases, for example, when all elements are ε-close to each other. As a compromise, we seek an approximate solution instead.

For the minimum finding problem, the authors of [40] implicitly gave a characterization of the *approximate minimum index*, namely an index j whose value x_j is ε-close to the true minimum value. This characterization, however, could be interpreted and generalized in various different ways when k-minimum finding is considered.

Weak Approximate Minimum Index Set. One way to understand the approximate minimum index is that the solution we find are *entry-wise* ε-close to the ideal solution. Generalizing this idea to the k-minimum case, we propose the definition of the *weak approximate minimum index set*, which requires the approximate solution to be entry-wise ε-close to the k minima (see Definition 11 for a formal definition).

Strong Approximate Minimum Index Set. The other way to interpret the approximate minimum is to regard it as an ε-approximate version of a *global* condition. To be more precise, the minimum has the property that it is no more than all other elements; and the approximate minimum index requires this condition to hold within at most ε-error. Generalizing this idea to the k-minimum case, we propose the definition of the *strong approximate minimum index set*, which requires the approximate solution to be bounded by the other values whose indices are not in the set within at most ε-error (see Definition 13 for a formal definition).

Relationships between the two definitions. To the best of our knowledge, neither of the above definitions has been proposed for $k \geq 2$, and their relationships are not well understood before. Although the above definitions are equivalent for $k = 1$, this equivalence does not

hold for larger k anymore. In the paper, we show that a strong approximate minimum index set must be a weak one, but not vice versa (see Section 4 for more details). We also note that, in certain scenarios, a strong (k, ε)-approximate minimum index set is needed and a weak one does not suffice (see Remark 2 for more details).

1.2 Main Results

Given an approximate oracle as an input, we design *almost optimal* quantum algorithms for k-minimum finding, based on the concepts of weak and strong approximate minimum index sets.

▶ **Theorem 1** (Quantum Approximate k-minimum Finding, informal version of Theorem 17 and Theorem 18). *Suppose V is an (ε, δ)-approximate oracle for a vector $v \in \mathbb{R}^n$, where $\delta > 0$ is sufficiently small, and suppose k is the desired number of minimal indices.*

- *Weak Approximate Minimum Index Set Finding: There is a quantum algorithm that, with high probability, outputs a weak $(k, 2\varepsilon)$-approximate minimum index set S for v, using $\widetilde{O}(\sqrt{nk})$ queries to V.*
- *Strong Approximate Minimum Index Set Finding: There is a quantum algorithm that, with high probability, outputs a strong $(k, 7\varepsilon)$-approximate minimum index set S for v, using $\widetilde{O}(\sqrt{nk})$ queries to V.*

Actually, a weak approximate minimum index set is not necessarily a strong one (see Example 15). This is because the weak approximate minimum index set focuses on entry-wise approximation while the strong one considers the global property that small values should not be ignored. The need of strong approximate minimum index sets exists in quantum machine learning. For example, in [15], a strong approximate minimum index set is required; in case the approximate minimum index set is weak, their key intermediate conclusion [15, Proposition D.12] no longer holds. Due to the well-known $\Omega(\sqrt{nk})$ bound of quantum k-minimum finding [13], and the exact oracle being a special case of the approximate oracle, our algorithms for approximate k-minimum finding achieves almost optimal query complexity for this problem.

The assumption that δ needs to be sufficiently small is not critical in applications. In fact, by applying the median trick to the approximate oracle, similar to techniques used in phase estimation and amplitude estimation [30], the failure probability could be reduced with an extra logarithmic overhead if it is smaller than $1/3$.

1.3 Techniques

The algorithm for finding a weak minimum index set serves as the basis of the algorithm for finding a strong index set, although the techniques used in designing these algorithms are quite different. We now present the techniques for both algorithms separately.

To design a quantum algorithm for finding the weak approximate minimum index set, we first observe an important property of the weak approximate minimum index set which we call *error-tolerant incrementability* (Proposition 12). This property allows us to add an "approximately" $(k+1)$ smallest entry to a weak (k, ε)-approximate minimum index set and form a weak $(k+1, \varepsilon)$-approximate minimum index set. It serves as the key property in demonstrating weak approximate minimum index set finding (see the full version, specially [16, Theorem A.1]), and enables us to solve the weak k-minimum finding problem by repeatedly applying the quantum approximate minimum finding with an approximate oracle [40, 11]. We can therefore mimic the way how the quantum k-minimum finding [13] can be solved using

original quantum minimum finding [14]. To be more concrete, our algorithm (Algorithm 1) works as follows: in the j-th iteration of the k runs, the algorithm finds an element not selected yet with value (approximately) no greater than the $(k-j+1)$-th element. While this approach may seem adaptable to more general cases, it turns out that it can only return a *weak* approximate minimum index set, since strong approximate minimum index sets do not exhibit the error-tolerant incrementability property.

Nevertheless, we still manage to design a quantum algorithm for finding a strong approximate minimum index set with several additional new ideas. The key insight is that the algorithm for finding a weak approximate minimum index set could be used to estimate the value of the k-th smallest element approximately. Thus, we could set a threshold u which is slightly smaller than the value of the k-th smallest element. The threshold u can be understood as a filter, capturing small elements that must not be ignored. Having obtained the threshold u, we focus on all estimates with value lower than u (including those estimates whose corresponding elements are larger than u), and then sample (at most) $\widetilde{O}(k)$ indices. From these, we can find k indices that form a *strong* approximate minimum index set.

More specifically, we begin with a quantum state $|\Phi\rangle = \frac{1}{\sqrt{n}}\sum_{j=1}^{n}|j\rangle|\Lambda_j\rangle$, where $|\Lambda_j\rangle = \sum_\nu \alpha_{j,\nu}|\nu\rangle$ is a superposition of all possible estimates of the j-th element. If we focus on the register containing the estimate values ν, then we can write $|\Phi\rangle = \sum_\nu |S_\nu\rangle|\nu\rangle$, where $|S_\nu\rangle$ is a superposition of indices. If we now condition on whether ν is lower than the threshold u, the quantum state can be written as $|\Phi\rangle = |\Phi_{\le u}\rangle + |\Phi_{>u}\rangle$, where $|\Phi_{\le u}\rangle = \sum_{\nu\le u}|S_\nu\rangle|\nu\rangle$. By quantum amplitude amplification [9], we can sample one index from $|\Phi_{\le u}\rangle$ by using $O(1/\||\Phi_{\le u}\rangle\|)$ queries to the unitary oracle that prepares $|\Phi\rangle$. Using an argument similar to the coupon collector's problem, we prove that sampling sufficiently many (precisely, $\Theta(n\||\Phi_{\le u}\rangle\|^2)$) times will collect all the "small" elements with high probability. Finally, we use simple post-processing to find a strong approximate minimum index set (see Section 6 for more details).

1.4 Applications

Our algorithms for finding approximate minimum index sets can be applied to several practical scenarios, some of which are formally discussed in Section 7.

Approximate k minima in multiple expectations. Given purified access to n quantum states $\{\rho_j\}_{j=1}^n$, and block-encoding access (see Definition 4) to n observables $\{O_j\}_{j=1}^n$, the task is to approximately find the k minimal values among expectations $\mathrm{tr}(O_j\rho_j)$. The expectation $\mathrm{tr}(O\rho)$ of an observable O with respect to a quantum state ρ can be estimated, for example, by the Hadamard test [2] or quantum measurements. The problem has been considered in the literature in various situations where, in most of the cases, the same quantum state are used for different observables. These include quantum SDP solvers [8, 40, 39], shadow tomography [22, 1, 10] and multiple expectation estimation [23]. By our quantum approximate k-minimum finding with amplitude estimation, we can find an ε-approximate of the k minimal expectations in $\widetilde{O}(\sqrt{nk}/\varepsilon)$ time.

Approximate k ground state energies. Given block-encoding access to a Hamiltonian with the known eigenbasis, our algorithm can be used to find an ε-approximate of the k ground state energies in $\widetilde{O}(\sqrt{nk}/\varepsilon)$ time, which could be seen as a generalization of the approximate ground state energy finding [40].

In the special case when the Hamiltonian is diagonal (i.e., with the computational basis as its eigenbasis), our quantum algorithm for approximate k-minimum finding can be used in quantum multi-Gibbs sampling [15]. See Remark 2 for a more detailed discussion.

▶ Remark 2. For the special case where the approximate oracle is implemented by quantum amplitude estimation, a quantum k-minimum finding was claimed in [15]. However, an error in their Theorem C.3 was found due to their incorrect error-reduction of consistent quantum amplitude estimation. Nevertheless, using our quantum *strong* approximate k-minimum finding with an approximate oracle, we can fill the gaps in their proof, and their result still holds.

2 Preliminaries

2.1 Notations

For $n \in \mathbb{Z}_+$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$, and we set $[0] = \emptyset$ for convenience. $\mathbb{R}^n_{\geq 0}$ stands for the set of n-dimensional vectors with non-negative entries. For sets A and B, we use $A - B$ to denote the set consisting of the elements in A but not in B. We assume the readers are familiar with the basic knowledge of quantum computing (see [31] for a reference). In this paper, when discussing the query complexity of U, we assume having query access to U, $U^\dagger$ and their controlled versions.

2.2 Concepts in Quantum Computing

We use a quantum data structure called QSet to store the found indices in our algorithm. The functionality of this data structure is characterized by the following proposition.[1]

▶ **Proposition 3.** *There is a quantum data structure* QSet *that supports the following operations.*

- *Initialization* QSet.Initialize(n)*: return an instance $\mathcal{S}$ of* QSet*, with a set S initialized to empty set which can store at most n elements.*
- *Add* S.Add(j) *for an $j \in [n]$: insert the element j into the set S.*
- *Hide* S.Hide()*: output a unitary satisfying* S.Hide()$|i\rangle|j\rangle = |i\rangle|2 \times \mathbf{1}_S(i) + j\rangle$ *for any computational basis $|j\rangle$, where $\mathbf{1}_S(\cdot)$ is the indicator function of the set S, i.e, $\mathbf{1}_S(i) = 1$ if $i \in S$, and $\mathbf{1}_S(i) = 0$ if $i \in [n] - S$.*

Block-encoding is a very useful concept in quantum algorithms proposed in [17], and we recall its definition as follows.

▶ **Definition 4** (Block-encoding [17, Definition 24])**.** *Let A be a linear operator acting on an s-qubit Hilbert space. For $\alpha, \varepsilon > 0$ and integer a, we say an $(a+s)$-qubit unitary operator U is an (α, a, ε)-block-encoding of A, if $\|A - \alpha\langle 0|^{\otimes a} U |0\rangle^{\otimes a}\| \leq \varepsilon$.*

When a and ε could be omitted (for instance, $a = O(1)$ and $\varepsilon = 0$, or the values of a and ε have shown up in previous context), we simply call U an α-block-encoding of A.

3 Approximate Oracle

▶ **Definition 5** (Approximate Oracle)**.** *Suppose $v = (v_1, \ldots, v_n) \in \mathbb{R}^n_{\geq 0}$ is an n-dimensional vector with $v_j \in [0, 1]$ for $j \in [n]$. For $\varepsilon \in [0, 1/2)$ and $\delta \in [0, 1/2)$, a unitary V is called an (ε, δ)-approximate oracle for v, if it satisfies $V|i\rangle|j\rangle = |i\rangle|\Lambda_i + j\rangle$ for all $i \in [n]$, where*

[1] If we allow the use of quantum(-read classical-write) random access memory (QRAM) (see [6] for a more detailed discussion), all these operations could be implemented efficiently (i.e., in $\widetilde{O}(1)$ one- and two-qubit gates and QRAM read and write operations).

$$|\Lambda_i + j\rangle = \sqrt{p_i^\varepsilon}|\Lambda_i^\varepsilon + j\rangle + \sqrt{1-p_i^\varepsilon}|\Lambda_i^{\varepsilon\perp} + j\rangle,$$

is a normalized state in a d-dimensional space, $|\Lambda_i^\varepsilon + j\rangle = \sum_{\nu:\,|\nu - v_i|\le\varepsilon} \alpha_\nu^{(i)}|\nu + j\rangle$, $|\Lambda_i^{\varepsilon\perp} + j\rangle$ *is the state orthogonal to* $|\Lambda_i^\varepsilon + j\rangle$*, and* $p_i^\varepsilon \ge 1 - \delta$ *for all* $i \in [n]$.[2]

▶ Remark 6. In simple words, the definition states that V is an (ε, δ)-approximate oracle for v, if for each $i \in [n]$, by querying i, with probability at least $1 - \delta$, V will return (a superposition) of estimated values in $[v_i - \varepsilon, v_i + \varepsilon]$. As an example, such a scenario shows up when the phase estimation procedure is used to estimate the values of v_i.

By using the median trick [30], we have the following proposition stating that the success probability of the approximate oracle can be efficiently amplified.

▶ **Proposition 7** (Failure probability reduction of the approximate oracle)**.** *Suppose* $v \in \mathbb{R}^n$ *is an* n*-dimensional vector and* V *is an* (ε, δ)*-approximate oracle for* v*, for some* $\varepsilon \in (0, 1/2)$ *and* $\delta \in (0, 1/3)$*. Then, for any* $\delta' \in (0, \delta)$*, one can construct an* (ε, δ')*-approximate oracle for* v*, using* $O(\log(\delta/\delta'))$ *queries to* V*.*

3.1 Comparisons of Different Oracles

In this part, we briefly investigate different input oracles considered in previous literature for the quantum minimum finding problem, namely the bounded-error oracle and the untrustworthy oracle.

Bounded-error Oracle. As the name suggests, bounded-error considers the case when only bounded-error access to the input is provided for computation tasks, such as quantum search [21] and quantum minimum finding [42]. This kind of oracle model could return the *correct exact answer* with bounded error (i.e., high probability), while the exact oracle returns the correct exact answer with certainty. Therefore, bounded-error oracle could be seen as a slight generalization of the exact oracle. Their computational powers are close, as the bounded-error oracle can simulate the exact oracle by majority voting with logarithmic repetitions.

Untrustworthy Oracle. The precision issue for the oracle model motivates researchers to consider another oracle model [32], which we call the untrustworthy oracle.[3] The untrustworthy oracle will return the correct comparison result between two elements if they are far away enough; and an unknown, possibly adversary (but deterministic) comparison result, if the distance between them is less than a fixed threshold ε. In this model, quantum minimum finding over n elements could be done in time $\widetilde{O}(\sqrt{n(1+\Delta)})$, where Δ is the upper bound on the number of elements within any interval of length ε. The result shows that, achieving quadratic speedup for the minimum finding in this model may pose strict requirements on the inputs.

[2] We also allow the case where V produces ancillary states, namely $V|i\rangle|j\rangle|0\rangle = |i\rangle|\Lambda_i + j\rangle|\psi_j\rangle$, which will not influence our proofs.

[3] The untrustworthy oracle is originally called "noisy oracle" in [32]. However, we renamed it according to its behavior, to avoid the conflict with the noisy oracle related to physical noises.

We note that our definition of approximate oracle includes exact oracle and bounded-error oracle as special cases. In fact, a bounded-error oracle with bounded-error δ is a $(0, \delta)$-approximate oracle by definition. The approximate oracle model is generally incomparable to the untrustworthy oracle, see Remark 8 for a detailed discussion.

▶ Remark 8. The untrustworthy oracle is incomparable with the approximate oracle. This can be seen from two perspectives. First, the latter cannot simulate the former, because the comparison between close elements by the approximate oracle is always in superposition. Second, the former reveals more information than the latter when comparing two elements x_1, x_2 with distance slightly above the threshold, say $|x_1 - x_2| = 1.1\varepsilon$. In this case, the former returns the correct comparison with certainty, while the latter may not be able to even distinguish the two elements (in extreme cases, the approximate values of the two elements x_1, x_2 can be the same, i.e. $(x_1 + x_2)/2$, with certainty).

There are other types of input oracles that are quantum channels due to imperfect implementations and noise; they were considered in the unstructured search problem in the literature, e.g., [38, 5, 26, 35, 20, 36]. In particular, in [35, 36], it was shown that no quadratic speedup exists in the presence of (constant-rate) depolarizing or dephasing noise after each application of the exact oracle. By contrast, the bounded-error, untrustworthy and approximate oracles are all unitary.

4 Approximate k-Minimum Index Set: Weak and Strong Definitions

The problem of finding the smallest element from a set of n numbers using an oracle that provides an approximate value has been a topic of interest in quantum computation studies, specifically quantum algorithms for optimization tasks [40, 11]. As discussed in Section 3, a natural setup where the values of vector v are prepared involves employing the amplitude estimation procedure with a limited precision parameter ε using V. Given the limitation of the estimation precision, it is reasonable to expect that we could only find an *approximate* minimum value that is $O(\varepsilon)$-close to the true minimum. We also observe that, instead of finding the approximate minimum value, it suffices to find the index of the approximate minimum value. Thus, motivated by prior results (Theorem 50 in [40] and Theorem 2.4 in [11]), we present the following notion of ε-approximate minimum index, which were implicit in those prior works.

▶ **Definition 9** (Approximate Minimum Index)**.** *Let $v = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n_{\geq 0}$ be an n-dimensional vector. Suppose the coordinates of v are sorted from small to large as*

$$v_{s_1} \leq v_{s_2} \leq v_{s_3} \leq \cdots \leq v_{s_n},$$

then, an ε-approximate minimum index for v is an index $i \in [n]$ satisfying $v_i \leq v_{s_1} + \varepsilon$.

▶ Remark 10. For some v and ε, there may exist multiple indices that are ε-approximate minimum indices of v. For instance, if $v = (1, 1, \ldots, 1) \in \mathbb{R}^n$, then, for any $\varepsilon > 0$ and any $j \in [n]$, j is an ε-approximate minimum index of v by definition.

In simple terms, i is an ε-approximate minimum index for v if $v_i \leq \min_{j\in[n]} v_j + \varepsilon$. Therefore, after identifying an ε-approximate minimum index i, one can use amplitude estimation to obtain an ε-approximate estimate of v_i, which is $O(\varepsilon)$-close to the actual minimum $\min_{j\in[n]} v_j$.

Interestingly, generalizing this concept for finding k smallest elements approximately is challenging. In this section, we introduce two definitions of the approximate k-minimum index set, extending the notion of approximate minimum index in two distinct ways. Furthermore, we explore the relationships between these definitions.

We first propose the definition of weak (k, ε)-approximate minimum index set, which requires that after sorting the elements in the set from small to large, they are entry-wise ε-close to the real minima.

▶ **Definition 11** (Weak Approximate Minimum Index Set). *Let $v = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n_{\geq 0}$ be an n-dimensional vector. Suppose the coordinates of v are sorted from small to large as $v_{s_1} \leq v_{s_2} \leq v_{s_3} \leq \cdots \leq v_{s_n}$. A set S of size k is called a* weak (k, ε)-approximate minimum index set *for v if there is an enumeration $i_1, i_2, \ldots, i_k$ of elements of S such that $v_{s_j} \leq v_{i_j} \leq v_{s_j} + \varepsilon$ for all $j \in [k]$.*

The following incrementability property plays a crucial role in finding weak approximate minimum index sets. Essentially, it states that from a weak (k, ε)-approximate minimum index set, we can add an index whose value is no more than the $(k+1)$-smallest value plus ε, yielding a weak $(k+1, \varepsilon)$-approximate minimum index set. This is also stated and proved in [16, Proposition A.2] in the full version.

▶ **Proposition 12** (Error-tolerant Incrementability). *Let $v \in \mathbb{R}^n_{\geq 0}$ be an n-dimensional vector, and its coordinates can be sorted as $v_{s_1} \leq v_{s_2} \leq \cdots \leq v_{s_n}$. Suppose S is a weak (k, ε)-approximate minimum index set for a positive integer $k \in [n-1]$ and a positive real ε. Let $a_{k+1} \in [n] - S$ be an index satisfying $v_{a_{k+1}} \leq v_{s_{k+1}} + \varepsilon$. Then, $S \cup \{a_{k+1}\}$ is a weak $(k+1, \varepsilon)$-approximate minimum index set for v.*

Next, we introduce a strong notion of approximate minimum index set, as the weaker version might not be adequate for certain applications. The definition builds upon the following characterization of a k-minimum index set: For a set $S \subseteq [n]$ of size k that constitutes a k-minimum index set, it holds that $v_j \geq v_i$ for every $i \in S$ and $j \in [n] - S$. In other words, the inequality $\max_{i \in S} v_i \leq v_j$ must be satisfied for every $j \in [n] - S$. If we relax these constraints to allow for an epsilon margin of error, we obtain the following definition.

▶ **Definition 13** (Strong Approximate Minimum Index Set). *Let $v = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n_{\geq 0}$ be an n-dimensional vector. Then, a set $S = \{i_1, i_2, \ldots, i_k\} \subseteq [n]$ of size k is called a* strong (k, ε)-approximate minimum index set *for v, if for all $j \in [n] - S$, we have $\max_{i \in S} v_i \leq v_j + \varepsilon$.*

As the name suggests, a strong (k, ε)-approximate minimum index set is also a weak (k, ε)-approximate minimum index set, and we formally state this proposition as follows, which is also stated and proved in [16, Proposition A.7] in the full version.

▶ **Proposition 14.** *For an n-dimensional vector $v \in \mathbb{R}^n_{\geq 0}$, let $S \subseteq [n]$ with size k be a strong (k, ε)-approximate minimum index set for v. Then, S is also a weak (k, ε)-approximate minimum index set for v.*

However, a weak (k, ε)-approximate minimum index set is not necessarily a strong one. In particular, the strong approximate minimum index set is not error-tolerant incrementable, which is shown in the following example.

▶ **Example 15** (No Error-Tolerant Incrementability for Strong Approximate Minimum Index Set). Consider a 5-dimensional vector $v = (0.1, 0.2, 0.3, 0.4, 0.5)$. Let $\varepsilon = 0.15$. By definition, $\{2\}$ is a strong $(1, \varepsilon)$-approximate minimum index set for v. Also, for the index 3, we know $v_3 = 0.3 \leq v_2 + \varepsilon$. Therefore, by definition, $\{2, 3\}$ is a weak $(2, \varepsilon)$-approximate minimum index set for v. However, since $0.3 > 0.1 + \varepsilon$, $\{2, 3\}$ is *not* a strong $(2, \varepsilon)$-approximate minimum index set for v.

This illustration could be extended to encompass arbitrary values of n and k, demonstrating that for any fixed constant $c \geq 1$, there exists a weak (k, ε)-approximate minimum index set that is not a strong $(k, c\varepsilon)$-approximate minimum index set.

5 Quantum Algorithms for Finding Weak Approximate Minimum Index Sets

In this section, we will develop an approximate version of k-minimum finding that enables us to find a weak (k, ε)-approximate minimum index set, given access to an *approximate oracle*. The technique of generalized minimum finding from [40] is an important subroutine in our algorithm, which could be stated as follows.

▶ **Theorem 16** (Generalized Minimum Finding, [40, Theorem 49]). *Let n be a positive integer, and $\delta \in (0, 1/3)$. Suppose U is an $n \times n$ unitary operator satisfying $U|0\rangle = \sum_{\ell=1}^{n} \sqrt{p_\ell}|\psi_\ell\rangle|x_\ell\rangle$, where $p_\ell \in [0, 1]$, $\sum_{\ell=1}^{n} p_\ell = 1$, $|\psi_\ell\rangle$ are normalized states, and $\{ x_\ell : \ell \in [n] \}$ are distinct real numbers satisfying $x_1 < x_2 < \cdots < x_n$. Let X denote a discrete random variable with $p_\ell = \Pr(X = x_\ell)$, and $M \in (0, 1]$ be a lower bound on the probability $\Pr(X \leq m)$ for some $m \in \mathbb{R}$. Then, there exists a quantum algorithm $\mathsf{FindMin}(U, M, \delta)$ that, with probability at least $1 - \delta$, outputs a state $|\psi_i\rangle|x_i\rangle$ where $x_i \leq m$, using $\widetilde{O}(1/\sqrt{M})$ queries to U.*

Using this theorem, we can establish the following result for finding a weak (k, ε)-approximate minimum index set, extending and generalizing the approximate minimum finding procedure proposed in [40]. To express the algorithm more compactly, let $\mathsf{Uniform}$ be a unitary such that $\mathsf{Uniform}|0\rangle = \frac{1}{\sqrt{n}} \sum_{j=1}^{n} |j\rangle$, which can be implemented by $\widetilde{O}(1)$ one- and two-qubit gates.

Algorithm 1 $\mathsf{FindApproxWeakMin}(V, k, \varepsilon, \delta)$.

Require: An n-dimensional unitary V, $k \in [n]$, $\varepsilon \in (0, 1/2)$, $\delta \in (0, 1)$, with V being an (ε, δ)-approximate oracle of an n-dimensional vector v.
Ensure: A weak (k, ε)-approximate minimum index set S for v.
1: $S \leftarrow \emptyset, \mathcal{S} \leftarrow \mathsf{QSet.Initialize}(n)$.
2: **for** $t = k, \ldots, 1$ **do**
3: $\quad |\psi_\nu\rangle|\nu\rangle \leftarrow \mathsf{FindMin}(\mathcal{S}.\mathsf{Hide}() \cdot V \cdot \mathsf{Uniform}, t/n, \delta/k)$.
4: $\quad$ Perform computational measurement on the state $|\psi_\nu\rangle$, let the result be w_t.
5: $\quad S \leftarrow S \cup \{w_t\}, \mathcal{S}.\mathsf{Add}(w_t)$.
6: **end for**
7: Return set S.

▶ **Theorem 17** (Approximate Weak Minimum Index Set Finding). *Suppose $v = (v_1, \ldots, v_n) \in \mathbb{R}^n_{\geq 0}$ is an n-dimensional vector with $v_j \in [0, 1]$ for $j \in [n]$. For $\varepsilon \in [0, 1/2)$ and $\delta \in [0, 1/2)$, let V be a d-dimensional (ε, δ)-approximate oracle for v, Then, there is a quantum algorithm $\mathsf{FindApproxWeakMin}(V, k, \varepsilon, \delta)$ that, with probability at least $1 - \delta - \widetilde{O}(nd\delta\sqrt{kd})$, outputs a weak $(k, O(\varepsilon))$-approximate minimum index set S for v, using $\widetilde{O}(\sqrt{nk})$ queries to V.*

The proof of this theorem is given in the full version (see [16, Theorem B.4]).

We further note that, the failure probability could be made to $1 - O(\delta)$ by introducing an extra logarithmic multiplicative term in query complexity as Proposition 7 suggests.

6 Quantum Algorithms for Finding Strong Approximate Minimum Index Set

In this section, we present a quantum algorithm, $\mathsf{FindApproxStrongMin}$, which returns a strong approximate minimum index set. This algorithm builds on the method used for finding a weak approximate minimum index set.

The core idea of our algorithm is to first employ the algorithm FindApproxWeakMin for an estimate v_g of the k-th smallest entry of v, and then repeatedly sample from an "amplified" approximate oracle which facilitates querying entries that are approximately less than $v_g - 5\varepsilon$. The algorithm, explicitly given in Algorithm 2 uses two key subroutines $\mathsf{QCount}(V, c, \delta)$ and $\mathsf{AmpSamp}(V, c, \delta)$ where V is an approximate oracle for the vector v, c is a positive number, and $\delta \in [0, 1/2]$. The goal of $\mathsf{QCount}(V, c, \delta)$ is to approximately estimate the amplitude of all indices whose estimated value is no more than c, while the goal of $\mathsf{AmpSamp}(V, c, \delta)$ is to perform amplitude amplification on this state to achieve constant success probability. The formal statements of these procedures can be found in the full version, particularly in [16, Proposition C.3, Proposition C.6].

To describe the algorithm concisely, we use $\mathsf{ApproxQuery}(V, i)$ to denote a single computational measurement result of the state $V|i\rangle|0\rangle$. For an index set $S \subseteq [n]$, $\mathsf{ApproxQuery}(V, S)$ represents the collection of computational measurement outcomes of $V|i\rangle|0\rangle$ for all $i \in S$.

Algorithm 2 $\mathsf{FindApproxStrongMin}(V, k, \varepsilon, \delta)$.

Require: An n-dimensional unitary V, $k \in [n]$, $\varepsilon \in (0, 1/2)$, $\delta, \delta_0 \in (0, 1)$, with V being an (ε, δ_0)-approximate oracle of an n-dimensional vector v.

Ensure: A set S of size k being a strong $(k, O(\varepsilon))$-approximate minimum index set for v.

1: $S_0 \leftarrow \mathsf{FindApproxWeakMin}(V, k, \varepsilon, \delta/10)$.
2: $v' \leftarrow \mathsf{ApproxQuery}(V, S_0)$.
3: $v_g \leftarrow \max_{i \in S_0} v'_i$.
4: $\ell \leftarrow \mathsf{QCount}(U, v_g - 5\varepsilon, \delta/10)$, $R \leftarrow \emptyset$.
5: **for** $t = 1, 2, \ldots, \Theta(\ell \log(k) \log(10/\delta))$ **do**
6: $\quad |\phi\rangle \leftarrow \mathsf{AmpSamp}(V, v_g - 5\varepsilon, \delta/5n\ell)$, measure the first register of the state $|\phi\rangle$ in computational basis, and add the result to the set R.
7: **end for**
8: $\widetilde{v} \leftarrow \mathsf{ApproxQuery}(V, R \cup S_0)$.
9: **return** the index set of the k smallest elements in $\{\, \widetilde{v}_i : i \in R \cup S_0 \,\}$.

The following theorem states the correctness and the time efficiency of this algorithm.

▶ **Theorem 18** (Approximate Strong Minimum Index Set Finding). *Suppose $v = (v_1, \ldots, v_n) \in \mathbb{R}^n_{\geq 0}$ is an n-dimensional vector with $v_j \in [0, 1]$ for $j \in [n]$. For $\varepsilon \in [0, 1/2)$, $\delta \in [0, 1/2)$, and $\delta_0 \in [0, 1/2)$, let V be a d-dimensional (ε, δ_0)-approximate oracle for v, Then, there is a quantum algorithm $\mathsf{FindApproxStrongMin}(V, k, \varepsilon, \delta)$ that, with probability at least $1 - \delta - \widetilde{O}(nd\delta_0\sqrt{kd})$, outputs a strong $(k, O(\varepsilon))$-approximate minimum index set S for v, using $\widetilde{O}(\sqrt{nk})$ queries to V.*

Due to the length limit, we leave a complete proof to Appendix C in [16] in the full version.

7 Applications

7.1 Approximate k-minimum Expectations

As an application of our quantum approximate k-minimum finding algorithms, we address the problem of finding approximate k minima in multiple expectations. This can be directly solved by combining our algorithms with quantum amplitude estimation. The formal statement of the problem is as follows.

▶ **Definition 19** (Approximate k-minimum Expectations). *For a positive integer n, let $\{\rho_i\}_{i=1}^n$ be a set of s-qubit density matrices, and $\{O_i\}_{i=1}^n$ be a set of s-qubit positive semi-definite observables with $\|O_i\| \leq 1$ for $i \in [n]$. Given the unitary U being the purified access to ρ_i, i.e., $U|i\rangle_A|0\rangle_B|0\rangle_C = |i\rangle_A|\rho_i\rangle_{BC}$, such that $\mathrm{tr}_C(|\rho_i\rangle\langle\rho_i|) = \rho_i$, and a unitary $V = \sum_{j\in[n]}|j\rangle\langle j| \otimes V_j$, where V_j is a 1-block-encoding of the observable O_j for each $j \in [n]$. The* approximate k-minimum expectations *problem requires to find a strong (k,ε)-approximate minimum index set for the vector v with coordinates $v_i = \mathrm{tr}(O_i\rho_i)$.*

By leveraging our algorithms for finding strong approximate minimum index sets alongside quantum expectation estimation algorithms, we present the following theorem, which effectively addresses the problem of finding approximate k-minimum expectations.

▶ **Theorem 20.** *For a positive integer n, let $\{\rho_i\}_{i=1}^n$ be a set of s-qubit density matrices with purified access U_ρ, and let $\{O_i\}_{i=1}^n$ be a set of s-qubit positive semi-definite observables with block-encoding access V where $\|O_i\| \leq 1$ for $i \in [n]$. Then, there is a quantum algorithm* FindApproxExpectationMin$(U_\rho, V, k, \varepsilon, \delta)$ *that, with probability at least $1-\delta$, finds a strong (k,ε)-approximate minimum index set for the vector v with coordinates $v_i = \mathrm{tr}(O_i\rho_i)$, using $\widetilde{O}(\sqrt{nk}/\varepsilon)$ queries to U_ρ and V.*

Proof. See [16, Theorem D.5] in the full version. ◀

7.2 Approximate k-ground Energy Problem

We now turn to the approximate k-ground state energy problem, a generalization of the task of approximating the ground state energy of a Hamiltonian. The formal definition of the problem is as follows.

▶ **Definition 21** (Approximate k-ground State Energy). *Let H be a Hamiltonian acting on n-dimensional Hilbert space satisfying $\|H\| \leq \beta$, with a spectral decomposition $H = \sum_{j=1}^n \lambda_j|\varphi_j\rangle\langle\varphi_j|$. Assume that the eigenvalues $\{\lambda_j\}_{j=1}^n$ of H can be sorted as $\lambda_{s_1} \leq \lambda_{s_2} \leq \cdots \leq \lambda_{s_n}$. Let a unitary U_{enc} be an 1-block encoding of H, and U_{basis} be a unitary satisfying $U_{\mathrm{basis}}|j\rangle|0\rangle = |j\rangle|\varphi_j\rangle$. For a positive integer k and a positive real number ε, the* approximate (k,ε)-ground state energy *problem is to find a strong (k,ε)-approximate minimum index set for the vector v with coordinates $v_i = \lambda_i$.*

By integrating our quantum algorithms for finding strong approximate minimum index sets with quantum phase estimation, we present the following theorem that effectively solves the problem of finding approximate k-ground state energy, as also stated and proved in [16, Theorem D.9] in the full version.

▶ **Theorem 22.** *For $k \in \mathbb{N}^+$ and $\varepsilon, \delta \in (0, 1/3)$, there is a quantum algorithm that, with probability at least $1-\delta$, solves the problem defined in Definition 21 using $\widetilde{O}(\beta\sqrt{nk}/\varepsilon)$ queries to U_{enc} and U_{basis}.*

▶ Remark 23. The algorithm described in Theorem 22 can be employed in the case when the Hamiltonian is diagonal (with respect to the computational basis, i.e., $U_{\mathrm{basis}}|j\rangle|0\rangle = |j\rangle|j\rangle$). This scenario aligns precisely with the particular case discussed in Line 2 of Algorithm 2 in [15]. We therefore address the issue noted in Remark 2.

8 Discussion

In this paper, we carefully examined quantum algorithms for k-minimum finding with access to approximate oracles for function values. We introduced two distinct notions of approximate minimum index sets. The weak notion is defined entry-wise and possesses an error-tolerant

incrementability property, which facilitates the design of efficient algorithms. However, this notion may be insufficient for certain applications. To address this, we introduced a stronger notion of approximate minimum index sets and developed algorithms by combining the weak k-minimum finding algorithm with a coupon collector argument. We also discussed two potential applications related to the expectations of observables and the ground energy of Hamiltonians. Several intriguing open problems remain as a result of our work.

- The query complexity of the approximate minimum index set finding problem is not optimal yet. Is it possible to remove the extra logarithmic factors from the query complexity in Theorem 1?
- Can we approximate the k ground state energies of a Hamiltonian without knowing its eigenbasis?
- Can we find more types of input oracles for the minimum finding problem which are reasonable abstractions of practical scenarios? Is it possible to design algorithms for minimum finding using these oracles with optimal query complexity?
- Can we find more applications of quantum approximate k-minimum finding? We believe that our quantum algorithms in Theorem 1 could be applicable in other scenarios.

References

1 Scott Aaronson. Shadow tomography of quantum states. *SIAM Journal on Computing*, 49(5):STOC18–368–STOC18–394, 2020. doi:10.1137/18M120275X.

2 Dorit Aharonov, Vaughan Jones, and Zeph Landau. A polynomial quantum algorithm for approximating the Jones polynomial. *Algorithmica*, 55(3):395–421, 2009. doi:10.1007/s00453-008-9168-0.

3 Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Quantum speed-up for unsupervised learning. *Machine Learning*, 90(2):261–287, 2013. doi:10.1007/s10994-012-5316-5.

4 Shyan Akmal and Ce Jin. Near-optimal quantum algorithms for string problems. *Algorithmica*, 85(8):2260–2317, 2023. doi:10.1007/s00453-022-01092-x.

5 Andris Ambainis, Arturs Bačkurs, Nikolajs Nahimovs, and Alexander Rivosh. Grover's algorithm with errors. In *Proceedings of the 8th International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 180–189, 2012. doi:10.1007/978-3-642-36046-6_17.

6 Simon Apers and Ronald de Wolf. Quantum speedup for graph sparsification, cut approximation, and Laplacian solving. *SIAM Journal on Computing*, 51(6):1703–1742, 2022. doi:10.1137/21M1391018.

7 Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017. doi:10.1038/nature23474.

8 Fernando G. S. L. Brandão, Amir Kalev, Tongyang Li, Cedric Yen-Yu Lin, Krysta M. Svore, and Xiaodi Wu. Quantum SDP solvers: Large speed-ups, optimality, and applications to quantum learning. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming*, volume 132, pages 27:1–27:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.27.

9 Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Information*, volume 305, pages 53–74. American Mathematical Society, 2002. doi:10.1090/conm/305/05215.

10 Costin Bădescu and Ryan O'Donnell. Improved quantum data analysis. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1398–1411. Association for Computing Machinery, 2021. doi:10.1145/3406325.3451109.

11 Yanlin Chen and Ronald de Wolf. Quantum Algorithms and Lower Bounds for Linear Regression with Norm Constraints. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming*, volume 261, pages 38:1–38:21, 2023. doi:10.4230/LIPIcs.ICALP.2023.38.

12 Ronald de Wolf. Quantum computing: Lecture notes, 2023. arXiv:1907.09415.

13 Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006. doi:10.1137/050644719.

14 Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. ArXiv e-prints, 1996. arXiv:quant-ph/9607014.

15 Minbo Gao, Zhengfeng Ji, Tongyang Li, and Qisheng Wang. Logarithmic-regret quantum learning algorithms for zero-sum games. In *Advances in Neural Information Processing Systems*, volume 36, pages 31177–31203, 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/637df18481a6aa74238bd2cafff94cb9-Abstract-Conference.html.

16 Minbo Gao, Zhengfeng Ji, and Qisheng Wang. Quantum approximate k-minimum finding, 2024. The full version of this paper also includes references [12, 18, 28, 29, 33, 41]. arXiv:2412.16586.

17 András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 193–204, 2019. doi:10.1145/3313276.3316366.

18 András Gilyén and Alexander Poremba. Improved quantum algorithms for fidelity estimation, 2022. arXiv:2203.15993.

19 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996. doi:10.1145/237814.237866.

20 Yassine Hamoudi, Qipeng Liu, and Makrand Sinha. The NISQ complexity of collision finding. In *Proceedings of the 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–32, 2024. doi:10.1007/978-3-031-58737-5_1.

21 Peter Høyer, Michele Mosca, and Ronald de Wolf. Quantum search on bounded-error inputs. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming*, pages 291–299, 2003. doi:10.1007/3-540-45061-0_25.

22 Hsin-Yuan Huang, Richard Kueng, and John Preskill. Predicting many properties of a quantum system from very few measurements. *Nature Physics*, 16(10):1050–1057, 2020. doi:10.1038/s41567-020-0932-7.

23 William J. Huggins, Kianna Wan, Jarrod McClean, Thomas E. O'Brien, Nathan Wiebe, and Ryan Babbush. Nearly optimal quantum algorithm for estimating multiple expectation values. *Physical Review Letters*, 129:240501, December 2022. doi:10.1103/PhysRevLett.129.240501.

24 Ce Jin and Jakob Nogler. Quantum speed-ups for string synchronizing sets, longest common substring, and k-mismatch matching. *ACM Transactions on Algorithms*, 20(4):32:1–32:36, 2024. doi:10.1145/3672395.

25 A. Yu. Kitaev. Quantum measurements and the Abelian stabilizer problem. ArXiv e-prints, 1995. arXiv:quant-ph/9511026.

26 Dmitry Kravchenko, Nikolajs Nahimovs, and Alexander Rivosh. Grover's search with faults on some marked elements. *International Journal of Foundations of Computer Science*, 29(4):647–662, 2018. doi:10.1142/S0129054118410095.

27 Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. ArXiv e-prints, 2013. arXiv:1307.0411.

28 Guang Hao Low and Isaac L. Chuang. Optimal hamiltonian simulation by quantum signal processing. *Physical Review Letters*, 118:010501, January 2017. doi:10.1103/PhysRevLett.118.010501.

29 Guang Hao Low and Isaac L. Chuang. Hamiltonian Simulation by Qubitization. *Quantum*, 3:163, July 2019. doi:10.22331/q-2019-07-12-163.

30 Daniel Nagaj, Pawel Wocjan, and Yong Zhang. Fast amplification of QMA. *Quantum Information and Computation*, 9(11–12):1053–1068, November 2009. doi:10.26421/QIC9.11-12-8.

31 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. doi:10.1017/CBO9780511976667.

32 Yihui Quek, Clement Canonne, and Patrick Rebentrost. Robust quantum minimum finding with an application to hypothesis selection. ArXiv e-prints, 2020. arXiv:2003.11777.

33 Patrick Rall. Quantum algorithms for estimating physical quantities using block encodings. *Phys. Rev. A*, 102:022408, August 2020. doi:10.1103/PhysRevA.102.022408.

34 H. Ramesh and V. Vinay. String matching in $\widetilde{O}(\sqrt{n}+\sqrt{m})$ quantum time. *Journal of Discrete Algorithms*, 1(1):103–110, 2003. doi:10.1016/S1570-8667(03)00010-8.

35 Ansis Rosmanis. Quantum search with noisy oracle. ArXiv e-prints, 2023. arXiv:2309.14944.

36 Ansis Rosmanis. Addendum to "quantum search with noisy oracle". ArXiv e-prints, 2024. arXiv:2405.11973.

37 Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015. doi:10.1080/00107514.2014.964942.

38 Neil Shenvi, Kenneth R. Brown, and K. Birgitta Whaley. Effects of a random noisy oracle on search algorithm complexity. *Physical Review A*, 68(5):052313, 2003. doi:10.1103/PhysRevA.68.052313.

39 Joran van Apeldoorn and András Gilyén. Improvements in quantum SDP-solving with applications. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming*, volume 132, pages 99:1–99:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.99.

40 Joran van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. Quantum SDP-Solvers: Better upper and lower bounds. *Quantum*, 4:230, 2020. doi:10.22331/q-2020-02-14-230.

41 Qisheng Wang. Optimal trace distance and fidelity estimations for pure quantum states. *IEEE Transactions on Information Theory*, 70(12):8791–8805, 2024. doi:10.1109/tit.2024.3447915.

42 Qisheng Wang and Mingsheng Ying. Quantum algorithm for lexicographically minimal string rotation. *Theory of Computing Systems*, 68(1):29–74, 2024. doi:10.1007/s00224-023-10146-8.

43 Nathan Wiebe, Ashish Kapoor, and Krysta M. Svore. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *Quantum Information and Computation*, 15(3–4):316–356, 2015. doi:10.26421/QIC15.3-4-7.

Beating Competitive Ratio 4 for Graphic Matroid Secretary

Kiarash Banihashem
University of Maryland, College Park, MD, USA

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

Dariusz R. Kowalski
Augusta University, GA, USA

Piotr Krysta
Augusta University, GA, USA
University of Liverpool, Liverpool, UK

Danny Mittal
University of Maryland, College Park, MD, USA

Jan Olkowski
University of Maryland, College Park, MD, USA

Abstract

One of the classic problems in online decision-making is the *secretary problem*, where the goal is to hire the best secretary out of n rankable applicants or, in a natural extension, to maximize the probability of selecting the largest number from a sequence arriving in random order. Many works have considered generalizations of this problem where one can accept multiple values subject to a combinatorial constraint. The seminal work of Babaioff, Immorlica, Kempe, and Kleinberg (SODA'07, JACM'18) proposed the *matroid secretary conjecture*, suggesting that there exists an $O(1)$-competitive algorithm for the matroid constraint, and many works since have attempted to obtain algorithms for both general matroids and specific classes of matroids. The ultimate goal of these results is to obtain an e-competitive algorithm, and the *strong matroid secretary conjecture* states that this is possible for general matroids.

One of the most important classes of matroids is the *graphic matroid*, where a set of edges in a graph is deemed independent if it contains no cycle. Given the rich combinatorial structure of graphs, obtaining algorithms for these matroids is often seen as a good first step towards solving the problem for general matroids. For matroid secretary, Babaioff et al. (SODA'07, JACM'18) first studied graphic matroid case and obtained a 16-competitive algorithm. Subsequent works have improved the competitive ratio, most recently to 4 by Soto, Turkieltaub, and Verdugo (SODA'18).

In this paper, we break the 4-competitive barrier for the problem, obtaining a new algorithm with a competitive ratio of 3.95. For the special case of simple graphs (i.e., graphs that do not contain parallel edges) we further improve this to 3.77. Intuitively, solving the problem for simple graphs is easier as they do not contain cycles of length two. A natural question that arises is whether we can obtain a ratio arbitrarily close to e by assuming the graph has a large enough girth.

We answer this question affirmatively, proving that one can obtain a competitive ratio arbitrarily close to e even for constant values of girth, providing further evidence for the strong matroid secretary conjecture. We further show that this bound is tight: for any constant g, one cannot obtain a competitive ratio better than e even if we assume that the input graph has girth at least g. To our knowledge, such a bound was not previously known even for simple graphs.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, graphic matroids, secretary problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.52

Related Version *Full Version*: https://arxiv.org/abs/2501.08846



© Kiarash Banihashem, MohammadTaghi Hajiaghayi, Dariusz R. Kowalski, Piotr Krysta, Danny Mittal, and Jan Olkwoski;
licensed under Creative Commons License CC-BY 4.0

33rd Annual European Symposium on Algorithms (ESA 2025).
Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman; Article No. 52; pp. 52:1–52:16



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

Funding The work is partially supported by DARPA QuICC, ONR MURI 2024 award on Algorithms, Learning, and Game Theory, Army-Research Laboratory (ARL) grant W911NF2410052, NSF AF:Small grants 2218678, 2114269, 2347322, and Royal Society grant IES\R2\222170.

1 Introduction

In the past two decades there has been a renewed interest in online item selection problems where a sequence of items arrive one by one, revealing their *weight*, and a decision maker needs to irrevocably decide whether or not to accept each item as it arrives. The goal is to maximize the total accepted weight, subject to a feasibility constraint on the chosen items. An algorithm's performance is typically measured by its *competitive ratio*, which compares the algorithm's total weight with the offline optimum – the total weight achievable if all item weights were known in advance. These problems are appealing both from a mathematical perspective, as they are concise models for online decision making, and from an economical perspective, as they have close connection to pricing and auction theory [30, 29, 8, 12].

In the absence of any information about future items, the problem is essentially hopeless as any single item could be significantly heavier than all others, and the decision maker has no way of deciding which it is. As such, most works make distributional assumptions on either the weight of the arriving items, or their arrival order. The former class of problems are generally referred to as *prophet inequalities* while the latter are known as *secretary problems*. Numerous works have studied secretary problems for a large class of combinatorial constraints [38, 19, 34, 33, 20, 44, 46, 23, 31] and objective functions [10, 25], and considered close variants of these problems such as the prophet-secretary problem [22].

Perhaps the most important open question in the area of online decision making is the *matroid secretary problem* posed by the seminal work of Babaioff, Immorlica, Kempe, and Kleinberg (SODA'07,JACM'18) [8, 7]. In the matroid secretary problem, items arrive in a random order, and each item corresponds to an element in a matroid $M = (E, I)$,where E is the set of elements and I is the collection of independent sets in matroid M. Upon arrival, the weight of each item is revealed, and the decision maker must immediately decide whether to accept or reject it. The goal is to maximize the total weight of accepted items, with the constraint that the selected items form an independent set in the matroid. The matroid secretary conjecture [8, 7] states that there exists a constant-competitive ratio algorithm for this problem, and the *strong matroid secretary conjecture* (e.g., see [46]) states that there exists an e-competitive algorithm. Many works have studied the problem for both general matroids and specific cases.

In this paper, we focus on the specific case of graphic matroids. In this case, the arriving items correspond to the edges of a graph and the goal is to accept a set of edges that do not contain a cycle. Solving problems for graphic matroids is often viewed as a promising first step toward addressing arbitrary matroids, as graphs possess rich combinatorial structures and graphic matroids, along with linear matroids, are among the most intuitive examples of non-trivial matroids. Many counterexamples for candidate matroid secretary algorithms are, in fact, graphic matroids [8, 9]. It is common to illustrate the main ideas behind general algorithms by showing their behavior on this specific case [47]. Recent work has also explored whether techniques for graphic matroids can be extended to general matroids [1].

The seminal paper of Babaioff et al. originally studied the graphic matroid secretary problem, obtaining a 16-competitive algorithm for the problem. Babaioff et. al. [6] designed a $3e$-competitive algorithm. The competitive ratio was later improved to $2e$ by Korula and Pal [38] and later to 4 by Soto, Turkieltaub, and Verdugo [46]. Whether or not the ratio can be improved has been open at least since 2018.[1]

1.1 Our results and techniques

In this paper we obtain an algorithm with competitive ratio 3.95 for the problem, breaking the 4-competitive barrier. We further improve this result for the specific case of *simple graphs*, i.e., graphs that do not have parallel edges. Intuitively, simple graphs represent an easier special case as they don't have any cycles of length 2; since the algorithm is forbidden from accepting edges that form a cycle, the lack of 2-cycles gives the algorithm more freedom to accept edges. Our main result is the following theorem.

▶ **Theorem 1.** *There exists a* 3.95*-competitive algorithm for the graphic matroid secretary problem. Furthermore, if the input graph is assumed to be simple, there exists an algorithm with competitive ratio* 3.77*.*

The basic approach for this result is to, at each step, compute a set of outgoing edges such that each node has exactly one outgoing edge. More exactly, at each step we compute a maximum spanning forest, then direct the edges in this forest toward an arbitrary root; each node then has a unique outgoing edge. We then argue that if the algorithm only accepts an edge for whose endpoints it has not previously accepted an outgoing edge, then the algorithm's set of taken edges will always be independent.

Given this, the key aspect of Algorithm 1 that allows it to obtain an improved competitive ratio is a slightly stronger condition used to determine whether an edge is taken. Specifically, while for one endpoint we only demand that we have not taken an outgoing edge, for the other endpoint we demand that we have not even seen an outgoing edge. We can then show an increased probability for the former condition being satisfied for a currently considered edge by using the fact that the latter condition may not be satisfied by a previously seen outgoing edge, causing said edge to not be taken. Algorithm 1 furthermore makes use of random choice in determining which endpoint to apply the stronger condition to, which is crucial for handling the case of duplicate edges. When the input graph is guaranteed to be simple, this random choice is unnecessary – removing this random choice gives our algorithm for simple graphs (see the full version of the paper) which allows us to obtain an even lower competitive ratio.

Motivated by the improvement for simple graphs, we additionally study, for the first time, the landscape of the graphic matroid secretary problem for graphs of high *girth*, where we recall that the girth of a graph is the length of its shortest cycle. We note that the fact that a graph is simple follows from the assumption that girth is at least 3. We show that when the graph has large (but constant) girth, one can obtain a competitive ratio arbitrarily close to e. Formally, we prove the following theorem.

▶ **Theorem 2.** *For any graph G with girth at least g, there exists an $(e + o_g(1))$-competitive algorithm.*

[1] This paper was previously submitted to STOC'25 on November 4, 2024. Independently of our work, on November 18, 2024, Bérczi et al. [11] posted a preprint on arXiv, obtaining a 3.99-competitive algorithm and a 3.71-competitive algorithm for general graphs and simple graphs, respectively.

The result builds on yet another property of the combinatorial structure of the set of outgoing edges introduced in the proof of Theorem 1. We first observe that the fact that every vertex has at most one outgoing edge implies that, in a graph induced by the outgoing edges, each edge belongs to at most one cycle. On the other hand, we prove that if Algorithm 1 were to accept outgoing edges without respecting the graphic matroid independence condition, it would yield a higher acceptance probability of $\frac{1}{e}$ for an edge from the maximum independent set. These observations lead to a natural approach: accepting an outgoing edge only with a certain probability such that, for every cycle of length g, the probabilities of taking each edge of this cycle are equal. Intuitively, as the length of the shortest cycle increases, this probability tends towards $\frac{1}{e}$. We refer the reader to the full version of the paper for more details.

The best competitive ratios we attain over the three algorithms we introduce are listed in Table 1 for girths less than 10.

Table 1 The best competitive ratio we obtain for graphic matroid secretary when the input graph is restricted to have girth at least g for $g < 10$. $g = 2$ corresponds to multigraphs, while $g = 3$ corresponds to simple graphs. As g approaches infinity, the competitive ratio approaches e. See the full version of the paper for the algorithms for simple graphs and high girth graphs.

Girth (g)	Competitive Ratio	Algorithm
2	3.95	Algorithm 1
3	3.77	Simple graph algorithm
4	3.77	Simple graph algorithm
5	3.76	High girth algorithm
6	3.61	High girth algorithm
7	3.50	High girth algorithm
8	3.42	High girth algorithm
9	3.35	High girth algorithm

Perhaps more surprisingly, we show that this is tight: no algorithm can achieve a competitive ratio better than e, even for graphs with high girth. To our knowledge, this lower bound was not previously known even for simple graphs, and our techniques may be of independent interest for related online arrival problems.

▶ **Theorem 3.** *For any $g \in \mathbb{N}$, there does not exist an algorithm for the graphic matroid secretary problem on graphs of girth at least g that obtains competitive ratio less than e.*

The two results fully characterize the landscape of the secretary problem in the high-girth setting, essentially showing that the problem becomes as easy as the single secretary problem in the limit. To prove the lower bound in Theorem 3, we first show that weighted single secretary, also known as a cardinal secretary, is hard "on average" in the following sense. We demonstrate the existence of a finite distribution over the instances of this problem such that even if the adversary samples instances, as opposed to choosing them in a worst-case manner, from that distribution, it is still hard for any algorithm to choose the maximum weight element.

We then construct a high girth bipartite graph based on Ramanujan graphs, where vertices of one bipartition have same degree d. The hard distribution of instances of weighted single secretary of size d can now be sampled independently and embedded on the edges incident to each degree-d vertex on one side of the constructed bipartite graph. Given an algorithm that performs well on this graph, we obtain an algorithm performing well on the original distribution of weighted single secretary by simulating the first algorithm and mimicking the choices that it makes on a single degree-d vertex.

Our construction of the input graph in addition to high-girth Ramanujan graphs uses the probabilistic method. Our construction of the sets of hard weights is based on an extension of a similar argument from an infinite to a finite version of Ramsey theorem. We employ zero-sum games duality argument to show an existence of the final probabilistic distribution over the instances.

2 Related work

Many other derivations and specific cases of the general matroid secretary problem have been given attention over the years. Hajiaghayi, Kleinberg and Parkes [30] had first introduced the *multiple-choice value version* of the problem, aka the uniform matroid secretary problem, in which the goal is to maximize the expected sum of the at most k selected numbers. Kleinberg [36] later presented a tight $(1-O(\sqrt{1/k}))^{-1}$-competitive algorithm for the k uniform secretary resolving an open problem of [30]. Transversal matroids were first considered in [8] who gave $4d$-competitive algorithm, where d is the degree of the transversal matroid. This was improved by Dimitrov and Plaxton [19] who showed a ratio of 16 for all transversal matroids. The optimal ratio of e follows from the e-competitive algorithm for secretarial online weighted bipartite matching problem [35], which generalizes the problem on transversal matroids. For the laminar matroids, a long line of work led to $3\sqrt{3}e$ competitive ratio [32, 33, 41], which was improved to 9.6 in [41], $3\sqrt{3}$ in [46], 4.75 in [31], and 3.26 in [11]. The challenging general class of regular matroids was proven to admit $9e$-competitive algorithm [20].

Other generalizations of the secretary problem such as the submodular variant have been initially studied by the Bateni, Hajiaghayi, and ZadiMoghaddam [10] and Gupta, Roth, Schoenebeck, and Talwar [28]. The connection between the secretary problem and online auction mechanisms has been explored by Kesselheim et al. [34], who give a e-competitive solution to the online bipartite weighted matching problem.

The *prophet secretary* problem is another well-studied variant of the secretary problem, closely related to prophet inequalities. In the *prophet inequality* setting, introduced by Krengel and Sucheston [39, 40], we know the distributions of n arriving items and aim to maximize the ratio of the expectation of the selected item's value to the expectation of the sequence maximum, with a tight competitive ratio of 2. Research connecting prophet inequalities and online auctions, initiated by Hajiaghayi, Kleinberg, and Sandholm [29], led to follow-up studies such as Alaei, Hajiaghayi, and Liaghat's work [3] on the bipartite matching variant of prophet inequality, also achieving a competitive ratio of 2 [3]. Feldman et al. [24] expanded the problem to combinatorial auctions with multiple buyers, achieving the same bound through a posted pricing scheme, and Kleinberg and Weinberg [37] extended this result to matroids with a 2-competitive algorithm. A recent paper explores matroid structures that allow breaking the tight competitive ratio of 2 [4]. Contrary to results presented in this paper, they prove that in the general matroid setting, a high girth is not a sufficient condition while considering k-fold matroid unions suffices.

The prophet secretary model, introduced by Esfandiari, Hajiaghayi, Liaghat, and Monemizadeh [22], assumes a random arrival order and known item distributions. They designed an algorithm achieving a competitive factor of $(\frac{e}{e-1})$, which has proven challenging to improve. However, Azar et al. [5] and Correa et al. [16, 17] improved this bound to $(1-1/e+1/30)^{-1} \approx 1.502$. For the single-item i.i.d. case, Abolhasani et al. [2] achieved a 1.37-competitive ratio, later improved to 1.342 by Correa et al. [14]. Recently, Peng and Tang [43] achieved a 1.379-competitive algorithm for the *free order* case, however finding the tight competitive bound for the general prophet secretary problem remains open.

Last but not least, extensions beyond matroids for the secretary problem have also been studied. For example, Kleinberg and Weinberg [37] provided an $O(p)$-competitive algorithm for the intersection of p matroids, later generalized to polymatroids by Dütting and Kleinberg [21]. Rubinstein [44] and Rubinstein and Singla [45] considered prophet inequalities and secretary problems in arbitrary downward-closed set systems. For such settings, Babaioff et al. [8] proved a lower bound of $\Omega(\log n \log\log n)$, and further studies have explored combinatorial optimization applications [18, 26, 27, 42].

3 Preliminaries

Graph Notation

In this paper, we assume all graphs are undirected, weighted, and may contain multiple edges between the same pair of vertices. Specifically, a graph is defined as a triple $G = (V, E, w : E \to \mathbb{R})$, where V, with $|V| = n$, is the set of vertices; E, with $|E| = m$, is the multiset of edges; and w assigns weights to the edges. We assume graphs do not contain loops (i.e., edges connecting a vertex to itself[2]). Given any vertex $u \in V$, we denote the degree of u by $deg_G(u)$. A graph is called *simple* if there is at most one edge connecting any pair of vertices. For a graph G, we denote g as the *girth* of G, representing the length of its shortest cycle. In graphs with multiple edges, a cycle is defined as any multiset of edges $C = \{(a_1, b_1), \ldots, (a_k, b_k)\}$, for $k \geq 2$, such that $\forall_{1 \leq i \leq k-1}\, b_i = a_{i+1}$ and $b_k = a_1$. For an integer $g \geq 2$, let $\mathcal{G}_g$ denote the set of all graphs with girth at least g. We also denote $[n] = \{1, 2, \ldots, n\}$.

Matroids

A *matroid* $M = (E, I)$ is a combinatorial structure that generalizes independence. It consists of a finite set E and a collection I of independent subsets of E satisfying: (1) the empty set is independent, (2) any subset of an independent set is also independent, and (3) if one independent set is larger than another, an element from the larger set can extend the smaller one while preserving independence. These properties capture the concept of independence, making matroids useful for modeling optimization problems where we seek a maximum-weight or maximum-cardinality independent subset of elements.

A *graphic matroid* $\mathcal{M}(G)$ associated with a graph G has as its elements the edges in the multiset E, and defines independent sets as acyclic subsets of edges (i.e., subgraphs without cycles). The weight of an independent set is the sum of its edge weights.

Problems

Consider a graphic matroid $\mathcal{M}(G)$ associated with a *multigraph* $G = (V, E, w : E \to \mathbb{R})$. In the online secretary problem on graphic matroids, the elements of E are presented to the algorithm in a random order, chosen uniformly from all possible permutations of the multiset E. Elements arrive one at a time, effectively creating m time steps during the algorithm's execution. Upon the arrival of an element, the algorithm must decide whether to accept or reject it, with the constraint that an element can only be accepted if it forms an independent set with the already accepted elements. Decisions are irrevocable.

[2] As explained in the following paragraph, such edges are irrelevant in the context of graphic matroids.

The objective is to design an algorithm that maximizes the expected sum of the weights of the accepted elements, referred to as the algorithm's gain. For an algorithm ALG, we denote its (random) gain by ALG and its expected gain by $\mathbb{E}(ALG)$.

Let OPT denote an independent set with maximum weight in the matroid $\mathcal{M}(G)$. We say an algorithm ALG is α-*competitive* (or has an α competitive ratio, for some $\alpha \geq 1$) for a family of matroids if $\alpha \cdot \mathbb{E}(ALG) \geq W(OPT)$ for all matroids in that family, where $W(OPT)$ is the weight of OPT.

4 Improved bounds for graphic matroid secretary: proof of Theorem 1

In this section, we prove the following theorem, which contains the result from Theorem 1 for general graphs.

▶ **Theorem 4.** *There exists a* 3.95*-competitive algorithm for graphic matroid secretary.*

To prove the theorem, we present an algorithm that attains a competitive ratio less than 3.95 for the graphic matroid secretary problem, surpassing a result of Soto, Turkieltaub, and Verdugo [46] attaining a competitive ratio of 4 that previously stood for six years as the best result achieved for graphic matroid secretary; this algorithm is described in Section 4.1.

To prove Theorem 1 for simple graphs, we consider a slight modification of our algorithm. See the full version of the paper for details.

4.1 Improvement for general graphs

Algorithm 1 New algorithm for graphic matroid secretary.

Let E' be the first m' edges `// set of observed edges`
$A \leftarrow \emptyset$ `// set of accepted edges`
$\forall v \in V$: $seen_outgoing(v) \leftarrow False$
$\forall v \in V$: $taken_outgoing(v) \leftarrow False$
for $t \in \{ m'+1, \ldots, m \}$ **do**
  Let e_t be the edge arriving in time t
  Add e_t to E'
  $T_t^{\text{opt}} \leftarrow$ maximum weighted directed forest on $G[E']$
  $\forall v \in V : outgoing(v) \leftarrow$ the edge directed away from v in T_t^{opt} if it exists
  $\forall v \in V$ such that $outgoing(v) = null : outgoing(v) \leftarrow$ a unique edge in E' not in T_t^{opt}
  if $e_t \in T_t^{\text{opt}}$ **then**
    $e_t \leftarrow (u, v)$ `// where` e_t `is directed from` u `to` v `in` T_t^{opt}
    $(a, b) \leftarrow$ either (u, v) or (v, u) with equal probability
    if $seen_outgoing(a) = False$ *and* $taken_outgoing(b) = False$ **then**
      $taken_outgoing(u) \leftarrow True$
      add e_t to A
    end
  end
  if $outgoing(w) = e_t$ *for some* $w \in V$ **then**
    $seen_outgoing(w) \leftarrow True$
  end
end

The pseudocode of the algorithm is provided in Figure 1. The algorithm consists of two phases. During the first m' steps (where m' depends only on m), edges are only observed without being taken. In later steps, let e be the presented edge. The algorithm computes a maximum spanning forest T_t^{opt} of all edges seen so far. We assume the forest is rooted, where the root for each tree is chosen lexicographically for simplicity; how we choose the root does not affect the algorithm as long as it depends only on the set of the edges already seen and not their order. This forest is directed towards the root, so that every edge is directed and each node has at most one outgoing edge. We additionally compute an array of *outgoing* edges: for each node v, if v has an outgoing edge in T_t^{opt}, then that is its outgoing edge; if it does not, then we choose an arbitrary edge not already assigned to be the outgoing edge of another vertex [3] and declare that to be the outgoing edge from v. Note that said edge does not even have to have v as an endpoint. Also note that crucially, the choice of edges declared to be outgoing for the vertices that need it depends on the set of edges observed so far but not their order.

We only consider e if it is in T_t^{opt}. In this case, it points from some u to some v, and so is the outgoing edge for u. In order to take e, we randomly order u, v as a, b and impose the following constraints on e:

- No edge e' has been taken such that e' was the outgoing edge for b when e' was presented.
- No edge e' has appeared such that e' was the outgoing edge for a when e' was presented.

If e satisfies both constraints then it is taken. Otherwise, the algorithm still notes that e could have been selected for u.

The distinct constraints used for a, b are key – the stronger condition applied to endpoint a allows us to lower bound the probability that an edge is blocked from being taken, which can then be used to show an increased probability that the weaker condition for endpoint b is satisfied for an edge that we would like to be taken. The randomness in ordering u, v into a, b is also essential. To see why, consider the specific example of an edge $e_t \in OPT$ with endpoints u, v that is presented at step t, and consider some step $j < t$. Suppose that T_j^{opt} contains an edge e^u outgoing from u to a third node x and an edge e^v outgoing from v to a fourth node y. We would like to argue that in the case that either e^u or e^v is the edge e_j presented at step j, there is a possibility that some outgoing edge from x or y respectively had already been presented prior to step j, meaning that e^u or e^v could not have been taken.

This means that, for example, if e^u were the edge presented at step j, we would desire that, in the case of edge e_t, $b = u$, and in the case of edge $e_j = e^u$, $a = x$ (and so $b = u$), so that e^u could be blocked by an edge outgoing from x simply being presented, while e_t is then not blocked because e^u was not actually taken. This suggests that we could simply always take $b = u$ and $a = v$, applying the stronger condition to the endpoint which the edge is directed towards.

However, as in general graphs it is possible for multiple edge to have the same endpoints, it is possible that we in fact have $x = v$, in which case an edge outgoing from x would block e_t itself. This is illustrated in the top-left quadrant of Figure 1. Therefore, in order to guarantee an improvement, we crucially allow for the possibility that in the case of edge e_t, $b = v$, and in the case of edge $e_j = e^v$, $a = y$ – if e^u has the same endpoints as e_t, then e^v cannot as T_j^{opt} is a spanning tree, and so y will necessarily be distinct from u, meaning that there is a possibility for e^v to be blocked by an edge outgoing from y while still allowing e_t to be taken. This is illustrated in the bottom-left quadrant of Figure 1.

[3] Note that if m is too small (specifically, if $m' + 1 < n$) then there may not exist sufficiently many edges for this to be possible. However, if this is the case, we can simply mix in "dummy edges" that the algorithm can treat the same as real edges in order to cause $m' + 1$ to be at least n. The details of this mixing in are described in the proof of Theorem 4.

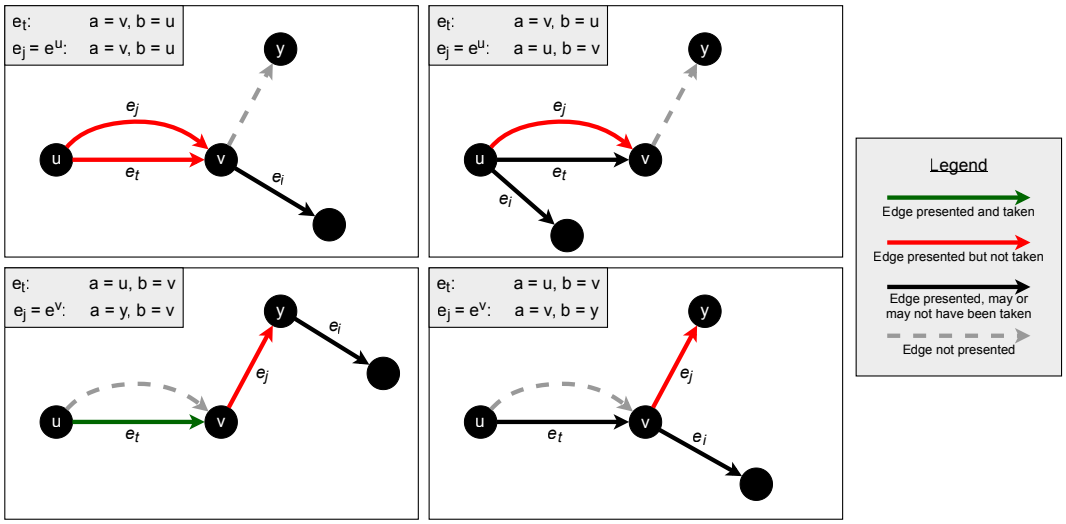


Figure 1 The described "worst-case" example for Algorithm 1. At step t, we are presented with an edge e_t in the optimum solution. e_t may be blocked from being taken due to an earlier step j. We suppose that the outgoing edge e^u from u in T_j^{opt} goes to v while the outgoing edge e^v from v in T_j^{opt} goes to a third vertex y; these are the only possible values of e_j with a potential to block e_t. All of e_t, e^u, e^v are depicted. Additionally depicted is an edge e_i outgoing from the endpoint of e_j selected as b – this edge being presented at step i would block e_j from being taken. Each of the four images depicts one equally likely possibility for the random choice of (a, b) in the case of e_t as well as in the case of e_j, where e_j is assumed to be the edge outgoing from b in T_j^{opt} (as this is the only case relevant for an increased probability arising from e_j not being taken). Only in one case is e_t guaranteed to be taken (assuming steps other than those depicted do not block e_t from being taken).

Analysis

We now proceed with a formal analysis. Due to space constraints, parts of the proof are deferred to the full version of the paper. We first show that any set of edges accepted by Algorithm 1 is independent, meaning that the algorithm is a valid graphic matroid secretary algorithm.

▶ **Lemma 5.** *The set A of accepted edges in Algorithm 1 is always an independent set in the graphic matroid.*

We now proceed to demonstrate that Algorithm 1 has the desired approximation factor for an appropriate choice of m'. We first define quantities f, g, h below that will be crucial for the analysis:

▶ **Definition 6.** *Define $f_{u,S}(i)$ to be the probability that after step i, seen_outgoing$(u) =$ False, given that S is the set of edges that are presented in steps $1, \ldots, i$.*

▶ **Definition 7.** *Define $g_{u,v,S}(i)$ to be the probability that after step i, both seen_outgoing$(u) =$ False and seen_outgoing$(v) =$ False, given that S is the set of edges that are presented in steps $1, \ldots, i$.*

▶ **Definition 8.** *Define $h_{u,v,S}(i)$ to be the probability that after step i, if we take (a, b) to be either (u, v) or (v, u) with equal probability, then seen_outgoing$(a) =$ False and taken_outgoing$(b) =$ False, given that S is the set of edges that are presented in steps $1, \ldots, i$.*

The condition used to define h is the same condition checked to see whether an edge between u and v can be taken.

A key tool for our analysis is the following. Note that for any node u, *seen_outgoing*(u) is set to $True$ when at some step j we observe an edge e such that e is in T_j^{opt} and is directed from u. Importantly, T_j^{opt} depends only on the set of edges presented up to step j and not their order. Therefore, conditioned on the set of edges that appears in steps $1, \ldots, j$, the probability that at step j we are presented with the outgoing edge from u is at most $\frac{1}{j}$ because there is exactly one such edge, and this edge, if it exists, is presented at step i with probability $\frac{1}{i}$.

The above idea is applied in the following lemma:

▶ **Lemma 9.** *For all vertices u, all i from m' to m, and all S with $|S| = i$, $f_{u,S}(i) = \frac{m'}{i}$.*

The next lemma uses the same idea to derive a similar expression for g:

▶ **Lemma 10.** *For all $u, v \in V$ and $i \in [m', m]$, and all S with $|S| = i$, $g_{u,v,S}(i) \geq \frac{m'(m'-1)}{i(i-1)}$.*

We now proceed to lower bound h; we again use the same idea but now additionally take advantage of the stronger condition applied to a in the algorithm to derive a bound for h superior to the one we derived for g by lower bounding the probability that an edge that would have been selected for one of u, v was not actually taken. As the lower bound for h is more complicated, we define its lower bound as an additional function h_δ.

▶ **Definition 11.** *Define $h_\delta(i)$ recursively by $h_\delta(m') = 1$ and $h_\delta(i) = (1 - \frac{2}{i})h_\delta(i-1) + \frac{\delta}{i} \cdot \frac{m'(m'-1)}{(i-1)(i-2)} \left[1 - \frac{m'}{i-1}\right]$.*

We parameterize h by δ as h will reused later with a different value of δ.

▶ **Lemma 12.** *For all vertices u, v, all i from m' to m, and all S with $|S| = i$, $h_{u,v,S}(i) \geq h_{\frac{1}{4}}(i)$.*

As mentioned before, the conditions defining $h_{u,v,S}(i)$ are identical to the conditions for taking an edge given that it is in T_i^{opt}. As any edge in the overall optimum is in T_i^{opt}, lower bounding the probability of such an edge being taken allows us to lower bound the overall approximation factor. This is expressed in the following lemma.

▶ **Lemma 13.** *Let $W(S)$ be the sum of the weights of edges in S, and recall that A is the set of edges accepted by Algorithm 1. Then,*

$$\frac{\mathbb{E}[W(A)]}{W(OPT)} \geq \frac{1}{m} \sum_{t=m'+1}^{m} h_{\frac{1}{4}}(t-1).$$

It now only remains to give a concrete lower bound for $\frac{1}{m}\sum_{t=m'+1}^{m} h_{\frac{1}{4}}(t-1)$. We proceed to derive such a bound in the following lemma.

▶ **Lemma 14.** *If we let $m' = \lfloor \alpha m \rfloor$, then*

$$\lim_{m\to\infty} \frac{1}{m} \sum_{i=m'+1}^{m} h_\delta(i-1) \geq \alpha - \alpha^2 + \delta\alpha^2 \ln \alpha + \frac{\delta}{2}\left(\alpha - \alpha^3\right).$$

Setting $\alpha = 0.4914$, we can show that the above lemma implies Theorem 4. We refer to the full version of the paper for the full proof.

5 Impossibility result for high girth graphs: proof of Theorem 3

In this section, we prove a hardness result for high-girth graphs. We begin by showing that the weighted single secretary problem remains hard even when the instance is drawn from a known distribution, using the Finite Ramsey Theorem and zero-sum game duality. We then reduce this to the weighted graphic matroid secretary problem on large-girth graphs via a construction based on Ramanujan graphs. Specifically, by carefully removing edges using the probabilistic method, we construct bipartite graphs with certain properties and embed hard instances of single secretary into them to obtain similarly hard graphic secretary instances. Due to space constraints, parts of the proofs are deferred to the full version of the paper.

We note that it is crucial that the hardness holds even when the distribution is known in advance, unlike deterministic instances that are only hard when the instance is hidden. This allows us to present multiple instances from the distribution without the algorithm gaining any advantage from earlier ones – since it already knows the distribution fully.

5.1 Hard inputs for single secretary

We first prove that no algorithm for the secretary problem can achieve a competitive ratio better than $1/e$, even if the weights are drawn from a known finite set.

▶ **Theorem 15.** *For any $\varepsilon > 0$ and for any $\rho > 1$, there exists an integer n and a finite set of numbers $W \subset \{1, \rho, \rho^2, \ldots\}$ over which no (randomized) algorithm for the single secretary problem with n items can choose the maximum with probability $\frac{1}{e} + \varepsilon$.*

See the full version of the paper for the proof. The result essentially serves as the starting point of our proof as it provides "hard instances" of the secretary problem which we then carefully embed in graphs of large girth. Correa et al. [15, 13] previously proved this theorem for infinite sets; our proof modifies their technique using the finite version of Ramsey's theorem.

While the generalization to finite sets may seem like a small technical issue, it has important consequences. As we will see, the finite assumption is necessary as it allows us to invoke the Minimax theorem for finite games. This in turn allows us to guarantee that there is a hard distribution over inputs, rather than just a single instance. As we will need to simultaneously embed multiple "independent" hard instances of the single secretary problem to obtain a hard instance of the graphical matroid secretary, this generalization is necessary.

We will now use the Minimax principle and Theorem 15 to prove the following.

▶ **Theorem 16.** *For any $\varepsilon > 0$ and for any $\rho > 1$, there exists an integer n, a finite set $W \subset \{1, \rho, \rho^2, \ldots\}$, and a finite distribution D over all subsets of size n from W with the following property. If a random instance of the secretary problem is drawn from D, for any algorithm, the probability of choosing the maximum element from the random instance is at most $\frac{1}{e} + \varepsilon$, where the probability is over the randomness of the instance and the algorithm.*

5.2 Embedding single secretary in simple graphs

This subsection presents the proof of Theorem 3. We first describe some preliminaries which are used in our analysis.

Preliminaries

It is straightforward to reduce the single secretary problem to the graphic matroid secretary problem when the graph allows for multiple edges. In that case, given the single secretary problem with n adversarial weights, we create a multi-graph with just two vertices u and v

and a set of n parallel edges between u and v, with each edge assigned exactly one of these n adversarial weights. Note, that this multi-graph has girth 2. We will need to work harder to reduce the single secretary problem to the graphic matroid problem on simple graphs, where we will even assume that the simple graph has an arbitrarily large girth.

For said reduction, we define below three different measures of performance of a secretarial algorithm. Given an algorithm $a \in \mathcal{A}$ and an instance I of the matroid secretary problem, we abuse the notation from the previous section and denote by $a(I)$ the set of elements of the matroid output by a on instance I. Let also $opt(I)$ be the optimal solution to the problem on the instance I. Given any subset X of the matroid's elements, we denote by $w(X)$ the sum of the weights of all elements in X.

Let Π be a matroid secretary problem on a matroid which has $n \in \mathbb{N}$ elements. Given a finite set of numbers (weights) $W \subset \mathbb{R}_+$, let D be a probability distribution over the set W^n. We will later on instantiate Π with the single secretary problem and with the graphic matroid secretary problem.

Let $a \in \mathcal{A}$ be a (randomized) algorithm for problem Π. Given an instance I of Π, let $S_a(I)$ be a random variable such that $S_a(I) = 1$ if a outputs the optimal solution on instance I, that is, when $a(I) = opt(I)$; and $S_a(I) = 0$ if $a(I) \neq opt(I)$. Then we have that $\mathbb{E}[S_a(I)] = \Pr[a(I) = opt(I)]$; note that the probability is over the internal randomness of algorithm a and over the randomness in the arrival order of the matroid elements.

For the convenience of this subsection, it will be useful to define an algorithm a as having a performance guarantee $\alpha \leq 1$ of

- type 1 if $\mathbb{E}_{I\sim D}[\mathbb{E}[S_a(I)]] \geq \alpha$.
  That is, the algorithm chooses the optimum with probability at least α in expectation over instances.
- type 2 if $\mathbb{E}_{I\sim D}[w(a(I))/w(opt(I))] \geq \alpha$.
  That is, the expected ratio of the algorithm's weight to the optimum's weight is at least α, where the expectation is over instances, algorithm randomness, and element order.
- type 3 if $\mathbb{E}_{I\sim D}[w(a(I))] \geq \mathbb{E}_{I\sim D}[w(opt(I))] \cdot \alpha$.
  That is, the algorithm's expected weight is at least an α fraction of the expected optimum weight. The first expectation includes all randomness; the second expectation is over instances only.

We next reformulate Theorem 16 in terms of the type 1 performance guarantee.

▶ **Theorem 17.** *For any $\epsilon > 0$, $\rho > 1$, there exists a finite distribution D of instances of weighted single secretary such that no algorithm's type 1 performance guarantee is $\geq \frac{1}{e} + \epsilon$. For any instance I of single secretary in the support of D, any two weights in I differ by at least a factor of ρ.*

We now extend the result of Theorem 17 to the type 2 performance guarantee. This proof will make use of the guarantee depending on ρ provided by Theorem 17 to bound the weight attained by an algorithm when it fails to select the maximum.

▶ **Lemma 18.** *For any $\epsilon > 0$, there exists a finite distribution D of instances of weighted single secretary such that no algorithm's type 2 performance guarantee is $\geq \frac{1}{e} + \epsilon$.*

The following lemma then shows an equivalence in hardness between the type 2 and type 3 performance guarantees. The idea of its proof is to reweight the distribution D by making the probability of each instance being chosen inversely proportional to its optimum weight.

▶ **Lemma 19.** *For any matroid secretary problem Π, there exists a bijection B between finite distributions over instances of Π such that if $D' = B(D)$, then there exists an algorithm with a type 2 performance guarantee of r on D iff there exists one with a type 3 performance guarantee of r on D'.*

We now move to the portion of the proof that extends the hardness of the single secretary problem to the graph setting. As part of this reduction, we first show the following lemma, demonstrating the existence of bipartite graphs of high girth with the key property that one part is both much larger than the other and has high degree. This construction proceeds by applying the probabilistic method to derive a graph with our desired properties by removing edges from a high-girth Ramanujan graph.

▶ **Lemma 20.** *For any $d, g, t \in \mathbb{N}$ with $d \geq 2$, $t \geq 4$, there exist $m, n \in \mathbb{N}$ such that $\frac{n}{m} \geq t$ and there exists a graph G' of girth at least g, whose vertices can be partitioned into sets A and B, such that $|A| = m$, $|B| = n$, all edges are between A and B, and each vertex in B has degree at least d.*

We now apply the graph provided by Lemma 20 to extend the hardness of a distribution of instances of the single secretary problem to hardness of matroid secretary on a high-girth graph. The proof, which is quite technical, is provided in the full version of the paper.

▶ **Lemma 21.** *If there exists a finite distribution D of instances of weighted single secretary such that no algorithm has a type 3 performance guarantee $\geq r$ on D, then for any $\epsilon > 0, g \in \mathbb{N}$ there exists a graph G with girth $\geq g$ and a finite distribution D_G of instances of weighted graph secretary on G such that no algorithm has a type 3 performance guarantee $\geq r + \epsilon$ on D'.*

Combining the above results, we obtain Theorem 3; see the full version of the paper for more details.

References

1. Dorna Abdolazimi, Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. Matroid partition property and the secretary problem. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 2:1–2:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ITCS.2023.2.
2. Melika Abolhassani, Soheil Ehsani, Hossein Esfandiari, MohammadTaghi HajiAghayi, Robert Kleinberg, and Brendan Lucier. Beating 1-1/e for ordered prophets. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 61–71. ACM, 2017. doi:10.1145/3055399.3055479.
3. Saeed Alaei, MohammadTaghi Hajiaghayi, and Vahid Liaghat. The online stochastic generalized assignment problem. In *Proceedings of the 16th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 11–25, 2013. doi:10.1007/978-3-642-40328-6_2.
4. Noga Alon, Nick Gravin, Tristan Pollner, Aviad Rubinstein, Hongao Wang, S Matthew Weinberg, and Qianfan Zhang. A bicriterion concentration inequality and prophet inequalities for k-fold matroid unions. *arXiv preprint arXiv:2411.11741*, 2024. doi:10.48550/arXiv.2411.11741.
5. Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Prophet secretary: Surpassing the 1-1/e barrier. In *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, pages 303–318, 2018. doi:10.1145/3219166.3219182.
6. Moshe Babaioff, Michael Dinitz, Anupam Gupta, Nicole Immorlica, and Kunal Talwar. Secretary problems: weights and discounts. In *Proceedings of the twentieth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1245–1254. SIAM, 2009. doi:10.1137/1.9781611973068.135.
7. Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. Matroid secretary problems. *J. ACM*, 65(6):35:1–35:26, 2018. doi:10.1145/3212512.

8 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 434–443. Society for Industrial and Applied Mathematics, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283429.

9 Maryam Bahrani, Hedyeh Beyhaghi, Sahil Singla, and S. Matthew Weinberg. Formal barriers to simple algorithms for the matroid secretary problem. In Michal Feldman, Hu Fu, and Inbal Talgam-Cohen, editors, *Web and Internet Economics - 17th International Conference, WINE 2021, Potsdam, Germany, December 14-17, 2021, Proceedings*, volume 13112 of *Lecture Notes in Computer Science*, pages 280–298. Springer, 2021. doi:10.1007/978-3-030-94676-0_16.

10 Mohammadhossein Bateni, Mohammadtaghi Hajiaghayi, and Morteza Zadimoghaddam. Submodular secretary problem and extensions. *ACM Trans. Algorithms*, 9(4):Art. 32, 23, 2013.

11 Kristóf Bérczi, Vasilis Livanos, José Soto, and Victor Verdugo. Matroid secretary via labeling schemes. *arXiv preprint arXiv:2411.12069*, 2024. doi:10.48550/arXiv.2411.12069.

12 Shuchi Chawla, Jason D. Hartline, David L. Malec, and Balasubramanian Sivan. Multi-parameter mechanism design and sequential posted pricing. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 311–320, 2010. doi:10.1145/1806689.1806733.

13 José Correa, Paul Dütting, Felix A. Fischer, and Kevin Schewior. Prophet inequalities for I.I.D. random variables from an unknown distribution. In Anna R. Karlin, Nicole Immorlica, and Ramesh Johari, editors, *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019*, pages 3–17. ACM, 2019. doi:10.1145/3328526.3329627.

14 José Correa, Patricio Foncea, Ruben Hoeksma, Tim Oosterwijk, and Tjark Vredeveld. Posted price mechanisms for a random stream of customers. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 169–186. ACM, 2017. doi:10.1145/3033274.3085137.

15 José R. Correa, Paul Dütting, Felix A. Fischer, and Kevin Schewior. Prophet inequalities for independent random variables from an unknown distribution. *CoRR*, abs/1811.06114v2, 2021. URL: http://arxiv.org/abs/1811.06114v2.

16 José R. Correa, Raimundo Saona, and Bruno Ziliotto. Prophet secretary through blind strategies. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1946–1961. SIAM, 2019. doi:10.1137/1.9781611975482.118.

17 José R. Correa, Raimundo Saona, and Bruno Ziliotto. Prophet secretary through blind strategies. *Math. Program.*, 190(1):483–521, 2021. doi:10.1007/S10107-020-01544-8.

18 Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Stochastic k-server: How should uber work? In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 126:1–126:14, 2017. doi:10.4230/LIPICS.ICALP.2017.126.

19 Nedialko B Dimitrov and C Greg Plaxton. Competitive weighted matching in transversal matroids. In *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I 35*, pages 397–408. Springer, 2008. doi:10.1007/978-3-540-70575-8_33.

20 Michael Dinitz and Guy Kortsarz. Matroid secretary for regular and decomposable matroids. *SIAM Journal on Computing*, 43(5):1807–1830, 2014. doi:10.1137/13094030X.

21 Paul Dütting and Robert Kleinberg. Polymatroid prophet inequalities. In *Algorithms-ESA 2015*, pages 437–449. Springer, 2015. doi:10.1007/978-3-662-48350-3_37.

22 Hossein Esfandiari, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Morteza Monemizadeh. Prophet secretary. *SIAM J. Discret. Math.*, 31(3):1685–1701, 2017. doi:10.1137/15M1029394.

23 Tomer Ezra, Michal Feldman, Nick Gravin, and Zhihao Gavin Tang. General graphs are easier than bipartite graphs: Tight bounds for secretary matching. In David M. Pennock, Ilya Segal, and Sven Seuken, editors, *EC '22: The 23rd ACM Conference on Economics and Computation, Boulder, CO, USA, July 11 - 15, 2022*, pages 1148–1177. ACM, 2022. doi:10.1145/3490486.3538290.

24 Michal Feldman, Nick Gravin, and Brendan Lucier. Combinatorial auctions via posted prices. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 123–135. SIAM, 2015. doi:10.1137/1.9781611973730.10.

25 Moran Feldman and Rico Zenklusen. The submodular secretary problem goes linear. *SIAM J. Comput.*, 47(2):330–366, 2018. doi:10.1137/16M1105220.

26 Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *SODA*, pages 942–951. Society for Industrial and Applied Mathematics, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347185.

27 Oliver Göbel, Martin Hoefer, Thomas Kesselheim, Thomas Schleiden, and Berthold Vöcking. Online independent set beyond the worst-case: Secretaries, prophets, and periods. In *International Colloquium on Automata, Languages, and Programming*, pages 508–519. Springer, 2014. doi:10.1007/978-3-662-43951-7_43.

28 Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *Proceedings of the 6th International Workshop on Internet and Network Economics (WINE'10)*, 2010.

29 Mohammad Taghi Hajiaghayi, Robert Kleinberg, and Tuomas Sandholm. Automated online mechanism design and prophet inequalities. In *AAAI*, volume 7, pages 58–65, 2007. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-009.php.

30 Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, and David C. Parkes. Adaptive limited-supply online auctions. In *Proceedings 5th ACM Conference on Electronic Commerce (EC-2004), New York, NY, USA, May 17-20, 2004*, pages 71–80. ACM, 2004. doi:10.1145/988772.988784.

31 Zhiyi Huang, Zahra Parsaeian, and Zixuan Zhu. Laminar matroid secretary: Greedy strikes back. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 73:1–73:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.73.

32 Sungjin Im and Yajun Wang. Secretary problems: Laminar matroid and interval scheduling. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1265–1274. SIAM, 2011. doi:10.1137/1.9781611973082.96.

33 Patrick Jaillet, José A Soto, and Rico Zenklusen. Advances on matroid secretary problems: Free order model and laminar case. In *Integer Programming and Combinatorial Optimization: 16th International Conference, IPCO 2013, Valparaíso, Chile, March 18-20, 2013. Proceedings 16*, pages 254–265. Springer, 2013. doi:10.1007/978-3-642-36694-9_22.

34 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2013. doi:10.1007/978-3-642-40450-4_50.

35 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2013. doi:10.1007/978-3-642-40450-4_50.

36 Robert Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 630–631. Society for Industrial and Applied Mathematics, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070519.

37 Robert Kleinberg and S. Matthew Weinberg. Matroid prophet inequalities. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 123–136, 2012. doi:10.1145/2213977.2213991.

38 Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 508–520. Springer, 2009. doi:10.1007/978-3-642-02930-1_42.

39 Ulrich Krengel and Louis Sucheston. Semiamarts and finite values. *Bull. Am. Math. Soc*, 1977.

40 Ulrich Krengel and Louis Sucheston. On semiamarts, amarts, and processes with finite value. *Advances in Prob*, 4:197–266, 1978.

41 Tengyu Ma, Bo Tang, and Yajun Wang. The simulated greedy algorithm for several submodular matroid secretary problems. *Theory of Computing Systems*, 58:681–706, 2016. doi:10.1007/S00224-015-9642-4.

42 Adam Meyerson. Online facility location. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 426–431. IEEE, 2001. doi:10.1109/SFCS.2001.959917.

43 Bo Peng and Zhihao Gavin Tang. Order selection prophet inequality: From threshold optimization to arrival time design. In *Proceedings of the 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2022.

44 Aviad Rubinstein. Beyond matroids: Secretary problem and prophet inequality with general constraints. *arXiv preprint arXiv:1604.00357*, 2016. arXiv:1604.00357.

45 Aviad Rubinstein and Sahil Singla. Combinatorial prophet inequalities. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1671–1687, 2017. doi:10.1137/1.9781611974782.110.

46 José A. Soto, Abner Turkieltaub, and Victor Verdugo. Strong algorithms for the ordinal matroid secretary problem. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 715–734. SIAM, 2018. doi:10.1137/1.9781611975031.47.

47 Ola Svensson. A simple o(loglog(rank))-competitive algorithm for the matroid secretary problem, 2016. Presentation at Microsoft Research available on Youtube; timestamp 35:42. URL: https://www.youtube.com/watch?v=hHBGKbq79yk&t=1776s.

When Is String Reconstruction Using de Bruijn Graphs Hard?

Ben Bals
CWI, Amsterdam, The Netherlands
Vrije Universiteit Amsterdam, The Netherlands

Sebastiaan van Krieken
CWI, Amsterdam, The Netherlands
TU Delft, The Netherlands

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit Amsterdam, The Netherlands

Leen Stougie
CWI, Amsterdam, The Netherlands
Vrije Universiteit Amsterdam, The Netherlands

Hilde Verbeek
CWI, Amsterdam, The Netherlands

Abstract

The reduction of the fragment assembly problem to (variations of) the classical Eulerian trail problem [Pevzner et al., PNAS 2001] has led to remarkable progress in genome assembly. This reduction employs the notion of *de Bruijn graph* $G = (V, E)$ of order k over an alphabet Σ. A single Eulerian trail in G represents a *candidate* genome reconstruction. Bernardini et al. have also introduced the complementary idea in data privacy [ALENEX 2020] based on z-anonymity. Let S be a private string that we would like to release, preventing, however, its full reconstruction. For a privacy threshold $z > 0$, we compute the largest k for which there exist at least z Eulerian trails in the order-k de Bruijn graph of S, and release a string S' obtained via a random Eulerian trail.

The pressing question is: How hard is it to reconstruct a best string from a de Bruijn graph given a function that models domain knowledge? Such a function maps every length-k string to an *interval of positions* where it may occur in the reconstructed string. By the above reduction to de Bruijn graphs, the latter function translates into a function c mapping every edge to an interval where it may occur in an Eulerian trail. This gives rise to the following basic problem on graphs:

Given an instance (G, c), can we efficiently compute an Eulerian trail respecting c?

Hannenhalli et al. [CABIOS 1996] formalized this problem and showed that it is NP-complete. Ben-Dor et al. [J. Comput. Biol. 2002] showed that it is NP-complete, even on de Bruin graphs with $|\Sigma| = 4$. In this work, we settle the lower-bound side of this problem by showing that finding a c-respecting Eulerian trail in de Bruijn graphs over alphabets of size 2 is NP-complete.

We then shift our focus to *parametrization* aiming to capture the quality of our domain knowledge in the complexity. Ben-Dor et al. developed an algorithm to solve the problem on de Bruijn graphs in $\mathcal{O}(m \cdot w^{1.5}4^w)$ time, where $m = |E|$ and w is the *maximum interval length* over all edges in E. Bumpus and Meeks [Algorithmica 2023] later rediscovered the same algorithm on temporal graphs, which highlights the relevance of this problem in other contexts. Our central contribution is showing how combinatorial insights lead to *exponential-time* improvements over the state-of-the-art algorithm. In particular, for the important class of de Bruijn graphs, we develop an algorithm parametrized by $w(\log w + 1)/(k-1)$: for a de Bruijn graph of order k, it runs in $\mathcal{O}(mw \cdot 2^{\frac{w(\log w+1)}{k-1}})$ time. Our result improves on the state of the art by roughly an exponent of $(\log w + 1)/(k-1)$. The existing algorithms have a natural interpretation for string reconstruction: when for each length-k string, we know a small range of positions it must lie in, string reconstruction can be solved in linear time. Our improved algorithm shows that *it is enough when the range of positions is small relative to k.*



© Ben Bals, Sebastiaan van Krieken, Solon P. Pissis, Leen Stougie, and Hilde Verbeek;
licensed under Creative Commons License CC-BY 4.0

33rd Annual European Symposium on Algorithms (ESA 2025).
Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman; Article No. 53; pp. 53:1–53:16



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

We then generalize both the existing and our novel FPT algorithm by allowing the cost at every position of an interval to vary. In this optimization version, our hardness result translates into inapproximability and the FPT algorithms work with a slight extension. Surprisingly, even in this more general setting, we extend the FPT algorithms to count and enumerate the min-cost Eulerian trails. The counting result has direct applications in the data privacy framework of Bernardini et al.

2012 ACM Subject Classification Theory of computation → Pattern matching; Theory of computation → Graph algorithms analysis

Keywords and phrases string algorithm, graph algorithm, de Bruijn graph, Eulerian trail

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.53

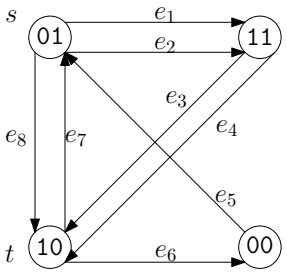
Related Version *Full Version*: https://arxiv.org/abs/2508.03433

Funding *Solon P. Pissis*: Supported in part by the PANGAIA and ALPACA projects that have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 872539 and 956229, respectively.
Leen Stougie: Supported in part by the Netherlands Organisation for Scientific Research (NWO) through project OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)" and Gravitation-project NETWORKS-024.002.003.
Hilde Verbeek: Supported by a Constance van Eeden Fellowship.

1 Introduction

One of the most important algorithmic tasks in bioinformatics is that of *genome assembly* (or *fragment assembly*) [30, 8, 23, 9]: the process of taking a large number of short DNA fragments and putting them back together to create a representation of the original chromosomes from which the DNA originated. The textbook reduction of the fragment assembly problem to (variations of) the classical Eulerian trail problem [26] has led to remarkable progress in the past three decades. This reduction is based on the notion of *de Bruijn graph* (dBG, in short).

Let $S = S[1] \dots S[|S|] = S[1 \,..\, |S|]$ be a *string* of length $|S|$ over an *alphabet* Σ. We fix a collection $\mathcal{S}$ of strings over Σ and define the *order-k de Bruijn graph* of $\mathcal{S}$ as a directed graph, denoted by $G_{\mathcal{S},k} = (V, E)$, where V is the set of length-$(k-1)$ substrings of the strings in $\mathcal{S}$ and E has an edge (u, v) if and only if $u[1] \cdot v = u \cdot v[k-1]$ and $u[1] \cdot v$ occurs in some string S of $\mathcal{S}$. In applications, we often consider a de Bruijn *multigraph* where the multiplicity of an edge is exactly the total number of these occurrences in the strings in $\mathcal{S}$. Then, a single Eulerian trail in $G_{\mathcal{S},k}$ represents a *candidate* string reconstruction [26]; see Figure 1.



Eulerian trails	String reconstructions
$e_1e_3e_6e_5e_2e_4e_7e_8$	0110011010
$e_1e_3e_7e_2e_4e_6e_5e_8$	0110110010
$e_1e_3e_7e_8e_6e_5e_2e_4$	0110100110
$e_8e_6e_5e_1e_3e_7e_2e_4$	0100110110
$e_8e_7e_1e_3e_6e_5e_2e_4$	0101100110
$e_1e_3e_6e_5e_8e_7e_2e_4$	0110010110

Figure 1 The de Bruijn multigraph $G_{\mathcal{S},k} = (V, E)$ (left), the set of node-distinct Eulerian trails from s to t (middle), and the corresponding set of string reconstructions (right) for the string collection $\mathcal{S} = \mathtt{001}, \mathtt{010}, \mathtt{011}, \mathtt{011}, \mathtt{100}, \mathtt{101}, \mathtt{110}, \mathtt{110}$, over the alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$, and $k = 3$.

Bernardini et al. have introduced the complementary idea in data privacy [4] based on the z-*anonymity* privacy property [27, 29]. Let S be a *private* string that we would like to release for data analysis, preventing, however, its full reconstruction. For a privacy threshold $z > 0$, we compute the largest k for which there exist at least z node-distinct Eulerian trails in the order-k dBG of $\mathcal{S} = \{S\}$, and release a string S' obtained via a random Eulerian trail. In Figure 1, we have 6 node-distinct Eulerian trails (and thus 6 distinct strings) corresponding to $\mathcal{S}$. Under the z-anonymity assumption, one cannot know which of the 6 strings is S unless they can rely on some additional information about S, such as on *domain knowledge*.

The pressing question arising from these applications is thus: How hard is it to reconstruct a best string from $G_{\mathcal{S},k}$ given a function modeling domain knowledge? This function maps every length-k string to an *interval of positions* where it may occur in the reconstructed string. By the above reduction, this translates into a function c mapping every edge to an interval where it may occur in an Eulerian trail, raising the following basic graph problem:

Given an instance (G, c), can we efficiently compute an Eulerian trail respecting c?

Although Hannenhalli et al. [16] formalized this basic problem in the context of dBGs, it has applications in temporal graphs [7, 22, 24] and other networks [11, 31, 20]. We denote it here by diET (see Section 2 for a formal definition). Hannenhalli et al. showed that diET is NP-complete when each node of G has in- and out-degree at most two. Even if their motivating application was fragment assembly and thus based on dBGs, their result was shown on general directed graphs. Ben-Dor et al. [3] then showed that diET is NP-complete, even on dBGs with $|\Sigma| = 4$. We settle the negative landscape by showing the following result:

▶ **Theorem 3 (*).** *The* diET *problem is* NP*-complete, even on de Bruijn graphs with* $|\Sigma| = 2$.

Beyond theoretically interesting, Theorem 3 shows that *in general* it is indeed hard to reconstruct a private (binary) string from a given dBG, thus providing theoretical justification for the privacy framework introduced by Bernardini et al. [4]. Given these negative results, we shift our focus to *parametrization* aiming to capture the quality of our domain knowledge in the complexity. This leads to algorithms that are efficient if the intervals are small [16]: every interval length is bounded by a natural number w, which we term the *interval width*. We denote this parametrized version of diET by diET$[w]$. Hannenhalli et al. [16] showed an algorithm for diET$[w]$ working in $\mathcal{O}(m^{2w+\log(2w)})$ time for any directed graph $G = (V, E)$, with $m = |E|$ and $w = \mathcal{O}(1)$. Ben-Dor et al. [3] developed an algorithm to solve the problem on dBGs in $\mathcal{O}(m \cdot w^{1.5}4^w)$ time, for any w. Bumpus and Meeks [7] later rediscovered the same algorithm on temporal graphs, which highlights the relevance of the diET problem in other contexts. We may thus summarize the state of the art in the following statement.

▶ **Theorem 10** (Theorem 11 of [3], Theorem 7 of [7])**.** *There is an $\mathcal{O}(m \cdot w^{1.5}4^w)$-time algorithm solving the* diET$[w]$ *problem. Therefore,* diET$[w]$ *is in* FPT.

We observe that the state-of-the-art algorithm does not exploit the dBG structure. Our central contribution is developing an algorithm for dBGs parametrized by $w(\log w+1)/(k-1)$:

▶ **Theorem 11.** *Let G be a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$. There is an $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$-time algorithm solving the* diET$[w]$ *problem, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Theorem 11 improves on the state of the art by roughly an exponent of $(\log w+1)/(k-1)$. The existing algorithms [16, 3, 7] all have a natural interpretation for string reconstruction: when for each length-k substring (k-*mer*), we know a small range of positions it must lie in, string reconstruction can be solved in linear time. Theorem 11 shows that *it is enough when*

the range of positions is small relative to the order k of the dBG. In particular, we show that for dBGs it is sufficient if $w \log w/(k-1)$ is relatively small, which significantly extends the practical applicability of our technique. For instance, in bioinformatics, it is standard to use $k = 31$ [21] and then we have $|\Sigma| = 4$ (the size of the DNA alphabet), which implies an exponential speedup by $\sqrt[30]{\cdot}$. Our approach of improving the FPT algorithms for diET based on combinatorial insights into the structure of the instances suggests further research into closely-related problems; e.g., the *Hierarchical Chinese Postman* (HCP) problem [15, 19, 11, 1] and the related *Time-Constrained Chinese Postman* (TCCP) problem [31, 28].

We then generalize the above results by allowing the cost at every position of an interval to vary. We denote this problem here by dicET (see Section 2 for a formal definition). In this setting, our hardness result (Theorem 3) translates into inapproximability.

▶ **Corollary 7 (*).** *If* P $\neq$ NP *there is no constant-factor polynomial-time approximation algorithm for the* dicET *problem, even on de Bruijn graphs with interval cost functions.*

We show that the FPT algorithms underlying both Theorems 10 and 11 also work in the optimization version with an interval cost function, which we denote by dicET$[w]$.

▶ **Corollary 20 (*).** *Given a* dicET$[w]$ *instance, there is an $\mathcal{O}(m \cdot w^{1.5} 4^w)$-time algorithm finding a min-cost Eulerian trail in G. On a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$, we can solve this problem in $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Surprisingly, even in this more general setting, we show how to extend our FPT techniques to count the number of min-cost Eulerian trails. We show the following result.

▶ **Theorem 21.** *Given a* dicET$[w]$ *instance, we can count the number of min-cost Eulerian trails in $\mathcal{O}(m \cdot w^{1.5} 4^w)$ time. On a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$, we can solve this problem in $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

We can also enumerate these trails in the same time as for counting (Theorem 21) plus time that is linear in the size of the output. Notably, all of our algorithmic results translate from directed to undirected graphs with the same complexities. It is easy to verify that none of the proofs depend on the directedness of the graph. For simplicity, we focus our discussion on directed graphs. Particularly our result for counting Eulerian trails in undirected graphs is surprising given that the problem is #P-complete in the standard setting [6]. We also show that most of our algorithmic results generalize to multigraphs. Finally, we show that the undirected version of dicET, which we denote by uicET, is also NP-complete.

Overview of Hardness Techniques. We consider dBGs over alphabets of size two. Unlike [16, 3], this highly structured setting requires intricate techniques to obtain an (elementary) reduction from the directed Hamiltonian path problem [18]. For our instances, we harness the fact that the shortest path between any two nodes in a complete dBG is unique and has a meaningful string interpretation. The structure of the reduction makes it directly translatable to the optimization setting implying inapproximability. Similarly, we reduce the undirected Hamiltonian path problem [18] to finding a min-cost c-respecting Eulerian trail in an undirected graph. This reduction requires a different approach than the directed case to ensure that critical parts of the graph can only be traversed in the desired order. We assign different costs at even and odd time steps to certain edges to achieve that goal.

Overview of Algorithmic Techniques. We apply tools from parametrized algorithms [10] to solve dicET$[w]$ efficiently. Our techniques can be viewed as a careful combination of searching in a well-bounded state space and dynamic programming. We further combine

approaches from graph and string algorithms to enhance the vanilla version of these tools with combinatorial insights into dBGs and obtain exponential-time improvements. The robustness of these tools allows us to generalize our algorithm for the decision version of dicET[w] to both the optimization and the counting versions. The counting result in particular relies on the fact that the state space compactly captures all possible Eulerian trails, by excluding impossible trails at the construction level, and representing the possible ones efficiently.

Other Related Work. Our work is closely related to exploring *temporal graphs* [24, 13, 14, 12, 2, 7, 22]: graphs where every edge is available at an arbitrary subset of the time steps. Most relevant to our work are perhaps the works by Bumpus and Meeks [7] and by Marino and Silva [22]. The former proved that deciding whether a temporal graph has an Eulerian trail is NP-complete even if each edge appears at most r times, for every fixed $r \geq 3$. They also apply similar parametrized tools to interval-membership-width, a temporal graph parameter related to our interval width, but they do not consider directed graphs, costs, or counting. Marino and Silva [22] showed, for undirected graphs, that, if the edges of a temporal graph (G, λ) with lifetime τ are always available during the lifetime, then deciding whether (G, λ) has an Eulerian trail is NP-complete even if $\tau = 2$. They also showed that this problem is in XP when parametrized by $\tau + \mathsf{tw}(G)$, where $\mathsf{tw}(G)$ is the treewidth of G.

Our work is also closely related to the HCP problem [15, 19, 11, 1] (and the related TCCP problem [31, 28]). In HCP, we are given an edge-weighted undirected graph $G = (V, E)$, a partition $\mathcal{P}$ of E into k classes, and a partial order $\prec$ on $\mathcal{P}$, and we are asked to find a least-weight closed walk traversing each edge in E at least once such that each edge e in a class E' is traversed only after all edges in all classes $E'' \prec E'$ are traversed. The main differences to our problem are two: (1) In HCP, there is a partition on the edges and then a partial order between the classes, whereas we have an arbitrary interval per edge; and (2) in HCP, one must traverse each edge at least once, whereas we have to do this exactly once.

Paper Organization. We begin by formalizing diET and dicET in Section 2. We present our hardness and inapproximability results in Section 3. We complement these negative results with positive algorithmic results for diET[w] in Section 4. In Section 5, we extend our algorithms to counting Eulerian trails and discuss the relationship to the data privacy applications. In the full version, we discuss how most of our algorithms extend to multigraphs. *The formal statements, with proofs or details deferred to the full version, are marked with* ⋆.

2 Preliminaries

For $a, b \in \mathbb{N}$, let $[a, b] \coloneqq \{x \in \mathbb{N} \mid a \leq x \leq b\}$, $[a, b) \coloneqq \{x \in \mathbb{N} \mid a \leq x < b\}$, and $[b] \coloneqq [1, b]$. Further let $\mathcal{I}_b \coloneqq \{[i, j] \mid 1 \leq i \leq j \leq b\} \cup \{\{\}\}$ be the set of intervals on $[b]$.

We imagine a trail as "walking" through a graph and thus refer to the ranks of the edges as happening at certain time steps (e.g., the second edge is associated with the time step 2). This vocabulary borrowed from temporal graphs makes the discussion more intuitive.

▶ **Definition 1.** *Let $G = (V, E)$ be a directed or undirected graph with $m = |E|$. We call a function $c\colon E \to \mathcal{I}_m$ an* interval function *for G. We extend this notion to an* interval cost function *where, for each $e \in E$, each time step from the interval $c(e)$ is associated with a cost from $\mathbb{Z}$. As an abuse of notation, we write $c(e, t)$ for this cost. If $t \notin c(e)$, we set $c(e, t) \coloneqq \infty$.*

When the relevant graph $G = (V, E)$ is clear from context, we set $n \coloneqq |V|$ and $m \coloneqq |E|$. We next formalize the main problems in scope on directed graphs.

▶ **Problem.** Eulerian Trails in Digraphs with Interval functions (diET)

Given: Directed graph $G = (V, E)$, interval function $c\colon E \to \mathcal{I}_m$

Decide: Is there an Eulerian trail $P = e_1 \dots e_m$ in G such that for all $t \in [m]$, $t \in c(e_t)$?

Similarly, we consider the version with interval *cost* functions.

▶ **Problem.** Eulerian Trails in Digraphs with Interval Cost functions (dicET)

Given: Directed graph $G = (V, E)$, interval cost function $c\colon E \times [m] \to \mathbb{Z} \cup \{\infty\}$, $c_{\text{budget}} \in \mathbb{N}$

Decide: Is there an Eulerian trail $P = e_1 \dots e_m$ in G such that $\sum_{t=1}^{m} c(e_t, t) \leq c_{\text{budget}}$?

In addition, we consider the uicET problem as the natural *undirected* version of dicET.

Note that in all cases, we allow setting every interval to the whole $[m]$. In the case without costs, this yields the classical Eulerian trail problem [17]. In the case with costs, this yields a min-cost Eulerian trail problem without any interval restrictions, which is a natural special case we will call out when considering it to be particularly relevant. See Figure 2 for an example of a graph with an interval function and the corresponding edge availabilities.

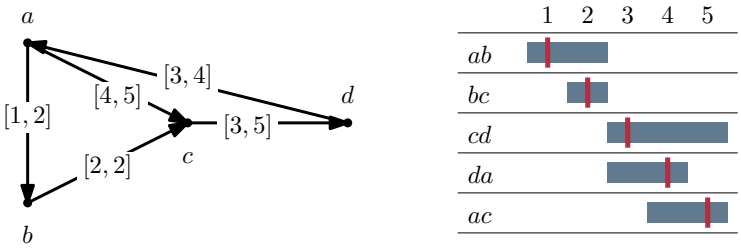


Figure 2 On the left is the input graph G: every edge is labeled with the time steps at which it is available. On the right is a table illustrating the interval every edge is available: *abcdac* is an Eulerian trail; the edge usages corresponding to this trail are indicated with red vertical lines.

The following definition makes our arguments based on an interval function more legible.

▶ **Definition 2.** *We call an edge $e \in E$* available *at $t \in [m]$, if $t \in c(e)$. For a time step $t \in [m]$, we write $E(t)$ to denote the set of edges available at t. Given an interval $T \subseteq [m]$, we similarly write $E(T)$ for the set of edges available at at least one time step from T.*

3 NP-hardness and Inapproximability

In this section, we consider both the directed and undirected case from the hardness perspective. Since in the directed case, dBGs are a particularly relevant application, we give strong results, even when restricting the instances to this graph class. We are the first to show the hardness for alphabets of all sizes (except $|\Sigma| = 1$, where the problem is trivial). In particular, we settle the question for the important case of binary alphabets. In the undirected case, we consider general graphs, which we believe is nonetheless insightful regarding what makes this problem class hard.

We first settle the computational complexity of diET.

▶ **Theorem 3 (*).** *The* diET *problem is* NP*-complete, even on de Bruijn graphs with $|\Sigma| = 2$.*

We reduce from the directed Hamiltonian path problem, which is NP-complete [18]. Let $G = (V, E)$ be the directed graph in which we want to find a Hamiltonian path.

Construction of the dBG. We consider any two-letter alphabet; for example, here we will consider the subset $\Sigma \coloneqq \{\mathtt{A}, \mathtt{T}\}$ of the DNA alphabet. We set $\ell \coloneqq \lceil \log_2(|V|) \rceil$ and construct a diET instance $(G' = (V', E'), c)$ as the complete dBG on the set of nodes each labeled by a string of length $k - 1 = 4\ell + 10$, as well as an interval function c that we will describe later. Note that, by the choice of ℓ and Σ, G' has $|\Sigma|^{k-1} = 2^{4\ell+10} = \mathcal{O}(|V|^4)$ nodes. We will pay special attention to some of the nodes in G', which we assign an interpretation in terms of G.

For each $v \in V$, let $\mathrm{id}_v \in \Sigma^\ell$ be a unique identifier among V. By the choice of ℓ, this numbering is always possible. Similarly, let $\mathrm{id}_{v'} \in \Sigma^{4\ell+10}$ be the string associated with a node $v' \in V'$. With each node $v \in V$, we associate two nodes v'_1 and v'_2 in V' such that:

$$\mathrm{id}_{v'_1} \coloneqq \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \cdot \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \text{ and } \mathrm{id}_{v'_2} \coloneqq \mathtt{A}^{\ell+2} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \cdot \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \cdot \mathtt{A},$$

where $\mathtt{A}^i$ is the string of i $\mathtt{A}$'s. We use the notation $a \xrightarrow{x} b$ to denote a directed edge ab labeled x, where x is the letter used to obtain b from a in the dBG. We extend this notation to $a \overset{X}{\rightsquigarrow} b$ to denote the path from a to b using string X. Observe that by the choice of the underlying strings, we have an edge $v'_1 \xrightarrow{\mathtt{A}} v'_2$ in the dBG. We call this the *inner edge* for v. With each edge $e = vu \in E$, we associate two nodes e'_1 and e'_2 such that:

$$\mathrm{id}_{e'_1} \coloneqq \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \cdot \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_u \cdot \mathtt{T} \text{ and } \mathrm{id}_{e'_2} \coloneqq \mathtt{A}^{\ell+2} \cdot \mathtt{T} \cdot \mathrm{id}_v \cdot \mathtt{T} \cdot \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_u \cdot \mathtt{T} \cdot \mathtt{A}.$$

Again, we have an edge $e'_1 \xrightarrow{\mathtt{A}} e'_2$ in the dBG. We call this the *inner edge* for e. For every other string of length $4\ell + 10$ over Σ a node v' exists in V', but it is not associated with a node $v \in V$ or an edge $e \in E$. See Figure 3 for an illustration of this construction. A crucial property that we will later utilize is that for every pair of nodes $v, u \in V$, such that $vu \in E$, there is a unique shortest path of length $2\ell + 4$ in G' between v'_2 and $(vu)'_1$; this path is associated with the string $\mathtt{A}^{\ell+2} \cdot \mathtt{T} \cdot \mathrm{id}_u \cdot \mathtt{T}$ (see Figure 3). The analogous property holds for $(vu)'_2$ and u'_1: there is a unique shortest path of length $2\ell + 4$ in G' from $(vu)'_2$ to u'_1.

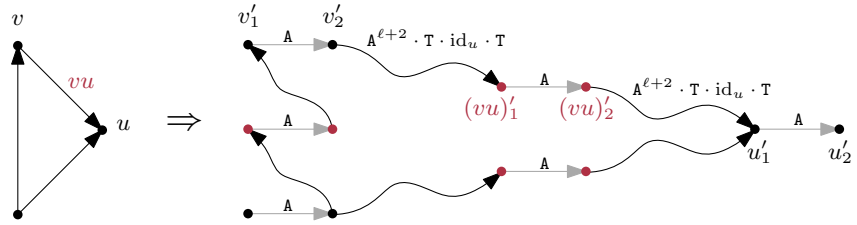


Figure 3 A directed graph G (left) and the corresponding part of graph G' (right). The nodes in G' that correspond to nodes in G are colored black; the nodes in G' that correspond to edges in G are colored red. Inner edges in G' are colored gray and squiggly lines indicate *paths* of length $2\ell + 4$.

It is straightforward to verify that the claimed path (from v'_2 to $(vu)'_1$) exists. To see that this is the shortest possible, notice that $\mathrm{id}_{v'_2}$ contains the substring $\mathtt{A}^{\ell+3}$ starting at position $2\ell + 5$ (i.e., after the first occurrence of $\mathrm{id}_v \cdot \mathtt{T}$). Any one edge on a path to $(vu)'_1$ shifts this substring left by one letter. By the strategic placement of $\mathtt{T}$ around the id parts, the substring $\mathtt{A}^{\ell+3}$ occurs precisely at positions 1 and $2\ell + 6$ in $\mathrm{id}_{(vu)'_1}$. Thus, the closest occurrence of $\mathtt{A}^{\ell+3}$ left of position $2\ell + 6$ in $\mathrm{id}_{(vu)'_1}$ starts at position 1 (i.e., requires a left shift by at least $2\ell + 4$ letters). Therefore, any path from v'_2 to $(vu)'_1$ must contain at least $2\ell + 4$ edges. The argument is analogous for the path from $(vu)'_2$ to u'_1.

Construction of the Interval Function. Let us set $\tau \coloneqq 2n - 1 + 2(n - 1)(2\ell + 4)$. We dub the time steps 1 to τ *early* and all others *late*. The intuition is that the first τ edges on any c-respecting Eulerian trail in G' will correspond to a Hamiltonian path in G. For ease of notation, we set $m' \coloneqq |\Sigma|^k = 2^{4\ell+11}$ to be the number of edges in G'.

For the interval function c, we set $c(v'_1v'_2) \coloneqq [1, \tau]$, for all $v \in V$. For any non-edge $e \notin E$ we set $c(e'_1e'_2) \coloneqq [\tau + 1, m']$. All other edges in G' have the full interval $[m']$. This also holds for all inner edges for the edges in E. Note that with these choices, the inner edges for nodes $v \in V$ are only available early, and for a node pair $v, u \in V$, the edge $(vu)'_1 \to (vu)'_2$ is available early only if $v \to u$ is an edge in E.

▶ **Lemma 4.** *If G has a Hamiltonian path, there is a c-respecting Eulerian trail in (G', c).*

Proof. Let $P = v_1 \dots v_n$ be a Hamiltonian path in G. Then for the Eulerian trail in G', set

$$P'_1 \coloneqq (v_1)'_1 \xrightarrow{\mathtt{A}} (v_1)'_2 \overset{\mathtt{A}^{\ell+2}\cdot\mathtt{T}\cdot\mathrm{id}_{v_2}\cdot\mathtt{T}}{\rightsquigarrow} (v_1v_2)'_1 \xrightarrow{\mathtt{A}} (v_1v_2)'_2 \rightsquigarrow \cdots \rightsquigarrow (v_n)'_1 \xrightarrow{\mathtt{A}} (v_n)'_2.$$

Notice that P'_1 traverses all inner edges for nodes and that P'_1 has length exactly τ. Thus all inner edges for nodes are available by construction at their time steps on P'_1. Similarly, for any $i \in [n-1]$, we have that $v_iv_{i+1} \in E$ since P is a Hamiltonian path in G. Thus, the inner edges for edges are available on P'_1 as well. Finally, all other edges on P'_1 have their $\mathtt{A}^{\ell+3}$ substring in such positions that they do not fall under any of the edge classes whose interval is restricted, thus they are available for the full interval $[m']$. We conclude that P'_1 respects c.

As we want P'_1 to be the prefix of an Eulerian trail, we must ensure that no edge repeats in P'_1. The nodes of types v'_1, v'_2, e'_1, and e'_2 in G', for some $v \in V$ or $e \in E$, each appears at most once on P'_1 by construction, and thus so must their inner edges. Therefore, if there is an edge xy in G', for two nodes x and y, that appears twice on P'_1, it must be on some subpath of type $(v_i)'_2 \overset{\mathtt{A}^{\ell+2}\cdot\mathtt{T}\cdot\mathrm{id}_{v_{i+i}}\cdot\mathtt{T}}{\rightsquigarrow} (v_iv_{i+1})'_1$ (or analogously of the type $(v_iv_{i+1})'_2 \overset{\mathtt{A}^{\ell+2}\cdot\mathtt{T}\cdot\mathrm{id}_{v_{i+1}}\cdot\mathtt{T}}{\rightsquigarrow} (v_{i+1})'_1$) as well as at some other part of P'_1. Let us analyze the string corresponding to node x. As x appears on the path from $(v_i)'_2$ to $(v_iv_{i+1})'_1$, it must have the form $r_1 \cdot \mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_{v_i} \cdot \mathtt{T} \cdot r_2$, where r_1 is a suffix of $\mathtt{A}^{\ell+2} \cdot \mathtt{T} \cdot \mathrm{id}_{v_i} \cdot \mathtt{T}$ and r_2 is a prefix of $\mathtt{A}^{\ell+3} \cdot \mathtt{T} \cdot \mathrm{id}_{v_{i+1}} \cdot \mathtt{T}$. Consider that therefore the string corresponding to x contains the substring $\mathtt{A}^{\ell+3}\cdot\mathtt{T}\cdot\mathrm{id}_{v_i}\cdot\mathtt{T}$ somewhere in the middle. Since, however, our id's are unique, x may only appear on the subpath corresponding to an out-edge of v_i in G and a Hamiltonian path can only include one out-edge per node. Finally, no edge can repeat within the same such subpath as this is a shortest path. The above discussion contradicts our assumption that xy repeats on P'_1. Therefore, we conclude that all edges on P'_1 are unique.

We are left to show that we can complete P'_1 to traverse every edge of G' exactly once. Observe that since G' is a complete dBG, the in- and out-degree of every node is $|\Sigma| = 2$. Therefore, in $G' - P'_1$ (which we define as G' without the edges in P'_1) the in- and out-degree of every node is the same except for $(v_1)'_1$ (where the out-degree is one less than the in-degree) and for $(v_n)'_2$ (where the out-degree is one more than the in-degree). Also notice that $G' - P'_1$ remains weakly connected (as any strongly connected 2-regular directed graph remains weakly connected if a path is removed). By these properties, there is an Eulerian trail P'_2 in $G' - P'_1$. In particular, it must start at $(v_n)'_2$ and end at $(v_1)'_1$. Hence $P' \coloneqq P'_1P'_2$ is an Eulerian trail in G'. As the time steps of all edges from P'_1 remain the same, that prefix of P' still respects c. For the suffix of P' defined by P'_2, observe that all edges are used at time steps later than τ and that none of the edges are inner edges for nodes. Thus, all edges in the P'_2 part of P' are also available under c. We conclude that P' is an Eulerian trail in (G', c). ◀

Similar ideas allow us to prove the other direction, yielding the following lemma.

▶ **Lemma 5 (*).** *If there is a c-respecting Eulerian trail in G', G has a Hamiltonian path.*

Lemma 4 and Lemma 5 allow us to deduce Theorem 3.

Consequences. The construction we used is quite restrictive in that it uses only two letters to construct the dBG and is within the framework of interval functions without costs. This allows us to deduce the following two corollaries, narrowing down the hardness landscape of the general problem even further. Corollary 6 was first shown by Hannenhalli et al. [16].

▶ **Corollary 6** ([16]). *The* diET *problem remains* NP*-hard even on 2-regular graphs.*

Proof. Notice that the construction for Theorem 3 uses only the letters $\{\mathtt{A}, \mathtt{T}\}$ and thus the complete dBG (over that alphabet) is 2-regular. ◀

Our hardness reduction also directly implies that dicET is NP-hard: we set the costs of every edge to 0 within its interval and to 1 outside its interval. We also set $c_{\text{budget}} := 0$. As our proof of Theorem 3 relies only on distinguishing zero and non-zero cost instances, we deduce the following inapproximability result, which is to the best of our knowledge new.

▶ **Corollary 7** (*). *If* P $\neq$ NP *there is no constant-factor polynomial-time approximation algorithm for the* dicET *problem, even on de Bruijn graphs with interval cost functions.*

Interestingly, the problem remains hard and inapproximable, even when restricting dicET to instances where every interval is the full interval $[m]$ (effectively dropping the interval requirement and only considering edge costs). This reduction simply extends, for each edge, its interval and let all costs outside the original interval be $c_{\text{budget}} + 1$.

Although we can find an Eulerian trail in an undirected graph in linear time by Hierholzer's algorithm [17], counting the number of Eulerian trails in an undirected graph is #P-complete [6]. Given this complexity landscape and our results on directed graphs, it is natural to ask about the complexity of deciding if an Eulerian trail of at most some fixed cost exists. We show that this problem is NP-complete and inapproximable in the full version.

4 FPT by Interval Width

After these strong negative results, it is now natural to ask whether there is a restricted setting in which we can efficiently solve diET. Interval functions (see Definition 1) are a restricted class of functions that are natural to the string reconstruction application. In this setting, it is equally natural to ask if more precise domain knowledge leads to more efficient algorithms. Providing tighter intervals for the time step each edge is to be used reduces the combinatorial complexity as there are fewer valid combinations to consider. This observation also translates to string reconstruction, where we might expect that reconstructing a string is easier when the location of every k-mer is narrowed down to a small interval.

▶ **Definition 8.** *Given a* diET *instance* $((V, E), c)$ *with interval function* c*, we define the instance's* interval width w *as the length of the longest interval of any edge* $e \in E$ *under* c*.*

Interestingly, this concept is closely related to the notion of *interval-membership width* (imw, in short) studied by Bumpus and Meeks in the context of temporal graphs [7]. This notion asks how many edges in the graph are relevant to the decision at a specific time step t by looking at the set of edges which have been available at or before t *and* will also be available at or after t. The imw is the largest size of such a set for any t. In Lemma 9, we show that this parameter is related to the length of the longest interval in our setting (Definition 8). A special case of this observation also underlies the FPT algorithm of Ben-Dor et al [3]. This relates the length of time any single edge is available to the total number of edges relevant at a given time. With this observation, we unveil the equivalence of the algorithms by Bumpus and Meeks and by Ben-Dor et al.

For dBGs of order k, we severely speed up these algorithms by roughly an exponent of $(\log w + 1)/(k-1)$. In fact, we show that on dBGs, the problem is not only in FPT by w, but also by $w \log w/k$; namely, we show that it suffices if w is small compared to k to make the problem tractable. We achieve this by extending the techniques in [3, 7] with the properties of dBGs and their underlying string interpretation. By utilizing a significantly more general setting, Bumpus and Meeks [7] have shown that the algorithm by Ben-Dor et al. [3] applies to a significantly broader graph class than dBGs. We observe that this leaves plenty of dBG-specific structure unexploited to speed up the task. Finally, we show how to extend both the existing and our novel approach to find min-cost Eulerian trails.

Considering Definition 8, the definition of the parametrized problem diET[w] is canonical. We first consider simple graphs, deferring the discussion of multigraphs to the full version.

▶ **Problem.** diET[w]

Parameter:	$w \in \mathbb{N}^+$
Given:	Directed graph $G = (V, E)$, interval function $c\colon E \to \mathcal{I}_m$ with interval width w
Decide:	Is there an Eulerian trail $P = e_1 \dots e_m$ such that for all $t \in [m]$, $t \in c(e_t)$?

The following structural insight (Lemma 9) generalizes [3, Lemma 8] and underpins their algorithm and our reduction to the Bumpus and Meeks version of the FPT algorithm: Since any edge is available only at a bounded number of time steps, a similar bound applies to the set of edges available at an individual time step. This insight implies that there are relatively few choices on which edge to select at a specific time step in an Eulerian trail.

▶ **Lemma 9 (*).** *If (G, c, w) is a YES-instance of* diET[w] *and T is an interval in $[m]$, then $|E(T)| \leq |T| + 2w - 2$.*

In particular, Lemma 9 implies that at any specific time step (i.e., an interval of length 1), there are at most $2w - 1$ edges available. Thus, we have that any graph with interval width w has imw at most $2w - 1$. The condition of this lemma is easy to check in time $\mathcal{O}(mw)$ by testing if the number of edges available at each time step is at most $2w - 1$. If this is not the case for any time step, we deduce that (G, c, w) is a NO-instance for diET[w]. We will henceforth assume that the condition has been checked and deduce the following result.

▶ **Theorem 10** (Theorem 11 of [3], Theorem 7 of [7])**.** *There is an $\mathcal{O}(m \cdot w^{1.5} 4^w)$-time algorithm solving the* diET[w] *problem. Therefore,* diET[w] *is in* FPT*.*

Three remarks are in order regarding the version of this theorem by Bumpus and Meeks [7]. First, note that this rendering of the theorem requires a small modification, as we are interested in any Eulerian trail, not necessarily a cycle. This allows us to slash a factor of w. Second, their algorithm is for undirected graphs, but it naturally translates to directed graphs. Third, their version of the algorithm does not exclude a small number of irrelevant states and is thus slightly slower (by another factor of $\sqrt{w}$) than the version by Ben-Dor et al. For the sake of completeness, we provide a proof with these modifications in the full version.

Specializing the parametrized technique of [3, 7] to dBGs allows us to design an algorithm that achieves a significant speedup. Our improvements rely on exploiting the specific structure in the dBG graph class in which we are particularly interested in finding Eulerian trails.

▶ **Theorem 11.** *Let G be a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$. There is an $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$-time algorithm solving the* diET[w] *problem, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Since λ is defined as a minimum, the algorithm underlying Theorem 11 always runs in time $\mathcal{O}(mw \cdot 2^{\frac{w(\log w+1)}{k-1}})$, as shown by the following lemma.

▶ **Lemma 12.** *For any* $w, k \in \mathbb{N}^+$, $(2w-1)^{\frac{w}{k-1}+1} = \mathcal{O}(w \cdot 2^{\frac{w(\log w+1)}{k-1}})$.

Proof. Note that $(2w-1)^{\frac{w}{k-1}+1} = (2w-1) \cdot (2w-1)^{\frac{w}{k-1}}$. We first bound the exponential term. Then multiplying by $(2w-1)$ gives the claimed upper bound. We have that

$$(2w-1)^{\frac{w}{k-1}} = 2^{\frac{w \log(2w-1)}{k-1}} \leq 2^{\frac{w \log(2w)}{k-1}} = 2^{\frac{w(\log w + \log 2)}{k-1}} = 2^{\frac{w(\log w+1)}{k-1}}.$$ ◀

Notice that the exponential part is higher than for the $\mathcal{O}(mw^{1.5} \cdot 4^w)$-time algorithm only if $w(\log w + 1)/(k-1) > 2w$, that is, if $w = \Omega(4^k)$; but in this case, since λ is defined by the minimum, our new algorithm takes $\mathcal{O}(mw \cdot |\Sigma|^w)$ time. Thus, for alphabets of size at most 4, in the extreme edge case, our new algorithm is at least a $\sqrt{w}$ factor faster than the state of the art. Note that since $n \leq |\Sigma|^{k-1}$, for alphabets of size at most 4, this extreme edge case can only occur if w is linear in n, where by our Theorem 3 the problem is not tractable. Specifically, in that case, the running time of both the state of the art and our novel approach become unrealistic.

We describe how to encode a diET$[w]$ instance $(G = (V, E), c, w)$, where G is a dBG, using an auxiliary directed graph $H = (V', E')$ of size $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ in which there is a path between the designated source and target nodes if and only if the instance contains a c-respecting Eulerian trail. The graph H consists of a sequence of $m+1$ *layers* of nodes with edges only between consecutive layers. In this model, traversing an edge of H between layers $t-1$ and t means traversing a corresponding edge of G at time step t. We show the construction of graph $H = (V', E')$ and start by defining the construction of the nodes in V'.

The algorithms in [3, 7] rely on enumerating states, where a state is described by a node v of G (this is the end location of the current partial trail), a time step t, and some information about the edges of the partial trail. Both [3] and [7] achieve their FPT runtime by observing that it is unnecessary to encode the entire partial trail; instead, it suffices to consider the last w edges of it. They are even able to show that it is unnecessary to remember the order these edges are used on the trail. Applying Lemma 9 shows that, for each $v \in V$ and $t \in [m]$, it suffices to consider all subsets of a fixed size. These are at most $\binom{2w-1}{w} = \mathcal{O}(4^w/\sqrt{w})$; see the full version. We will use additional structural insights into dBGs to severely reduce the number of states that need to be considered in many relevant cases.

Henceforth, we set $\lambda \coloneqq \min(|\Sigma|^{k-1}, 2w-1)$ as in Theorem 11. For ease of notation, we also set $\ell \coloneqq \lceil \min(w,t)/(k-1) + 1 \rceil$ when t is clear from the context. Intuitively, ℓ is the number of nodes required to uniquely represent a path of length $\min(w,t)$ in G. By id_v we denote the string of length $k-1$ associated with the node $v \in V$.

The Nodes of H. We add a node to V' for every tuple (t, α) such that:

- $t \in [0, m]$ is a time step; and
- $\alpha \in \Sigma^{\min(w,t)+k-1}$ is a string of length $\min(w,t)+k-1$ over Σ, such that, for all $i \in [\ell]$, the node of G corresponding to the substring $\alpha[(i-1)(k-1)+1 \,..\, i(k-1)]$ of α has an incoming edge labeled $\alpha[i(k-1)]$, available at time step $t - w + (i-1)(k-1)$.

In order to avoid edge casing, we say that the incoming-edge requirement is fulfilled vacuously for time step 0: for layers $t \leq w$, we do not require the first encoded node to have an incoming edge. We also add to V' auxiliary source and target nodes, denoted by s and z, respectively.

▶ Remark 13. While this construction appears overtechnical at first glance, by the definition of a dBG $G = (V, E)$, the string α is equivalent to a sequence of nodes $v_\ell, \ldots, v_1 \in V$, such that v_i has an incoming edge labeled $\mathrm{id}_{v_i}[k-1]$ available at time step $t - (i-1) \cdot (k-1)$.

Intuitively, this construction relies on the insight that any path (of length r) in a dBG is fully described by the string (of length $r+k-1$) it generates and that this string in turn is fully described by examining every $(k-1)$-th node on the path.

The Edges of H. For a node (t, α) in V', we now potentially create an outgoing edge for each letter $x \in \Sigma$ to $(t+1, \alpha[2\,..\,|\alpha|] \cdot x)$, essentially deleting the leftmost letter of α and then appending x. However, we add the edge $(t, \alpha) \xrightarrow{x} (t+1, \alpha[2\,..\,|\alpha|] \cdot x)$ to E' only if:

- both the head and tail node exist in V'; and
- $\alpha[|\alpha| - k + 2\,..\,|\alpha|] \cdot x$ (the last $k-1$ letters of α appended with x) does not occur in α.

The first condition ensures that the head and tail nodes fulfill the condition that the encoded nodes have incident edges at the required time steps. The second condition ensures that the edge of G with label x that starts at the node v with $\mathrm{id}_v = \alpha[|\alpha| - k + 2\,..\,|\alpha|]$ is not traversed more than once in the (Eulerian) trail we aim to construct.

We also add an edge from the source node s to all nodes in layer 0 (i.e., all nodes where $t = 0$), and from all nodes in layer m (i.e., all nodes where $t = m$), to the target node z.

See Figure 4 for an illustration of this construction.

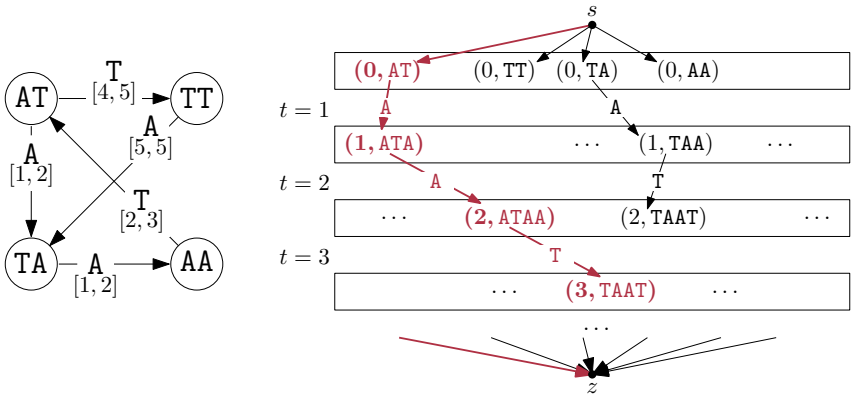


Figure 4 The input de Bruijn graph G (on the left) and the construction of graph H (on the right) for $k = 3$, $|\Sigma| = 2$, $w = 2$, and thus $w + k - 1 = 4$. For instance, for $t = 2$, $\ell = 2$, and $i = 2$, we add node $(t, \alpha) = (2, \texttt{ATAA})$ to layer $t = 2$ in H, because the node of G corresponding to $\alpha[(i-1)(k-1)+1\,..\,i(k-1)] = \texttt{AA}$ has an incoming edge labeled $\alpha[i(k-1)] = \texttt{A}$ available at time step $t - w + (i-1)(k-1) = 2$. A c-respecting Eulerian trail in G is indicated in red in H.

▶ **Lemma 14.** *Graph H has $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ nodes and $\mathcal{O}(m\,|\Sigma| \cdot \lambda^{\frac{w}{k-1}+1})$ edges, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$. Graph H can be constructed in time linear in its maximum size.*

Proof. Firstly, for any layer $t > 0$ in H, there can be at most $|\Sigma|^{w+k-1} = \left(|\Sigma|^{k-1}\right)^{\frac{w}{k-1}+1}$ choices for α. Secondly, by Lemma 9, there are at most $2w-1$ edges available at any time step $t > 0$, thus there can be at most that many nodes with at least one incoming edge available at t. Thus, by Remark 13, there can be at most $(2w-1)^\ell$ such strings α.

By construction, there are precisely n nodes in layer 0. Counting s and z as well and by the choice of λ, there are at most $2 + n + m \cdot \lambda^{\frac{w}{k-1}+1}$ nodes in H. For the edges, observe that every node (except s) has out-degree at most $|\Sigma|$. Node s has out-degree n and node z has in-degree the size of the last layer, that is at most $\lambda^{\frac{w}{k-1}+1}$. This yields the bound on the number of edges. We next discuss how H can be efficiently constructed.

To construct H efficiently, we perform a DFS over its *implicit* representation. For each node (t, α), we store the last letter of α, and associate each incoming edge with the letter it removed from the string in the previous layer to obtain α. The only part that requires a little care is when our DFS is at some node (t, α) and has to decide whether to add an edge with letter $x \in \Sigma$. In such a case, we have to check the condition that the string $\alpha[|\alpha|-k+2\,..\,|\alpha|] \cdot x$ does not occur in α. Instead of storing the entire string α and performing

the pattern matching explicitly, we use a single bit-string of length m during the DFS that maintains the set of w previously used edges (which are equivalent to the strings of length k that can appear in this form). This set can be maintained by correctly flipping the bits corresponding to the edges at the top and at the w-th position in the DFS stack every time the DFS moves forward or backward. In the word RAM model, each edge id is from $[m]$ and so it fits into one machine word. We therefore decide on the existence of each possible edge of H in $\mathcal{O}(1)$ time. To conclude, constructing H takes linear time in its maximum size. ◀

Formalizing the intuition behind the aim of the construction yields the following central lemma. See also Figure 4 that illustrates the properties stated in the lemma.

▶ **Lemma 15 (*).** *For any $t \in [m]$ and $\alpha \in \Sigma^{\min(w,t)+k-1}$, the graph H contains a path from s to the node labeled (t, α) if and only if there exists a trail $W = e_1 \dots e_t$ in G such that:*

1. *every edge from G occurs at most once in W;*
2. *for every $t' \in [t]$, the edge $e_{t'}$ is available at time step t';*
3. *the last w edges of W correspond to α; namely, they start at the node labeled $\alpha[1\,..\,k-1]$ and then follow the edges corresponding to $\alpha[k\,..\,|\alpha|]$.*

From Lemma 15, we can deduce the following crucial property, which both proves the correctness of our algorithm (i.e., Theorem 11) and underpins our extensions.

▶ **Lemma 16 (*).** *Graph H contains an s-to-z path if and only if there exists an Eulerian trail $W = e_1 \dots e_m$ in G, such that for every $i \in [m]$, edge e_i is available at time step i.*

Lemma 14 and Lemma 16 imply Theorem 11. We stress that, by Lemma 14, if we waive the assumption that $|\Sigma| = \mathcal{O}(1)$, we get an algorithm that is slower only by a factor of $|\Sigma|$.

Generalizations. We now discuss how to exploit this construction to solve more general problems. These extensions work analogously on the algorithm underlying Theorem 10 for general directed graphs. As a first extension, we can use these algorithms to report a witness: a c-respecting Eulerian trail in G. The following result makes this precise.

▶ **Corollary 17 (*).** *For any instance (G, c), where c is an interval function with width w, there is an $\mathcal{O}(m \cdot w^{1.5} 4^w)$-time algorithm finding a c-respecting Eulerian trail in G if it exists and correctly deciding non-existence otherwise. On a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$, we can solve this problem in $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda \coloneqq \min(|\Sigma|^{k-1}, 2w-1)$.*

For small $w \log w/(k-1)$ values, the algorithm on dBGs is highly efficient. At first glance, that is unintuitive, as even if for each pair of consecutive positions there are only two available k-mers (edges), there could already be $2^{m/2}$ possible orderings. Yet, our result shows that the additional structure of the graph allows us to efficiently select the correct ordering. The following remark captures the significant improvement over previous works which required w to be tractably small to efficiently solve the string reconstruction problem.

▶ Remark 18. Let $|\Sigma| = \mathcal{O}(1)$. For any $\frac{w \cdot \log w}{k-1} = \mathcal{O}(1)$, the diET$[w]$ problem on dBGs can be solved in $\mathcal{O}(mw)$ time. For any $w = \mathcal{O}(1)$, diET$[w]$ on dBGs can be solved in $\mathcal{O}(m)$ time.

It is important to note that in many practical applications, especially in bioinformatics and data privacy (see Figure 1), the dBGs considered are multigraphs (have parallel edges). For our FPT technique, the analysis above relies on the fact that an edge is uniquely determined by its endpoints, but with some extra care it can be adjusted to extend to multigraphs. We give a full discussion in the full version, but we restate the following main observation here.

▶ **Observation 19.** *If there is a c-respecting Eulerian trail in a de Bruijn multigraph $G = (V, E)$, with $m = |E|$, then there is a c-respecting Eulerian trail $W = e_1 \dots e_m$ such that for any time step t where the edge e_t has a parallel edge e' that is also available at t and appears after e_t on W, we have that* $\max c(e_t) \leq \max c(e')$.

Moreover, as mentioned above, we can generalize our technique to interval cost functions and retain the same running time, for both general directed graphs and for dBGs.

▶ **Corollary 20 (*).** *Given a* dicET$[w]$ *instance, there is an $\mathcal{O}(m \cdot w^{1.5} 4^w)$-time algorithm finding a min-cost Eulerian trail in G. On a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$, we can solve this problem in $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Note that in this setting, we can allow an edge to become temporarily unavailable by setting its cost to at least $c_{\text{budget}} + 1$. In such a case, the width of the edge's interval remains the difference between the last and the first time step at which the edge is available.

Corollary 20 directly implies that dicET parametrized by w is in FPT. On dBGs, dicET is also in FPT when parametrized by $w \log w/(k-1)$.

5 Counting Eulerian Trails with Applications in Data Privacy

Our results are of crucial relevance for the data privacy applications presented in [4] and [5]. On the one hand, our hardness results (Theorem 3 and Corollary 7) show that *in general* it is indeed hard to reconstruct a private (binary) string from a given dBG, thus providing theoretical justification for the privacy model introduced in [4]. On the other hand, our algorithmic results (Theorem 11 and Corollary 20) show that, with sufficiently specific domain knowledge, one can reconstruct such a (best) string in linear time. To address this concern, the algorithms of [4, 5] rely on a counting routine to efficiently ensure that the released string has at least κ possible reconstructions. To verify that the κ-anonymity property continues to hold even when reconstruction is aided by domain knowledge, we can directly employ the following result as the counting routine in the main algorithm of [4, 5].

▶ **Theorem 21.** *Given a* dicET$[w]$ *instance, we can count the number of min-cost Eulerian trails in $\mathcal{O}(m \cdot w^{1.5} 4^w)$ time. On a de Bruijn graph of order k over alphabet Σ, $|\Sigma| = \mathcal{O}(1)$, we can solve this problem in $\mathcal{O}(m \cdot \lambda^{\frac{w}{k-1}+1})$ time, where $\lambda := \min(|\Sigma|^{k-1}, 2w-1)$.*

Proof. Observe that by Lemma 16 and Corollary 20, there is a one-to-one correspondence between min-cost s-to-z paths in H and min-cost c-respecting Eulerian trails in G. Thus, we can apply the folklore linear-time shortest-path counting DAG dynamic program [25]. ◀

▶ Remark 22. The algorithm underlying Theorem 21 can be trivially extended to enumerate all such Eulerian trails in additional time that is linear in the size of the output.

References

1 Vsevolod A. Afanasev, René van Bevern, and Oxana Yu. Tsidulko. The hierarchical Chinese postman problem: The slightest disorder makes it hard, yet disconnectedness is manageable. *Oper. Res. Lett.*, 49(2):270–277, 2021. doi:10.1016/J.ORL.2021.01.017.

2 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Christoforos L. Raptopoulos. The temporal explorer who returns to the base. *J. Comput. Syst. Sci.*, 120:179–193, 2021. doi:10.1016/J.JCSS.2021.04.001.

3 Amir Ben-Dor, Itsik Pe'er, Ron Shamir, and Roded Sharan. On the complexity of positional sequencing by hybridization. *J. Comput. Biol.*, 8(4):361–371, 2002. doi:10.1089/106652701752236188.

4 Giulia Bernardini, Huiping Chen, Gabriele Fici, Grigorios Loukides, and Solon P. Pissis. Reverse-safe data structures for text indexing. In Guy E. Blelloch and Irene Finocchi, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020, Salt Lake City, UT, USA, January 5-6, 2020*, pages 199–213. SIAM, 2020. doi:10.1137/1.9781611976007.16.

5 Giulia Bernardini, Huiping Chen, Gabriele Fici, Grigorios Loukides, and Solon P. Pissis. Reverse-safe text indexing. *ACM J. Exp. Algorithmics*, 26:1.10:1–1.10:26, 2021. doi:10.1145/3461698.

6 Graham R. Brightwell and Peter Winkler. Counting Eulerian circuits is #P-complete. In Camil Demetrescu, Robert Sedgewick, and Roberto Tamassia, editors, *Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics, ALENEX /ANALCO 2005, Vancouver, BC, Canada, 22 January 2005*, pages 259–262. SIAM, 2005. URL: http://www.siam.org/meetings/analco05/papers/09grbrightwell.pdf.

7 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. *Algorithmica*, 85(3):688–716, 2023. doi:10.1007/S00453-022-01018-7.

8 Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I. Tomescu. Optimal omnitig listing for safe and complete contig assembly. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 29:1–29:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.CPM.2017.29.

9 Massimo Cairo, Romeo Rizzi, Alexandru I. Tomescu, and Elia C. Zirondelli. Genome assembly, from practice to theory: Safe, complete and *Linear-Time*. *ACM Trans. Algorithms*, 20(1):4:1–4:26, 2024. doi:10.1145/3632176.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Moshe Dror, Helman Stern, and Pierre Trudeau. Postman tour on a graph with precedence relation on arcs. *Networks*, 17(3):283–294, 1987. doi:10.1002/NET.3230170304.

12 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *J. Comput. Syst. Sci.*, 119:1–18, 2021. doi:10.1016/J.JCSS.2021.01.005.

13 Thomas Erlebach and Jakob T. Spooner. Faster exploration of degree-bounded temporal graphs. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 36:1–36:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.MFCS.2018.36.

14 Thomas Erlebach and Jakob T. Spooner. Non-strict temporal exploration. In Andréa Werneck Richa and Christian Scheideler, editors, *Structural Information and Communication Complexity - 27th International Colloquium, SIROCCO 2020, Paderborn, Germany, June 29 - July 1, 2020, Proceedings*, volume 12156 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2020. doi:10.1007/978-3-030-54921-3_8.

15 Gianpaolo Ghiani and Gennaro Improta. An algorithm for the hierarchical Chinese postman problem. *Oper. Res. Lett.*, 26(1):27–32, 2000. doi:10.1016/S0167-6377(99)00046-2.

16 Sridhar Hannenhalli, William Feldman, Herbert F. Lewis, Steven Skiena, and Pavel A. Pevzner. Positional sequencing by hybridization. *Comput. Appl. Biosci.*, 12(1):19–24, 1996. doi:10.1093/BIOINFORMATICS/12.1.19.

17 Carl Hierholzer and Chr Wiener. Über die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

18 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

19 Peter Korteweg and Ton Volgenant. On the hierarchical Chinese Postman Problem with linear ordered classes. *Eur. J. Oper. Res.*, 169(1):41–52, 2006. doi:10.1016/J.EJOR.2004.06.003.

20 Orna Kupferman and Gal Vardi. Eulerian paths with regular constraints. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *MFCS*, volume 58 of *LIPIcs*, pages 62:1–62:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.62.

21 Antoine Limasset, Jean-François Flot, and Pierre Peterlongo. Toward perfect reads: self-correction of short reads via mapping on de Bruijn graphs. *Bioinform.*, 36(2):651, 2020. doi:10.1093/BIOINFORMATICS/BTZ548.

22 Andrea Marino and Ana Silva. Eulerian walks in temporal graphs. *Algorithmica*, 85(3):805–830, 2023. doi:10.1007/S00453-022-01021-Y.

23 Paul Medvedev. Modeling biological problems in computer science: a case study in genome assembly. *Briefings Bioinform.*, 20(4):1376–1383, 2019. doi:10.1093/BIB/BBY003.

24 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theor. Comput. Sci.*, 634:1–23, 2016. doi:10.1016/J.TCS.2016.04.006.

25 Matús Mihalák, Rastislav Srámek, and Peter Widmayer. Approximately counting approximately-shortest paths in directed acyclic graphs. *Theory Comput. Syst.*, 58(1):45–59, 2016. doi:10.1007/S00224-014-9571-7.

26 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001. doi:10.1073/pnas.171285098.

27 Pierangela Samarati and Latanya Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In Alberto O. Mendelzon and Jan Paredaens, editors, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, page 188. ACM Press, 1998. doi:10.1145/275487.275508.

28 Jinghao Sun, Guozhen Tan, and Xianchao Meng. Graph transformation algorithm for the time dependent Chinese Postman Problem with Time Windows. In *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, pages 955–960, 2011. doi:10.1109/MEC.2011.6025623.

29 Latanya Sweeney. k-Anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.*, 10(5):557–570, 2002. doi:10.1142/S0218488502001648.

30 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *J. Comput. Biol.*, 24(6):590–602, 2017. doi:10.1089/CMB.2016.0141.

31 Hsiao-Fan Wang and Yu-Pin Wen. Time-constrained Chinese postman problems. *Computers & Mathematics with Applications*, 44(3):375–387, 2002. doi:10.1016/S0898-1221(02)00156-6.

Non-Boolean OMv: One More Reason to Believe Lower Bounds for Dynamic Problems

Bingbing Hu
University of California San Diego, La Jolla, CA, USA

Adam Polak
Bocconi University, Milan, Italy

Abstract

Most of the known tight lower bounds for dynamic problems are based on the Online Boolean Matrix-Vector Multiplication (OMv) Hypothesis, which is not as well studied and understood as some more popular hypotheses in fine-grained complexity. It would be desirable to base hardness of dynamic problems on a more believable hypothesis. We propose analogues of the OMv Hypothesis for variants of matrix multiplication that are known to be harder than Boolean product in the offline setting, namely: equality, dominance, min-witness, min-max, and bounded monotone min-plus products. These hypotheses are a priori weaker assumptions than the standard (Boolean) OMv Hypothesis and yet we show that they are actually equivalent to it. This establishes the first such fine-grained equivalence class for dynamic problems.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Complexity classes

Keywords and phrases Fine-grained complexity, OMv hypothesis, reductions, equivalence class

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.54

Funding Part of this work was done when both authors were at Max Planck Institute for Informatics.

Acknowledgements We thank anonymous reviewers for pointing us to the literature on the cell-probe model and for many useful hints on improving the manuscript.

1 Introduction

The job of a dynamic algorithm is to keep its output up to date whenever its input undergoes a local change, e.g., maintaining a shortest s–t path in a graph while it undergoes vertex deletions. Ideally each such update should take at most a polylogarithmic time, but at the very least it should be faster than it takes to recompute a solution from scratch. Despite great progress in the field, for many dynamic problems that goal is beyond the reach of current algorithmic techniques. Starting from the seminal paper by Pătraşcu [19], we often get to explain this hardness by fine-grained conditional lower bounds.

Most of the known *tight* lower bounds for dynamic problems are based on the OMv Hypothesis [11]. This hypothesis is not as widely studied and as well understood as some other hypotheses in fine-grained complexity, such as SETH, 3SUM Hypothesis, and APSP Hypothesis (see, e.g., [22]). It would be more desirable to base hardness of dynamic problems on these more popular (and hence also more believable) assumptions. Unfortunately, the existing lower bounds conditional on them are often not tight for dynamic problems. It seems likely that these hypotheses are not strong enough to explain the complexity of many dynamic problems. We may need to search for a different approach to the following glaring question:

Can we have tight lower bounds for dynamic problems
based on a hypothesis that is more believable than OMv?


© Bingbing Hu and Adam Polak;
licensed under Creative Commons License CC-BY 4.0



Recall that the OMv Hypothesis is about *Boolean product*; it asserts that computing the Boolean product of two $n \times n$ matrices requires cubic $n^{3-o(1)}$ time if the second matrix is given column by column in an online fashion. In the static (i.e., non-online) setting, Boolean product is arguably the easiest of the many studied variants of matrix multiplication. Indeed, it can be computed in time $O(n^\omega)$, where $\omega < 2.372$ [2] is the (integer) matrix multiplication exponent.[1]

In the static matrix product world, if the $O(n^\omega)$ running time is on the "fast" end of the spectrum, then the min-plus product (related to distance computations in graphs) marks the other end: the fastest known algorithm shaves only a subpolynomial factor over the naive cubic running time [26], and the APSP Hypothesis from fine-grained complexity essentially says that no $n^{3-o(1)}$-time algorithm is possible [23].

There are also numerous variants of matrix multiplication that seem to have an "intermediate" hardness on this spectrum. Examples include min-max product [21, 9], min-witness product [8], equality product (a.k.a. Hamming product [18]), dominance product [17], threshold product [12], plus-max product [20], ℓ_{2p+1} product [13], and many others. The fastest known algorithms for these problems have running times that are functions of the matrix multiplication exponent ω, and they converge to $O(n^{2.5})$ when $\omega = 2$. Although it is still an open problem whether this is necessarily the right complexity for all these problems, there are some partial results in the form of tight fine-grained reductions that suggest it might be the case [13, 15, 24].

1.1 Our contributions

The OMv Hypothesis (and the lower bounds it implies) would be a priori more believable if we could replace in its statement the Boolean product with some other product known to be harder in the static world. For instance, we can define in a similar way the Min-Max-OMv problem using the min-max product: Pre-process a matrix $M \in \mathbb{Z}^{n\times n}$, and then answer (one by one, in an online fashion) n queries, each of them asking, given a vector $v \in \mathbb{Z}^n$, to compute the min-max product of M and v, i.e., the vector $u \in \mathbb{Z}^n$ such that

$$u[i] := \min_{k\in[n]} \max\{M[i,k], v[k]\}.$$

Then we can state a corresponding hypothesis, let us call it the Min-Max-OMv Hypothesis, asserting that the Min-Max-OMv problem cannot be solved in truly subcubic time $O(n^{3-\varepsilon})$, for any $\varepsilon > 0$. This of course brings a question:

Can we still give tight reductions from Min-Max-OMv to those dynamic problems for which there are known reductions from (Boolean-)OMv?

It turns out, yes, we can! Somewhat surprisingly[2], we can even give a tight reduction from Min-Max-OMv to Boolean-OMv. This shows that the Min-Max-OMv Hypothesis and the standard (Boolean-)OMv Hypothesis are actually equivalent. Moreover, the min-max product is not a unique example of this phenomenon. We show more equivalent hypotheses based on several matrix products, which are harder than the Boolean product in the static setting (see Section 2 for the formal definitions).

[1] Moreover, the fastest known "combinatorial" algorithm for the Boolean product [1] does not give a similar improvement for the integer product.

[2] This is perhaps less surprising to those readers who are familiar with how the subcubic algebraic algorithms for the intermediate problems work; see Section 1.2.

▶ **Theorem 1.** *The following problems either all have truly subcubic algorithms or none of them do:*

- *Boolean-OMv;* $(\exists_k M[i,k] \wedge v[k])$
- *∃Equality-OMv;* $(\exists_k M[i,k] = v[k])$
- *∃Dominance-OMv;* $(\exists_k M[i,k] \leqslant v[k])$
- *Min-Witness-OMv;* $(\min\{k \mid M[i,k] \wedge v[k]\})$
- *Min-Max-OMv;* $(\min_k \max\{M[i,k], v[k]\})$
- *Bounded Monotone Min-Plus-OMv.* $(\min_k M[i,k] + v[k])$

For the Bounded Monotone Min-Plus-OMv problem the implied algorithm is randomized.

This conglomeration of equivalent problems can be interpreted as making the OMv Hypothesis more believable, and the conditional lower bounds based on it stronger. We recall two analogous conglomerations: the NP-complete problems and the problems equivalent to the All-Pairs Shortest Paths (APSP) problem under subcubic reductions. One of the reasons behind the great success of the theory of NP-completeness is its structural simplicity: many natural problems are NP-complete, and solving any of them efficiently would solve all of them efficiently, so they are all hard for the same underlying reason. For all the multi-faceted NP-complete problems, researchers from different areas have not managed to find a single efficient algorithm, so it seems very plausible that no such algorithm exists. The fine-grained complexity theory at large does not enjoy a similar simplicity: there are multiple independent hardness assumptions, and the reductions often go only in one way, establishing *hardness* but not *equivalence*. A notable exception[3] is the APSP problem, which is conjectured to require cubic time and there are many other problems equivalent to it via subcubic reductions [23]. No truly subcubic algorithms have been found so far for any of these problems, which strengthens the APSP Hypothesis. Our Theorem 1 establishes another such class of problems equivalent under fine-grained subcubic reductions.

Subcubic algorithms in the cell-probe model. We remark that the above equivalences also hold in the cell-probe model, where the running time is measured solely by the number of memory cell probes, and computation is free of charge. To be more specific, from our proofs it follows that each of the above problems can be solved by solving $t^{O(1)}$ instances of any other of those problems and performing some additional computation that runs in time $O(n^3/t)$, in the word RAM model, where t is a parameter to be chosen. Since any algorithm that runs sequentially in time $O(n^{3-\varepsilon})$ in the word RAM model can access only a truly subcubic number of memory cells, the subcubic equivalence in the cell-probe model follows. Larsen and Williams [14] showed that the OMv Hypothesis is actually *false* in the cell-probe model: Every query can be computed with $O(n^{7/4}w^{-1/2})$ probes, where w is the word size. This means that all the other problems listed above have truly subcubic algorithms in the cell-probe model as well.

1.2 Technical overview

We prove Theorem 1 by a series of fine-grained reductions, depicted in Figure 1. The reductions are inspired by known subcubic algebraic algorithms for the corresponding (static) matrix product problems. Such algorithms have running times that are functions of the (integer) matrix multiplication exponent ω. For example, the fastest known (static) min-max

[3] Another such exception is a class of vector problems equivalent to the Orthogonal Vectors problem [7].

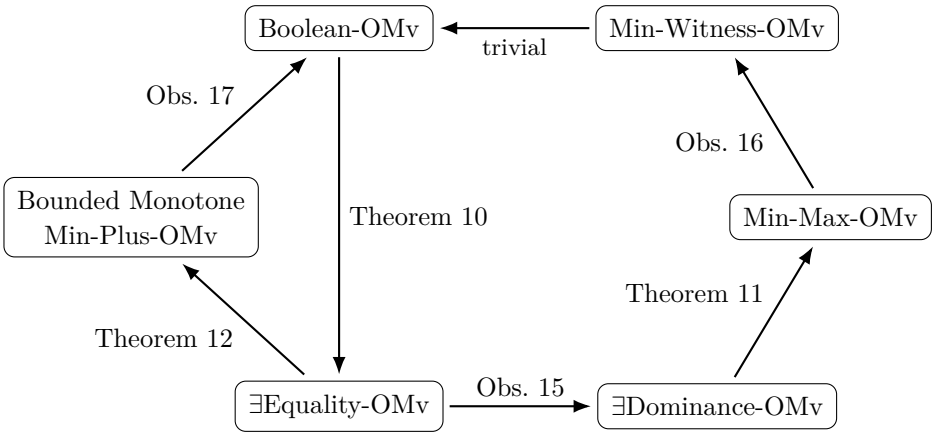


Figure 1 Fine-grained reductions that together prove Theorem 1. An arrow from problem A to problem B means that a subcubic algorithm for A implies a subcubic algorithm for B.

product algorithm [9] runs in time $O(n^{(3+\omega)/2+o(1)})$, which is subcubic for any value of $\omega < 3$. In other words, this algorithm is an implicit subcubic reduction from the (static) min-max product to the (static, integer) matrix multiplication problem. The same is true about all the other intermediate problems we consider in this paper. Our technical contribution is adapting these implicit reductions so that (1) they work in the online setting, and (2) they require only Boolean (and not integer) product.

Detour into static combinatorial algorithms. Before we say a few words about each of the reductions in the paper, let us take a detour from online problems so that we can explicitly state the observation that we already hinted at in the previous paragraph.

▶ **Observation 2.** *The static variants of all the matrix products listed in Theorem 1 are equivalent to each other under subcubic reductions, and the reductions are "combinatorial". That is, if any of the static problems admits a subcubic "combinatorial" algorithm, then all of them do.*

Why is this the case? In one direction, all those non-Boolean products can encode the Boolean product as a special case. In the other direction, all those non-Boolean products admit strongly subcubic algorithms with running times of the form $O(n^{f(\omega)})$ for a function f that happens to satisfy $f(x) < 3$ for every $x < 3$. Therefore, if we could replace in such an algorithm every use of fast algebraic (integer) matrix multiplication with a hypothesized $O(n^{3-\varepsilon})$-time "combinatorial" Boolean matrix multiplication algorithm, we would get a "combinatorial" algorithm for the corresponding non-Boolean product with the running time exponent $f(3-\varepsilon) < 3$. Such a replacement is possible whenever the non-Boolean algorithm does not use (or can be modified not to use) the full counting power of integer matrix multiplication but only checks which output entries are nonzero. This is explicitly the case for some of the non-Boolean products we consider (e.g., the min-witness product [8]). For some others, e.g., the bounded monotone min-plus product [25, 10], such a modification is nontrivial, but still possible.

While this observation is simple and in hindsight not really surprising, we are not aware of it being folklore in the field of fine-grained complexity, so we state it for completeness.

Subcubic reductions and where to find them. In Section 3 we show a subcubic reduction from ∃Equality-OMv to Boolean-OMv (Theorem 10), which can be seen as an adaptation to the online setting of the sparse matrix multiplication algorithm of Yuster and Zwick [27]. Specifically, our algorithm for ∃Equality-OMv handles the $n^{\varepsilon/2}$ most frequently appearing values with the help of $n^{\varepsilon/2}$ Boolean-OMv queries (about indicator vectors of those values). The remaining less frequent values can then be handled with a brute-force approach within the target running time bound.

In Section 4 we show how a subcubic algorithm for ∃Dominance-OMv would yield a subcubic algorithm for Min-Max-OMv (Theorem 11). The proof is inspired by the known (static) min-max product algorithms [21, 9], but it is at the same time simpler, because we do not have to optimize the dependence on ε in the running time of the resulting algorithm. In short, the algorithm splits each row of M into $n^{\varepsilon/2}$ buckets based on the values of the entries, and makes $n^{\varepsilon/2}$ many ∃Dominance-OMv queries in order to establish for each output entry to which bucket of the corresponding row it belongs; then, the algorithm exhaustively searches through the entire bucket.

In Section 5 we show a reduction from Bounded Monotone Min-Plus-OMv to ∃Equality-OMv. On a high level it follows some of the previous (static) algorithms for the bounded monotone min-plus product [25, 10]. However, it also gives a fresh perspective on the problem, because those previously known algorithms use a generalization of the min-witness and bounded (non-monotone) min-plus products (see [25, Theorem 1.2]), while ours deviates from this approach by using the equality product.

In Section 6 we show the remaining reductions (Observations 15, 16, 17). Each of them is either very simple or follows easily from folklore arguments.

1.3 Related work

Bringmann et al. [4] take a different approach at strengthening the OMv Hypothesis. They propose a hypothesis about the complexity of determining if a (nondeterministic) finite automaton accepts a word, and show that this hypothesis implies the OMv Hypothesis. While their new hypothesis is not as well supported as the three main fine-grained complexity hypotheses, it is remarkable that it is a statement about a *static* problem implying a tight lower bound for an online problem.

In a very recent work, Liu [16] shows that OMv is equivalent to the online problem of maintaining a $(1+\epsilon)$-approximate vertex cover in a fully dynamic bipartite graph.

To the best of our knowledge, the only other work that considers a variant of OMv for a non-Boolean product is by Chen et al. [6]. They use an assumption that the Min-Plus-OMv requires cubic time in order to separate partially retroactive from fully retroactive data structures. We note that this assumption seems too strong to be equivalent to the OMv Hypothesis. In particular, any "too simple" reduction from Min-Plus-OMv to Boolean-OMv would morally translate to a subcubic algorithm for the (static) Min-Plus Product problem, refuting the APSP Hypothesis.

1.4 Open problems

In this paper we manage to reduce to Boolean-OMv from OMv variants that *do not involve counting*. We leave it open whether a subcubic algorithm for Boolean-OMv would imply subcubic OMv algorithms for, e.g., the counting variants of the equality and dominance products (i.e., $u[i] := \#\{k \mid M[i,k] = v[k]\}$, and $u[i] := \#\{k \mid M[i,k] \leqslant v[k]\}$, respectively), or at least for the standard integer product ($u[i] := \sum_k M[i,k] \cdot v[k]$).

These open problems relate to the general quest for fine-grained *counting-to-decision* reductions. Chan, Vassilevska Williams, and Xu [5] gave such reductions for the Min-Plus Product, Exact Triangle, and 3SUM problems. Somewhat ironically, their reductions crucially rely on a fast algebraic algorithm for (static) integer matrix multiplication, so it seems unlikely that their techniques could be used to resolve the above open problems, which are about online problems.

2 Preliminaries

2.1 Notation

We use $[n] := \{1, 2, \ldots, n\}$.

2.2 Problems

In this section we formally define all the problems that appear in Theorem 1. Since the definitions are similar to each other, we underline the differences between them.

▶ **Definition 3** (Boolean-OMv). *We are first given for preprocessing a Boolean matrix $M \in \{0,1\}^{n\times n}$, and then we need to answer n queries: In the j-th query, we are given a column vector $v_j \in \{0,1\}^n$, and we have to compute the Boolean product $Mv_j \in \{0,1\}^n$ defined by*

$$(Mv_j)[i] := \begin{cases} 1, & \text{if } \exists_{k\in[n]}\, M[i,k] = 1 \wedge v_j[k] = 1, \\ 0, & \text{otherwise.} \end{cases}$$

We need to answer queries one by one in an online fashion, i.e., we have to output Mv_j before we can receive v_{j+1}.

▶ **Definition 4** (∃Equality-OMv). *We are first given for preprocessing an integer matrix $M \in \mathbb{Z}^{n\times n}$, and then we need to answer n queries: In the j-th query, we are given a column vector $v_j \in \mathbb{Z}^n$, and we have to compute the ∃equality product $M \ominus v_j \in \{0,1\}^n$ defined by*

$$(M \ominus v_j)[i] := \begin{cases} 1, & \text{if } \exists_{k\in[n]}\, M[i,k] = v_j[k], \\ 0, & \text{otherwise.} \end{cases}$$

We need to answer queries one by one in an online fashion, i.e., we have to output $M \ominus v_j$ before we receive v_{j+1}.

▶ **Definition 5** (∃Dominance-OMv). *We are first given for preprocessing an integer matrix $M \in \mathbb{Z}^{n\times n}$, and then we need to answer n queries: In the j-th query, we are given a column vector $v_j \in \mathbb{Z}^n$, and we have to compute the ∃dominance product $M \oslash v_j \in \{0,1\}^n$ defined by*

$$(M \oslash v_j)[i] := \begin{cases} 1, & \text{if } \exists_{k\in[n]}\, M[i,k] \leqslant v_j[k], \\ 0, & \text{otherwise.} \end{cases}$$

We need to answer queries one by one in an online fashion, i.e., we have to output $M \oslash v_j$ before we receive v_{j+1}.

▶ **Definition 6** (Min-Witness-OMv). *We are first given for preprocessing a Boolean matrix $M \in \{0,1\}^{n\times n}$, and then we need to answer n queries: In the j-th query, we are given a column vector $v_j \in \{0,1\}^n$, and we have to compute the min-witness product $M \circledcirc v_j \in ([n] \cup \{\infty\})^n$ defined by*

$$(M \circledcirc v_j)[i] := \min(\{k \in [n] \mid M[i,k] = 1 \wedge v_j[k] = 1\} \cup \{\infty\}).$$

We need to answer queries one by one in an online fashion, i.e., we have to output $M \circledcirc v_j$ before we can receive v_{j+1}.

▶ **Definition 7** (Min-Max-OMv). *We are first given for preprocessing an integer matrix $M \in \mathbb{Z}^{n\times n}$, and then we need to answer n queries: In the j-th query, we are given a column vector $v_j \in \mathbb{Z}^n$, and we have to compute the min-max product $M \circledcirc v_j \in \mathbb{Z}^n$ defined by*

$$(M \circledcirc v_j)[i] := \min_{k\in[n]} \max\{M[i,k], v_j[k]\}.$$

We need to answer queries one by one in an online fashion, i.e., we have to output $M \circledcirc v_j$ before we receive v_{j+1}.

▶ **Definition 8** (Bounded Monotone Min-Plus-OMv). *We are first given for preprocessing an integer matrix $M \in [n]^{n\times n}$, and then we need to answer n queries: In the j-th query, we are given a column vector $v_j \in [n]^n$, and we have to compute the min-plus product $M \oplus v_j \in \mathbb{Z}^n$ defined by*

$$(M \oplus v_j)[i] := \min_{k\in[n]} (M[i,k] + v_j[k]).$$

We need to answer queries one by one in an online fashion, i.e., we have to output $M \oplus v_j$ before we receive v_{j+1}. We are guaranteed that at least one of the following conditions holds:

- *each row of M is nondecreasing, i.e., $M[i,k] \leqslant M[i,k+1]$;*
- *each column of M is nondecreasing, i.e., $M[i,k] \leqslant M[i+1,k]$;*
- *each v_j is nondecreasing, i.e., $v_j[k] \leqslant v_j[k+1]$;*
- *for every k, $v_j[k]$ is a nondecreasing function of j, i.e., $v_j[k] \leqslant v_{j+1}[k]$;*

or it holds with "nondecreasing" replaced by "nonincreasing".

2.3 Hypotheses

Each of the problems defined above admits a naive cubic time algorithm, and for each of them we can conjecture that it is optimal up to subpolynomial factors.

▶ **Definition 9** (*-OMv Hypotheses). *For $x \in \{$Boolean, ∃Equality, ∃Dominance, Min-Witness, Min-Max, Bounded Monotone Min-Plus$\}$, the x-OMv Hypothesis is the statement that there is no algorithm for the x-OMv problem running in time $O(n^{3-\varepsilon})$, for any $\varepsilon > 0$.*

In other words, Theorem 1 says that all the hypotheses stated in Definition 9 are equivalent.

3 Reduction from ∃Equality-OMv to Boolean-OMv

In this section we give a subcubic reduction that can be seen as an easy adaptation to the online setting of the sparse matrix multiplication algorithm of Yuster and Zwick [27].

▶ **Theorem 10.** *If Boolean-OMv can be solved in time $O(n^{3-\varepsilon})$, for some $\varepsilon > 0$, then ∃Equality-OMv can be solved in time $O(n^{3-(\varepsilon/2)})$.*

Proof. Recall that M denotes the input matrix given for preprocessing in the ∃Equality-OMv problem. Let $t := \lceil n^{\varepsilon/2} \rceil$ be a parameter. For every $k \in [n]$ and every $\ell \in [t]$, let $f_k^{(\ell)}$ be the ℓ-th most frequent value appearing in the k-th column of matrix M (if there are less than ℓ distinct values in the column, let $f_k^{(\ell)}$ be some other arbitrary integer). Note that for any value x not in $\{f_k^{(1)}, f_k^{(2)}, \ldots, f_k^{(t)}\}$, x appears in the k-th column of M at most n/t times; we call such values *rare*. In the preprocessing phase, the algorithm prepares t Boolean matrices $M^{(1)}, M^{(2)}, \ldots, M^{(t)} \in \{0,1\}^{n \times n}$ defined as follows:

$$M^{(\ell)}[i,k] := \begin{cases} 1, & \text{if } M[i,k] = f_k^{(\ell)}, \\ 0, & \text{otherwise.} \end{cases}$$

Then, it instantiates the hypothesized Boolean-OMv algorithm for each of these matrices separately. Finally, for each column of M, the algorithm prepares a dictionary mapping each rare value in that column to a list of indices under which that value appears in the column. This ends the preprocessing phase.

Upon receiving a query $v \in \mathbb{Z}^n$, the algorithm first initializes the output vector to all zeros. Then, for every $\ell = 1, \ldots, t$, it creates the vector $v^{(l)}$ defined by

$$v^{(\ell)}[k] := \begin{cases} 1, & \text{if } v[k] = f_k^{(\ell)}, \\ 0, & \text{otherwise,} \end{cases}$$

and computes the Boolean product $M^{(\ell)}v^{(\ell)}$, using the ℓ-th instantiation of the hypothesized Boolean-OMv algorithm. Each such product gets then element-wise OR-ed to the output vector. Finally, for every $k = 1, \ldots, n$, if $v[k]$ is a rare value in the k-th column of matrix M, the algorithm goes through the list of all indices i such that $M[i][k] = v[k]$ (recall that there are at most n/t of them) and for each of them sets the corresponding i-th entry of the output vector to 1.

It is easy to see that whenever the algorithm sets an output entry to 1, it is because of some pair of entries $M[i][k]$ and $v[k]$ that have the same value. Conversely, if some pair of entries $M[i][k]$ and $v[k]$ have the same value, then either it is a frequent value and some $M^{(\ell)}v^{(\ell)}$ contributes a 1, or it is a rare value and gets manually matched.

Let us analyze the running of our ∃Equality-OMv algorithm. There are t instantiations of the hypothesized Boolean-OMv algorithm, which require $O(tn^{3-\varepsilon})$ time in total. Then, going through all rare values takes at most $O(n^2/t)$ time per v_j, and thus $O(n^3/t)$ time for all n queries. This adds up to total time $O(tn^{3-\varepsilon} + n^3/t)$. By choosing $t := \lceil n^{\varepsilon/2} \rceil$ we get the claimed running time $O(n^{3-(\varepsilon/2)})$. ◀

4 Reduction from Min-Max-OMv to ∃Dominance-OMv

In this we show how a subcubic algorithm for ∃Dominance-OMv would yield a subcubic algorithm for Min-Max-OMv. Our proof is inspired by the static min-max product algorithms [21, 9], but it is at the same time simpler, because we do not optimize the dependence on ε in the running time of the resulting algorithm.

▶ **Theorem 11.** *If ∃Dominance-OMv can be solved in time $O(n^{3-\varepsilon})$, for some $\varepsilon > 0$, then Min-Max-OMv can be solved in time $O(n^{3-(\varepsilon/2)})$.*

Proof. Let $t := \lceil n^{\varepsilon/2} \rceil$ be a parameter. For every $i \in [n]$, let R_i be the sorted i-th row of the input matrix M. Consider partitioning each R_i into t *buckets* of consecutive elements, with at most $\lceil n/t \rceil$ elements per bucket. For every $\ell \in [t]$, let $M^{(\ell)} \in (\mathbb{Z} \cup \{\infty\})^{n \times n}$ be the matrix defined as follows:

$$M^{(\ell)}[i,k] := \begin{cases} -M[i,k], & \text{if } M[i,k] \text{ lands in the } \ell\text{-th bucket of } R_i, \\ \infty, & \text{otherwise.} \end{cases}$$

Note that each row of $M^{(\ell)}$ contains $\Theta(n/t)$ finite entries.[4]

In the preprocessing phase, the algorithm instantiates the hypothesised ∃Dominance-OMv algorithm for each of the matrices $M^{(1)}, M^{(2)}, \ldots, M^{(\ell)}$, and also for the matrix M.[5]

Upon receiving a query $v \in \mathbb{Z}^n$, the algorithm proceeds to compute the product $M \circledcirc v$ in two steps. First, for every $i \in [n]$, it computes the minimum $M[i,k]$ such that $M[i,k] \geqslant v[k]$, and stores the results in a column vector u. Second, for every $i \in [n]$, it computes the minimum $v[k]$ such that $v[k] \geqslant M[i,k]$, and stores the results in a column vector w. At the very end the algorithm computes $(M \circledcirc v)[i] = \min\{u[i], w[i]\}$, for every $i \in [n]$.

In order to compute u, the algorithm first asks for the dominance products $M^{(\ell)} \circledgtr (-v)$, for all $\ell \in [t]$. Then, for each $i = 1, \ldots, n$, the algorithm finds the smallest ℓ such that $(M^{(\ell)} \circledgtr (-v))[i] = 1$, which corresponds to finding the first bucket in R_i containing an element *greater*[6] than or equal to the corresponding element in v. Hence, the algorithm can scan the elements in this bucket and pick the smallest one that is larger than or equal to the corresponding element in v; this element is then stored in $u[i]$.

Let us analyze the cost of computing u's over the span of n queries. The t dominance products require time $O(tn^{3-\varepsilon})$ in total. On top of that, for each of the n queries and for each of the n output coordinates, the algorithm scans one bucket of size $\Theta(n/t)$, which takes time $O(n^3/t)$ in total. All together, the algorithm spends time $O(tn^{3-\varepsilon} + n^3/t) = O(n^{3-(\varepsilon/2)})$ on computing u's.

Next, it is almost symmetric to calculate w. The algorithm sorts the entries of v into an ordered list S, and partitions S into t buckets, with at most $\lceil n/t \rceil$ elements per bucket. For each bucket $\ell \in [t]$, the algorithm computes the dominance product $M \circledless v^{(\ell)}$, where $v^{(\ell)} \in (\mathbb{Z} \cup \{-\infty\})^n$ is the column vector such that

$$v^{(\ell)}[k] = \begin{cases} v[k], & \text{if } v[k] \text{ lands in the } \ell\text{-th bucket of } S, \\ -\infty, & \text{otherwise.} \end{cases}$$

Then, for each $i = 1, \ldots, n$, the algorithm looks for the smallest ℓ such that $(M \circledless v^{(\ell)})[i] = 1$, and scans the elements in the ℓ-th bucket looking for the smallest $v[k]$ that is greater than or equal to the corresponding $M[i,k]$. By the same argument as before, computing all w's takes time $O(n^{3-(\varepsilon/2)})$. ◀

[4] If there are multiple entries with the same value, they may land in different buckets.

[5] Formally, the ∃Dominance-OMv algorithm may not accept infinite entries in the input, but we can replace each ∞ with $W + 2$, where W denotes the largest absolute value of any entry in M, and each entry greater than W in any query vector with $W + 1$.

[6] This is because the entries in $M^{(\ell)}$ and $-v$ are negated.

5 Reduction from Bounded Monotone Min-Plus-OMv to ∃Equality-OMv

In this section we show a reduction from Bounded Monotone Min-Plus-OMv to ∃Equality-OMv. We follow the high-level approach of some of the previous static algorithms for the bounded monotone min-plus product [25, 10]. However, we deviate from that approach by using the equality product where the known algorithms use a generalization of the min-witness and bounded (non-monotone) min-plus products (see [25, Theorem 1.2]).

▶ **Theorem 12.** *If ∃Equality-OMv can be solved in time $O(n^{3-\varepsilon})$, for some $\varepsilon > 0$, then Bounded Monotone Min-Plus-OMv can be solved in time $O(n^{3-(\varepsilon/3)} \log n)$ by a randomized algorithm that succeeds with probability*[7] *at least $1 - 1/n$.*

Before we present the algorithm itself let us introduce some notation and prove some preliminary facts. Let $\Delta := \lceil n^{\varepsilon/3} \rceil$ be a parameter. For a fixed query vector $v \in \mathbb{Z}^n$, let

$$u := M \oplus v, \quad \widehat{M} := \lfloor M/\Delta \rfloor, \quad \widehat{v} := \lfloor v/\Delta \rfloor, \quad \text{and} \quad \widehat{u} := \widehat{M} \oplus \widehat{v}.$$

Be mindful that it is *not* necessarily the case that $\widehat{u} = \lfloor u/\Delta \rfloor$. Finally, for every $i \in [n]$, let us define the set of *candidates for* $u[i]$ to be

$$C_i := \left\{ k \in [n] \;\middle|\; \widehat{M}[i,k] + \widehat{v}[k] \in \{\widehat{u}[i], \widehat{u}[i]+1\} \right\}.$$

▶ **Lemma 13.** *It suffices to check only $k \in C_i$ in order to compute $u[i]$, i.e.,*

$$\min_{k \in [n]} M[i,k] + v[k] = \min_{k \in C_i} M[i,k] + v[k].$$

Proof. First, for any pair $(i,j) \in [n] \times [n]$, due to rounding down we have

$$M[i,j] + v[j] - \Delta \cdot (\widehat{M}[i,j] + \widehat{v}[j]) \in [0, 2\Delta).$$

Now, suppose that k is a witness for $u[i]$, and l is a witness for $\widehat{u}[i]$, i.e., $M[i,k] + v[k] = u[i]$, and $\widehat{M}[i,l] + \widehat{v}[l] = \widehat{u}[i]$. We derive that

$$\begin{aligned} \Delta \cdot \widehat{u}[i] + 2\Delta &= \Delta \cdot (\widehat{M}[i,l] + \widehat{v}[l]) + 2\Delta \\ &> M[i,l] + v[l] \\ &\geqslant M[i,k] + v[k] \\ &\geqslant \Delta \cdot (\widehat{M}[i,k] + \widehat{v}[k]). \end{aligned}$$

Therefore, we have $\widehat{M}[i,k] + \widehat{v}[k] < \widehat{u}[i] + 2$. Since the matrix entries all take integer values, we have that if $k \in [n]$ is a witness for $u[i]$, then it must satisfy that $\widehat{M}[i,k] + \widehat{v}[k] \in \{\widehat{u}[i], \widehat{u}[i]+1\}$, i.e., $k \in C_i$. ◀

Now we argue that small sets of candidates can be enumerated efficiently.

▶ **Lemma 14.** *For a fixed query vector $v \in \mathbb{Z}^n$, there is an algorithm that runs in time $O(n^2 \log n/\Delta)$ and lists all elements of all sets C_i such that $|C_i| \leqslant n/\Delta$. In the case that $v_j[k]$ is a (weakly) monotone function of j (i.e., the 4-th case in Definition 8) this running time is amortized over n query vectors.*

[7] Note that the success probability can be amplified to $1 - 1/\operatorname{poly}(n)$ by running in parallel a constant number of copies of the algorithm and taking the majority vote.

Proof. We consider four cases, based on the direction of the monotonicity:

(1) Each column of M is monotone. In this case also the columns of $\widehat{M}$ are monotone, and their entries are bounded by $\lfloor n/\Delta \rfloor$. The algorithm uses a self-balancing binary search tree (BST) to maintain, while i iterates from 1 to n, the set of pairs

$$\big\{(\widehat{M}[i,k] + \widehat{v}[k], k) \;\big|\; k \in [n]\big\}.$$

Computing $\widehat{u}[i]$ is the standard tree operation of querying for the minimum. Moreover, the BST can report the number of elements smaller than a certain value in time $O(\log n)$, and enumerate them in time proportional to that number. This allows the algorithm to determine the size of C_i quickly, and enumerate it if it is small. As i iterates from 1 to n, the algorithm only needs to update the elements where there is an increase (or decrease) from $\widehat{M}[i,k]$ to $\widehat{M}[i+1,k]$. In each column of M there are at most n/Δ such changes, thanks to the boundedness and monotonicity.[8] Therefore the total number of updates over the n iterations is at most n^2/Δ, and each update takes time $O(\log n)$. The time spent on listing elements of C_i (for all i) is $O(n \log n + n^2/\Delta)$.

(2) For each k, $v_j[k]$ is a monotone function of j. This case is very similar to the previous one. The algorithm maintains (over the span of n queries) a separate BST for each i, and uses it to compute $(\widehat{M} \oplus \widehat{v}_j)[i]$ for all j's. When there is an increase (or decrease) from $\widehat{v}_j[k]$ to $\widehat{v}_{j+1}[k]$, the algorithm has to update an element in all n trees, but this happens at most n/Δ times for each k, so n^2/Δ times for all k's. Hence, the total time spent on such updates over the course of n queries is $O(n^3 \log n/\Delta)$, and the amortized time per query is $O(n^2 \log n/\Delta)$.

(3) Each row of M is monotone. Due to the monotonicity, we can think of the i-th row of $\widehat{M}$, for each $i = 1, \ldots, n$, as consisting of $\lfloor n/\Delta \rfloor + 1$ contiguous *blocks* $K_i^{(0)}, K_i^{(1)}, \ldots, K_i^{(\lfloor n/\Delta \rfloor)} \subseteq [n]$ of identical entries, i.e., $\forall_{k \in K_i^{(x)}}\, M[i,k] = x$. Upon receiving a query vector v, the algorithm builds (in linear time) a range minimum query (RMQ) data structure (see, e.g., [3]) in order to compute in constant time the minimum entry of $\widehat{v}$ in each of the $O(n^2/\Delta)$ blocks, i.e., $\widehat{v}[[K_i^{(x)}]] := \min\{\widehat{v}[k] \mid k \in K_i^{(x)}\}$. Adding each of these minima to their corresponding values from $\widehat{M}$ gives a list of candidate values for $\widehat{u}[i]$'s, i.e.,

$$\widehat{u}[i] = \min\big\{0 + \widehat{v}[[K_i^{(0)}]],\; 1 + \widehat{v}[[K_i^{(1)}]],\; \ldots,\; \lfloor n/\Delta \rfloor + \widehat{v}[[K_i^{(\lfloor n/\Delta \rfloor)}]]\big\}.$$

Thus, we already know how to compute $\widehat{u}$ is time $O(n^2/\Delta)$. Now let us explain how to extend this idea to also list elements of all small enough C_i's. For each value that appears in $\widehat{v}$, the algorithm calculates the sorted sequence of indices under which this value appears in $\widehat{v}$. This allows computing in time $O(\log n)$ how many times a given value appears in a given range of indices in $\widehat{v}$; indeed, it boils down to performing two binary searches of the two endpoints of the range in the sequence corresponding to the given value. Furthermore, all these appearances can be enumerated in time proportional to their count. For each block $K_i^{(x)}$ such that $x + \widehat{v}[[K_i^{(x)}]] = \widehat{u}[i]$ the algorithm enumerates all appearances of $\widehat{v}[[K_i^{(x)}]]$ and $\widehat{v}[[K_i^{(x)}]] + 1$ in the range $K_i^{(x)}$ in $\widehat{v}$, and adds them to C_i. If the total size of C_i would exceed n/Δ, the algorithm stops the enumeration and proceeds to the next block. Similarly, for each block such that $x + \widehat{v}[[K_i^{(x)}]] = \widehat{u}[i] + 1$ the algorithm enumerates all appearances of $\widehat{v}[[K_i^{(x)}]]$.

[8] The locations of all such changes can be, e.g., precomputed before processing any query, or computed during each query using binary search in time $O(\log n)$ per each location.

(4) Each v is monotone. This case is symmetric to the previous one. The difference is that now the algorithm splits $\widehat{v}$ into $O(n/\Delta)$ blocks, and prepares an RMQ data structure for each row of $\widehat{M}$. ◀

Now we are ready to present our subcubic algorithm for Bounded Monotone Min-Plus OMv, assuming a subcubic algorithm for ∃Equality-OMv.

Proof of Theorem 12. In the preprocessing, the algorithm samples uniformly and independently at random a set $R \subseteq [n]$ of columns of M, of size $|R| := \lceil 3\Delta \ln n \rceil$. For each $r \in R$, the algorithm prepares an ∃Equality-OMv data structure for matrix $M^{(r)}$ obtained from M by subtracting the r-th column from all the columns, i.e.,

$$M^{(r)}[i,k] := M[i,k] - M[i,r].$$

The algorithm handles each query in two independent steps. The goal of the first step is to compute $u[i]$ for those i that have $|C_i| \leqslant n/\Delta$, and the goal of the second step is to compute $u[i]$ for i with $|C_i| > n/\Delta$.

First step. For each $i \in [n]$, the algorithm either finds out that $|C_i| > n/\Delta$, or lists all elements of C_i and then computes $u[i] = \min_{k \in C_i}(M[i,k] + v[k])$. By Lemma 14, this takes time $O(n^2 \log n/\Delta)$, for all i's in total. The correctness of this step follows from Lemma 13.

Second step. In the second step, the algorithm must compute the remaining $u[i]$'s, i.e., those for which C_i's contain too many elements to be handled in the first step. To this end, for every $r \in R$ and every $\delta \in \{0, 1, \ldots, 3\Delta - 2\}$ the algorithm computes the equality product $M^{(r)} \ominus -(v - v[r] + \delta)$. For every $i \in [n]$, if $(M^{(r)} \ominus -(v - v[r] + \delta))[i]{=}1$, then there must exist $k \in [n]$ such that

$$M[i,k] - M[i,r] = -(v[k] - v[r] + \delta)$$

and hence

$$M[i,k] + v[k] = M[i,r] + v[r] - \delta.$$

The algorithm therefore adds $M[i,r] + v[r] - \delta$ to the list of possible values for $u[i]$, and at the end of the process it sets each $u[i]$ to the minimum over those values.

Analysis of the second step. We now argue that if $R \cap C_i \neq \emptyset$ (which holds with high probability when $|C_i| > n/\Delta$ via a standard hitting set argument, see below), then the algorithm correctly computes $u[i]$ in the second step. Indeed, pick $r \in R \cap C_i$ and let $k \in [n]$ be a witness for $(M \oplus v)[i]$, i.e., $M[i,k]+v[k] = (M \oplus v)[i]$. Let $\delta := (M[i,r]+v[r]) - (M[i,k]+v[k])$. Clearly, $(M^{(r)} \ominus -(v - v[r] + \delta))[i]{=}1$, so it only remains to show that $\delta \in \{0, 1, \ldots, 3\Delta - 2\}$. Obviously, $\delta \geqslant 0$, because k minimizes $M[i,k] + v[k]$. Now let us upper bound the offset δ. Since $r \in C_i$, we have $\widehat{M}[i,r] + \widehat{v}[r] \leqslant \widehat{u}[i] + 1$, and hence

$$M[i,r] + v[r] \;\leqslant\; (\Delta\widehat{M}[i,r] + \Delta - 1) + (\Delta\widehat{v}[r] + \Delta - 1) \;\leqslant\; \Delta\widehat{u}[i] + 3\Delta - 2.$$

Moreover, $\widehat{M}[i,k] + \widehat{v}[k] \geqslant \widehat{u}[i]$, and therefore

$$M[i,k] + v[k] \;\geqslant\; \Delta\widehat{M}[i,k] + \Delta\widehat{v}[k] \;\geqslant\; \Delta\widehat{u}[i].$$

We conclude that $\delta \leqslant (\Delta\widehat{u}[i] + 3\Delta - 2) - \Delta\widehat{u}[i] = 3\Delta - 2$, as required.

It remains to analyze the success probability of the whole algorithm. For a fixed output index $i \in [n]$ such that $|C_i| > n/\Delta$, the probability that the algorithm failed to sample an element r from C_i in all $|R| = \lceil 3\Delta \ln n \rceil$ rounds is at most $(1-1/\Delta)^{3\Delta \ln n} < (1/e)^{3 \ln n} = 1/n^3$. By a union bound over all n output indices for each of the n queries, the algorithm succeeds to correctly compute all n^2 output entries with probability at least $1 - n^2/n^3 = 1 - 1/n$.

Running time. The first step of each query (Lemma 14) takes time $O(n^2 \log n/\Delta)$, summing up to $O(n^3 \log n/\Delta)$ for all n queries. Regarding the second step, for each query the algorithm computes $O(|R|\Delta) = O(\Delta^2 \log n)$ equality matrix-vector products, and over the course of n queries this takes time $O(n^{3-\varepsilon}\Delta^2 \log n)$. The total running time is therefore $O(n^3 \log n/\Delta + n^{3-\varepsilon}\Delta^2 \log n) = O(n^{3-(\varepsilon/3)} \log n)$. ◀

6 Remaining reductions

In this section we state the remaining easy reductions that complete the proof of Theorem 1.

6.1 Reduction from ∃Dominance-OMv to ∃Equality-OMv

▶ **Observation 15.** *If ∃Equality-OMv can be solved in time $T(n)$, then ∃Dominance-OMv can be solved in time $O(T(n) \log n)$.*

Proof. The proof follows a folklore argument, see, e.g., [13]. It uses the fact that, for any two non-negative integers a and b, it holds that $a < b$ if and only if there exists $\ell \geqslant 0$ such that

- the ℓ-th least significant bit of a is 0; and
- the ℓ-th least significant bit of b is 1; and
- a agrees with b on all bits higher than the ℓ-th least significant, i.e., $\lfloor a/2^{\ell+1} \rfloor = \lfloor b/2^{\ell+1} \rfloor$.

Moreover, without loss of generality, all the input numbers are integers between 0 and $n^2 - 1$. Indeed, in the preprocessing, each entry of M can be replaced by its rank in the sorted order of all entries of M; then, during a query, each entry of v can be replaced by the rank of the smallest entry of M greater than or equal to it. Last but not least, $M[i,k] \leqslant v[k]$ if and only if $M[i,k] < v[k] + 1$, because the input numbers are integers. Hence, the algorithm sets, for $\ell = 0, 1, \ldots, \lceil \log(n^2) \rceil$,

$$M^{(\ell)}[i,k] := \begin{cases} \left\lfloor \frac{M[i,k]}{2^{\ell+1}} \right\rfloor, & \text{if the } \ell\text{-th least significant bit of } M[i,k] \text{ is } 0 \\ -1, & \text{otherwise,} \end{cases}$$

$$v^{(\ell)}[k] := \begin{cases} \left\lfloor \frac{v[k]+1}{2^{\ell+1}} \right\rfloor, & \text{if the } \ell\text{-th least significant bit of } v[k]+1 \text{ is } 1 \\ -2, & \text{otherwise,} \end{cases}$$

and uses the fact that $(M \circledcirc v)[i] = 1$ if and only if $\exists_\ell \ (M^{(\ell)} \ominus v^{(\ell)})[i] = 1$. ◀

6.2 Reduction from Min-Witness-OMv to Min-Max-OMv

▶ **Observation 16.** *If Min-Max-OMv can be solved in time $T(n)$, then Min-Witness-OMv can be solved in time $T(n) + O(n^2)$.*

Proof. The proof follows another folklore argument, see, e.g., [15]. The algorithm sets

$$M'[i,k] := \begin{cases} k, & \text{if } M[i,k] = 1 \\ \infty, & \text{otherwise,} \end{cases} \quad \text{and} \quad v'[k] := \begin{cases} k, & \text{if } v[k] = 1 \\ \infty, & \text{otherwise,} \end{cases}$$

and uses the fact $M \circledcirc v = M' \circledcirc v'$. ◀

6.3 Reduction from Boolean-OMv to Bounded Monotone Min-Plus-OMv

▶ **Observation 17.** *If Bounded Monotone Min-Plus-OMv can be solved in time $T(n)$, then Boolean-OMv can be solved in time $T(n) + O(n^2)$.*

Proof. The algorithm sets

$$M'[i,k] := 2 \cdot (i+k) - M[i,k], \quad \text{and} \quad v'_j[k] := 2 \cdot (j-k) - v_j[k],$$

and uses the fact that $(Mv_j)[i] = 1$ if and only if $(M' \oplus v'_j)[i] = 2 \cdot (i+j) - 2$. ◀

We remark that the above reduction produces Min-Plus-OMv instances that are monotone in all four directions simultaneously, while our Bounded Monotone Min-Plus-OMv algorithm of Theorem 12 works already for instances with monotonicity in one (arbitrarily chosen) direction.

References

1 Amir Abboud, Nick Fischer, Zander Kelley, Shachar Lovett, and Raghu Meka. New graph decompositions and combinatorial boolean matrix multiplication algorithms. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 935–943. ACM, 2024. doi:10.1145/3618260.3649696.

2 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.

3 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

4 Karl Bringmann, Allan Grønlund, Marvin Künnemann, and Kasper Green Larsen. The NFA acceptance hypothesis: Non-combinatorial and dynamic lower bounds. In *15th Innovations in Theoretical Computer Science Conference, ITCS 2024*, volume 287 of *LIPIcs*, pages 22:1–22:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ITCS.2024.22.

5 Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Fredman's trick meets dominance product: Fine-grained complexity of unweighted APSP, 3SUM counting, and more. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023*, pages 419–432. ACM, 2023. doi:10.1145/3564246.3585237.

6 Lijie Chen, Erik D. Demaine, Yuzhou Gu, Virginia Vassilevska Williams, Yinzhan Xu, and Yuancheng Yu. Nearly optimal separation between partially and fully retroactive data structures. In *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018*, volume 101 of *LIPIcs*, pages 33:1–33:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.SWAT.2018.33.

7 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 21–40. SIAM, 2019. doi:10.1137/1.9781611975482.2.

8 Artur Czumaj, Miroslaw Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theor. Comput. Sci.*, 380(1-2):37–46, 2007. doi:10.1016/J.TCS.2007.02.053.

9 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 384–391. SIAM, 2009. doi:10.1137/1.9781611973068.43.

10 Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhan Xu. Faster monotone min-plus product, range mode, and single source replacement paths. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 75:1–75:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.75.

11 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.

12 Piotr Indyk, Moshe Lewenstein, Ohad Lipsky, and Ely Porat. Closest pair problems in very high dimensions. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004*, volume 3142 of *Lecture Notes in Computer Science*, pages 782–792. Springer, 2004. doi:10.1007/978-3-540-27836-8_66.

13 Karim Labib, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Hamming distance completeness. In *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, volume 128 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.CPM.2019.14.

14 Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 2182–2189. SIAM, 2017. doi:10.1137/1.9781611974782.142.

15 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020*, volume 151 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.53.

16 Yang P. Liu. On approximate fully-dynamic matching and online matrix-vector multiplication. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 228–243. IEEE, 2024. doi:10.1109/FOCS61266.2024.00006.

17 Jirí Matousek. Computing dominances in e^n. *Inf. Process. Lett.*, 38(5):277–278, 1991. doi:10.1016/0020-0190(91)90071-O.

18 Kerui Min, Ming-Yang Kao, and Hong Zhu. The closest pair problem under the Hamming metric. In *Computing and Combinatorics, 15th Annual International Conference, COCOON 2009*, volume 5609 of *Lecture Notes in Computer Science*, pages 205–214. Springer, 2009. doi:10.1007/978-3-642-02882-3_21.

19 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.

20 Virginia Vassilevska. *Efficient algorithms for path problems in weighted graphs*. PhD thesis, Carnegie Mellon University, USA, 2008.

21 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory Comput.*, 5(1):173–189, 2009. Announced at STOC 2007. doi:10.4086/TOC.2009.V005A009.

22 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians, ICM 2018*, pages 3447–3487. World Scientific, 2018. doi:10.1142/9789813272880_0188.

23 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. Announced at FOCS 2010. doi:10.1145/3186893.

24 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and APSP. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 786–797. IEEE, 2020. doi:10.1109/FOCS46700.2020.00078.

25 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 12–29. SIAM, 2020. doi:10.1137/1.9781611975994.2.

26 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. Announced at STOC 2014. doi:10.1137/15M1024524.

27 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. Announced at ESA 2004. doi:10.1145/1077464.1077466.

Safe Sequences via Dominators in DAGs for Path-Covering Problems

Francisco Sena[1]
Department of Computer Science, University of Helsinki, Finland

Romeo Rizzi
Department of Computer Science, University of Verona, Italy

Alexandru I. Tomescu
Department of Computer Science, University of Helsinki, Finland

Abstract

A *path-covering* problem on a directed acyclic graph (DAG) requires finding a set of source-to-sink paths that cover all the nodes, all the arcs, or subsets thereof, and additionally they are optimal with respect to some function. In this paper we study *safe sequences* of nodes or arcs, namely sequences that appear in some path of every path cover of a DAG.

We show that safe sequences admit a simple characterization via cutnodes. Moreover, we establish a connection between maximal safe sequences and leaf-to-root paths in the source- and sink-dominator trees of the DAG, which may be of independent interest in the extensive literature on dominators. With dominator trees, safe sequences admit an $O(n)$-size representation and a linear-time output-sensitive enumeration algorithm running in time $O(m + o)$, where n and m are the number of nodes and arcs, respectively, and o is the total length of the maximal safe sequences.

We then apply maximal safe sequences to simplify Integer Linear Programs (ILPs) for two path-covering problems, LeastSquares and MinPathError, which are at the core of RNA transcript assembly problems from bioinformatics. On various datasets, maximal safe sequences can be computed in under 0.1 seconds per graph, on average, and ILP solvers whose search space is reduced in this manner exhibit significant speed-ups. For example on graphs with a large width, average speed-ups are in the range $50 - 250\times$ for MinPathError and in the range $80 - 350\times$ for LeastSquares. Optimizing ILPs using safe sequences can thus become a fast building block of practical RNA transcript assembly tools, and more generally, of path-covering problems.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Integer programming; Applied computing → Sequencing and genotyping technologies

Keywords and phrases directed acyclic graph, path cover, dominator tree, integer linear programming, least squares, minimum path error

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.55

Related Version *Full Version*: `https://arxiv.org/abs/2411.03871`

Supplementary Material *Software (Source Code)*: `https://github.com/algbio/safe-sequences` archived at `swh:1:dir:0e6e4d5e013053c8ba853d335c750cff701e09fa`

Funding This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO), and from the Research Council of Finland grants No. 346968, 358744. Co-funded by the European Union (ERC, SCALEBIO, 101169716). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements We are extremely grateful to Andrey Prjibelski for producing the IsoQuant graphs and to Manuel Cáceres for fruitful discussions on safe sequences.

[1] Corresponding author

© Francisco Sena, Romeo Rizzi, and Alexandru I. Tomescu;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Data reduction, namely simplifying the input to a problem while maintaining equivalent solutions, has proved extremely successful in solving hard problems in both theory and practice [26]. While the most well-known approach for data reduction is *kernelization* [37, 18], there are also other ways to pre-process the input in order to obtain practical algorithms.

For example, for graph problems where a solution is a set of vertices with certain properties, Bumpus et al. [8] and Jansen and Verhaegh [29] considered the notion of *c-essential vertex* as one belonging to *all* c-approximate solutions to the problem. They showed that for some vertex-subset finding problems (e.g. Vertex Cover) and some values of c, there exist algorithms with an exponential dependency only in the number of *non-c-essential vertices*.

In this paper we pursue a similar data reduction approach, via the notion of *safe partial solution* [52], defined as one "appearing" in all solutions to a problem. For solutions that are sets of paths satisfying certain properties, a *safe path* can be defined as one appearing as a subpath of some path of any solution. Intuitively, safety corresponds to 1-essentiality.

For an NP-hard problem, it is usually intractable also to compute all its safe partial solutions (see e.g. [15]). As such, in this paper we use the following property about safety. Let $\mathcal{S}$ be the (unknown) set of solutions to a problem, and let $\mathcal{S}' \supseteq \mathcal{S}$ be a superset of $\mathcal{S}$ whose safe partial solutions can be computed efficiently. Then, if a partial solution appears in all solutions in $\mathcal{S}'$, it also appears in all solutions in $\mathcal{S}$.

This property was used by Grigorjew et al. [25] when applying safety as a pre-processing step for the Minimum Flow Decomposition (MFD) problem. The MFD problem asks to decompose the flow in a directed acyclic graph (DAG) into a minimum number of weighted source-to-sink paths [4, 53]. As solution superset $\mathcal{S}'$, they considered flow decompositions with any number of paths, whose safe paths can be computed in polynomial time [30, 31, 2]. Instead of simplifying the input graph, they used the safe paths to simplify an integer linear program (ILP) for the MFD problem from [17], thus reducing the search space of the ILP solver. This resulted in speed-ups of up to 200 times on the hardest instances.

In this paper we focus on problems whose input consists of an s-t *DAG*[2] G where the solution superset $\mathcal{S}'$ contains all the sets of paths in G from s to t that together cover all the nodes or arcs of G (or given subsets thereof). We will refer to these objects simply as *path covers*. Note that a flow decomposition is also a path cover of the arcs carrying flow, but path covers generalize many other problems, especially those modeling practical problems from bioinformatics, see Section 1.2. Next, we present the contributions of the paper and further contextualization.

1.1 Optimal enumeration of maximal safe sequences via dominators

Safe sequences as a generalization of safe paths. The notion of a safe *path* has been extensively used to capture *contiguous* safe partial solutions (see e.g. [52, 48, 15, 40, 45, 31, 11]). However, for our purposes contiguity is not necessary, but only the fact that the nodes/arcs in a safe partial solution appear in the same order in some path of every solution. For example, it may be that the only way to reach a node u is via a node v, with some complex subgraph between them. The sequence u, v thus appears in some path of every path cover of the nodes.

[2] By an s-t *directed graph* we mean one with a unique source s and unique sink t such that all nodes are reachable from s and all nodes reach t; if the graph is acyclic, then it is an s-t *DAG*. Path-covering problems on DAGs typically require that paths either start in a source of the graph, or more generally, in a given set of starting nodes, and analogously end in a sink, or in a given set of terminal nodes. However, these start/end nodes can naturally be connected to s and t, and thus we can assume, without loss of generality, that our DAGs are s-t DAGs.

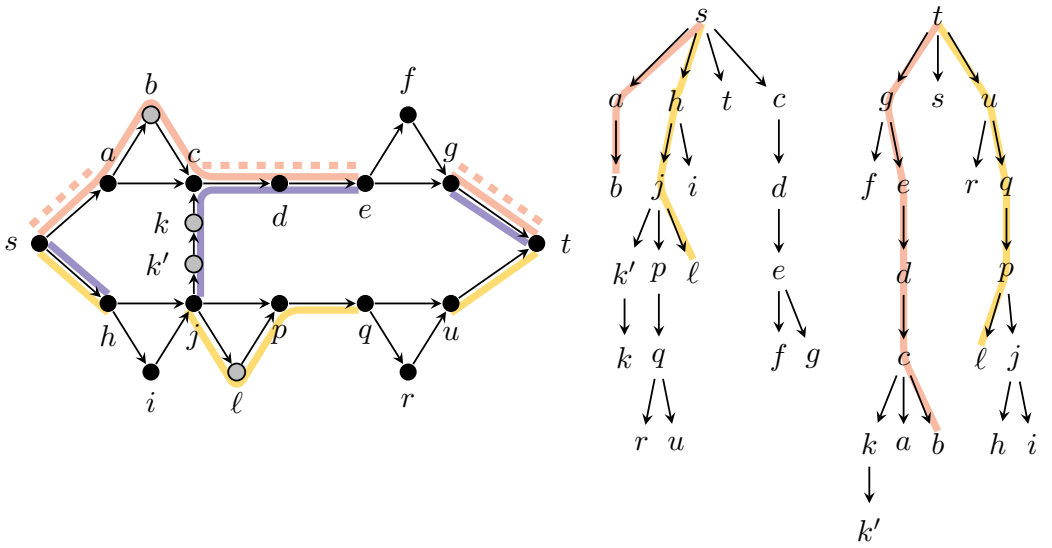


Figure 1 An s-t DAG and its s- and t-dominator trees. Three maximal safe sequences with respect to path covers of the nodes are shown in solid orange, violet, and yellow. Nodes b and ℓ are leaves in both dominator trees, and thus their extension into the orange and yellow sequences, resp., are maximally safe. (The maximal safe sequences obtained by extending nodes f, i and r are not shown.) Note that $k'k$ is a unitary path and that $\mathsf{extension}(k') = \mathsf{extension}(k)$. The dashed orange safe sequence is not maximal and can be obtained by extending a.

We thus define a *safe sequence* with respect to the path covers of a DAG G, either of the nodes or of the arcs, analogously to that of a safe path (see Figure 1 for an example). This notion is inspired by the work of Ma, Zheng and Kingsford [57], who proposed a problem (AND-Quant) and a max-flow-based algorithm deciding when a sequence of arcs appears in some path in every flow decomposition (with any number of paths) of a flow in a DAG. Here, we are particularly interested in characterizing but also in computing *maximal* safe sequences, i.e., safe sequences that are not a proper subsequence of another safe sequence.

Characterization of safe sequences. Assume that the input DAG has n nodes and m arcs. We show that safe sequences admit a simple characterization in terms of cutnodes. A u-v *cutnode* is a node such that all paths from u to v (u-v *paths*) pass through it. We present our results for path covers of the nodes and refer to the extended version for the results for path covers of the arcs. Omitted proofs can be found in the extended version.

▶ **Theorem 1** (Characterization of safe sequences). *Let $G = (N, A)$ be an s-t DAG and X be a sequence of nodes. The following statements are equivalent:*

1. *X is safe for path covers.*
2. *There is a node $u \in N$ such that every s-t path covering u contains X.*
3. *There is a node $v \in N$ such that X is contained in the sequence obtained as the concatenation of the sequences of s-v and v-t cutnodes.*

While outputting the sequences of s and t cutnodes for every node v in the graph can be done in time $O(m)$ per node, different nodes can thus be extended to the same maximal safe sequence, and some extensions may not even be maximal (see Figure 1). The challenge is to characterize those nodes identifying all and only maximal safe sequences, without duplicates.

Linear-time output-sensitive enumeration via dominators. We show that we can identify such nodes in an elegant manner using *dominators*, which is a well-studied concept related to cutnodes (see e.g. the survey [23] for an in-depth contextualization of dominators). In particular, the *s-dominator tree* of an *s-t* DAG (N, A) has node set N, and u is the parent of v iff all paths from s to v pass through u, and moreover there is no node reachable from u (except u itself) whose removal makes v unreachable from u.

The key idea behind our results is to work *also* with the t-dominator tree of G, which is defined in an analogous manner, but considering reachabilities *to* t. By analyzing the interplay between the two trees, we show that the nodes solving this problem (Theorem 9) are those that are *leaves in both* the s- and t-dominator trees, only after overcoming some technicalities arising from unitary paths (see Figure 1, Lemma 4 and Lemma 8). Since dominator tree leaves admit a characterization in terms of in- or out-neighborhoods on DAGs (Lemma 5), this leads to an $O(mn)$-time algorithm outputting all and only maximal safe sequences without duplicates, independently of dominator trees:

▶ **Theorem 2.** *Given an s-t DAG G with n nodes and m arcs, we can compute all and only the maximal safe sequences of G in $O(nm)$-time, without constructing dominator trees.*

To improve this time bound we resort to the fact that dominator trees admit an $O(m+n)$ time construction [6, 7, 20], although so far some of these algorithms do not seem to be practical. See [24, 22, 14] for experimental studies on different dominator tree algorithms. Regardless, any dominator algorithm serves our purposes and its effect on the runtime of our algorithm is clear in the proof of the theorem below.

▶ **Theorem 3** (Enumeration of maximal safe sequences)**.** *Let G be an s-t DAG with n nodes and m arcs. The following hold:*

1. *There is an $O(m + o)$ time algorithm outputting the set of all maximal safe sequences with no duplicates, where o denotes the total length of all the maximal safe sequences.*
2. *There is an $O(n)$-size representation of all the maximal safe sequences of G that can be built in $O(m + n)$ time.*

An analogous result to Theorem 3 can be obtained for path covers where only a given subset $C \subseteq N$ of nodes has to be covered. To achieve that, we have to analyze the dominator trees induced by the dominance relation restricted to C. Moreover, analogous results can be obtained also for path covers of the arcs. The details can be found in the extended version.

1.2 Scaling path-covering ILPs for bioinformatics problems

The motivation of our work comes from bioinformatics, where many NP-hard graph problems related to sequencing data require finding a set of paths in a DAG satisfying certain properties. For example, in the *RNA transcript assembly* problem, we need to recover the RNA transcripts produced in various abundances by a gene, given only smaller fragments (i.e. RNA-seq reads) sequenced from the RNA transcripts. This problem is usually solved by first building an arc-weighted DAG from the fragments, i.e., a *splice graph*. The RNA transcripts correspond to a set of source-to-sink weighted paths in the DAG that "best explain" the arc-weights, under various definitions of optimality, see e.g. [36, 35, 56, 46, 55, 47, 50, 19, 17, 15].

In our work we consider two path-covering problems used for RNA transcript assembly, in which we use safe sequences to fix some variables of the corresponding ILPs. We assume that in every arc uv of the input graph there is an associated positive weight $w(uv)$. We now briefly describe the ILPs.

We selected a classical least-squares model (which we call LeastSquares) that is at the core of several RNA assembly tools e.g. [34, 12, 21, 33, 38, 51]. This minimizes the sum of squared differences between the weight of an arc and the weights of the solution paths passing through the arc. The ILP formulation requires no additional constraints, and the ILP objective function is:[3]

$$\min \sum_{uv \in A} \left(w(uv) - \sum_{i=1}^{k} x_{uvi} w_i \right)^2 . \tag{1}$$

While this model is popular, it has quadratic terms in its objective function, which makes it harder to solve. Therefore, as second model (which we call MinPathError), we chose one that was recently introduced in [16]. Then, for k paths to be a solution to the ILP, we require that the absolute difference between the weight of an arc and the weights of the solution paths passing through the arc must be below the sum of the errors (or slacks) of the paths passing through the arc. Formally, we add the constraint:[4]

$$\left| w(uv) - \sum_{i=1}^{k} x_{uvi} w_i \right| \leq \sum_{i=1}^{k} x_{uvi} \rho_i, \quad \forall uv \in A, \tag{2}$$

with objective function: $\min \sum_{i=1}^{k} \rho_i$. By definition, the solution paths must cover every arc uv, since otherwise the above condition becomes $w(uv) \leq 0$, which contradicts the assumption that w is positive everywhere. See [17] for further details.

On various datasets, maximal safe sequences can be computed in under 0.1 seconds per graph, on average, and such safety-optimized ILPs show significant speed-ups over the original ones. More specifically, on the harder instances of graphs with a large width, average speed-ups are in the range $50 - 250\times$ for MinPathError and in the range $80 - 350\times$ for LeastSquares. As such, our optimizations can become a scalable building block of practical RNA transcript assembly tools, but also of general path-covering problems.

2 Notation and preliminaries

Graphs. In this paper we consider directed graphs, especially directed acyclic graphs (DAGs). For a DAG $G = (N, A)$ with node set N and arc set A, we let $n = |N|$ and $m = |A|$. We denote an arc from u to v as uv and a path p through nodes $v_1, \ldots, v_k$ as $p = v_1 \ldots v_k$. Two paths are considered distinct if they differ in at least one node. The set of *in-neighbors* (*out-neighbors*) of a node v is denoted N_v^- (N_v^+), and its *indegree* (*outdegree*) as d_v^- (d_v^+). For nodes u, v we say that there is a *u-v path* if there is a path whose first node is u and last node is v (we also say that u *reaches* v). An *antichain* is a set of pairwise unreachable nodes. The size of the largest antichain in a DAG is called *width*. Analogously, for arcs uv, xy we say that uv reaches xy if v reaches x. An *arc-antichain* is a set of pairwise unreachable arcs, and the *arc-width* of a DAG is the size of the largest arc-antichain of the graph.

For $X \subseteq N$ we denote a *sequence* X through nodes $u_1, \ldots, u_\ell$ as $X = u_1, \ldots, u_\ell$ if there is a u_i-u_{i+1} path for all $1 \leq i \leq \ell - 1$; in particular, a path is a sequence. We say that X *covers* or *contains* a node v if $v = u_i$ for some $i \in \{1, \ldots, \ell\}$, and we write $u \in X$. If Y is a sequence and each node of X is contained in Y we say that X is a *subsequence* of Y, or that X is *contained* in Y. Since G is a DAG, the nodes of X appear in the same order as in Y.

[3] Note that the terms $x_{uvi} w_i$ are not linear, but can be linearized using a simple technique, see [17, 16].
[4] Note again that the terms $x_{uvi} w_i$ and $x_{uvi} \rho_i$ are not linear, but can be linearized using a simple technique, see [16].

Let $X' = v_1, v_2, \ldots, v_{\ell'}$ be a sequence such that there is a path from the last node u_ℓ of X to v_1. The *concatenation* of X and X' is simply the sequence XX' obtained as X followed by X'. If $u_\ell = v_1$, then the repeated occurrence of v_1 is removed, i.e. $XX' := u_1, \ldots, u_\ell, v_2, \ldots, v_{\ell'}$. We adopt the convention that lowercase letters u, v, w, x, y, z denote nodes and uppercase letters X, Y denote sequences of nodes.

Safety. We define a *path cover* of G to be a set P of s-t paths such that $\forall u \in N\ \exists p \in P : u \in p$; we also say *cover* instead of "s-t path cover of G" when the graph G is clear from the context. Two path covers are distinct if they differ in at least one path. We assume that every node of G lies in some s-t path. We say that a sequence X is *safe* for G if in every path cover P of G it holds that X is a subsequence of some path in P. If X is not safe we say that it is *unsafe*. A sequence X is *maximally safe* if it is not a subsequence of any other safe sequence. We assume that there is always an s-t path containing a given sequence of nodes, otherwise that sequence is vacuously unsafe. Note that imposing the collection of s-t paths of a cover to be minimal with respect to inclusion results in an equivalent definition of safety: any sequence that is safe for path covers is also safe for minimal path covers since every minimal path cover is a path cover; but also, if X is safe for minimal path covers then it is also safe for general path covers, as otherwise we could have a path cover P avoiding X from which arbitrary s-t paths can be removed until P becomes minimal.

Graph compression. We define a *unitary path* as a path where each node has indegree equal to one but the first one, and each node has outdegree equal to one but the last one. For example, in Figure 1, the paths *de* and *pq* are unitary, the path *cde* is maximally unitary. With respect to G, we define the *compressed graph of* G as the graph where every maximal unitary path of G is contracted into a single node. It is a folklore result that compressing a graph can be done in linear time (see e.g. [13, 52]). We state this fact in the next lemma together with an additional property stating that compressed graphs cannot have arcs uv with v the unique out-neighbor of u and u the unique in-neighbor of v, since uv would then be a unitary path.

▶ **Lemma 4.** *Let G be a directed graph with n nodes and m arcs. Then the following hold:*
1. *We can compute the compressed graph of G in $O(n + m)$ time.*
2. *In a compressed graph there is no arc uv such that $N_u^+ = \{v\}$ and $N_v^- = \{u\}$.*

The compression of unitary paths preserves safety: every s-t path in G traversing the unitary path traverses the corresponding node in the compressed graph of G, and every s-t path traversing a node in the compressed graph of G necessarily traverses the unitary path it corresponds to in G. This idea appeared in the context of walk-finding problems in [11].

Cutnodes and dominators. In the dominator literature, the directed graphs on which dominators are computed have a single start node s from which every node is reachable. In some cases, the graphs have instead or additionally a unique sink t which is reached by every node, in which case the dominance relation is referred to as post-dominance (see e.g., [5, 27]). A prominent application of dominators is in the context of program analysis, where s corresponds to the entry point of the program and t to its exit point. Our graphs always have unique abstract nodes s and t, hence, diverging from standard notation, we parameterize the domination relation by s and t. We borrow standard concepts and notation from the literature of dominators, which we describe next.

Let G be an s-t directed graph. A node u is said to *s-dominate* a node v if every s-v path contains u, hence u is an s-v cutnode if and only if u s-dominates v. Every node s- and t-dominates itself. We also say that v *t-dominates* u if every u-t path contains v, which is equivalent to v being a u-t cutnode. We say that u *strictly s-dominates* v if u s-dominates v and $u \neq v$. For $v \neq s$, the *immediate s-dominator* of v is the strict s-dominator of v that is dominated by all the nodes that strictly dominate v, in which case we write $idom_s(v) = u$. The domination relation on G with respect to s can be represented as a rooted tree at s with the same nodes as G, called the *s-dominator tree* of G. In this tree, the parent of every node is its immediate s-dominator, every node is s-dominated by all its ancestors and s-dominates all its descendants and itself. A node is said to be an *s-dominator* if it strictly s-dominates some node. The same terminology will be used for the t-domination relation. We also define a function $dom : N \times \mathbb{N}^+ \to N$ on rooted trees as follows. Let $dom_s(v, k) := idom_s(v)$ when $k = 1$ and $dom_s(v, k) := dom_s(dom_s(v, 1), k - 1)$ when $k > 1$. If k is larger than the number of *strict* dominators of a given node, then it defaults to s (resp. t) in the s-dominator tree (resp. t-dominator tree). Essentially, dom gives the k-th ancestor of a node in a rooted tree, if it is well defined, otherwise it returns the root of the tree. For example, in Figure 1, the immediate s-dominator of b is a ($dom_s(b, 1) = a$) and $dom_t(q, 3) = t$. Node c is a strict t-dominator of the nodes $\{a, b, k, k'\}$. See [23] for an in-depth presentation of dominators.

In DAGs, dominators admit a characterization, which we state in Lemma 5 below. This was first presented by Ochranová [41], but with an incomplete proof. This characterization was also discussed by Alstrupt et al. [6, Sec. 5.1]. To the best of our knowledge, we did not find a complete proof of this characterization in the literature, and hence one can be found in the extended version.

▶ **Lemma 5** (Characterization of node-dominators in DAGs). *Let $G = (N, A)$ be an s-t DAG. A node $u \in N$ is an s-dominator (resp. t-dominator) if and only if there is a node v such that $N_v^- = \{u\}$ (resp. $N_v^+ = \{u\}$).*

3 Safe sequences via dominators

3.1 Characterization of safe sequences

We start by proving Theorem 1.

Proof of Theorem 1. ($1 \Rightarrow 2$) If for every node $u \in N$ there is an s-t path covering u that avoids X, then we can build a path cover by considering for every node u of G an s-t path avoiding X that covers u. This contradicts the safety of X.

($2 \Rightarrow 3$) Since every s-t path covering u contains the sequence of s-u cutnodes followed by the sequence of u-t cutnodes and X is contained in every such path, the implication follows by taking $u = v$.

($3 \Rightarrow 1$) Since X is a subsequence of a sequence containing every s-v and v-t cutnode and v must be covered in every path cover by some s-t path, any of which contains the s- and t-cutnodes of v, it follows that X is safe. ◀

We will heavily use the type of sequence obtained as in point 3 of Theorem 1. As such, we define the *extension* of a node v, $\mathsf{extension}(v)$, as the sequence obtained from the concatenation of the sequence of s-v cutnodes with the sequence of v-t cutnodes. By Theorem 1 every maximal safe sequence is the extension of some node, and so in a DAG with n nodes and m arcs there are at most n maximal safe sequences. Moreover, for any node $v \in N$, $\mathsf{extension}(v)$ can be computed in time $O(m + n)$ by computing the s-v cutnodes

and v-t cutnodes with e.g. the simple algorithm of Cairo et al. [9]. Thus, Theorem 1 implies that we can compute in time $O(nm)$ a set of safe sequences containing every maximal safe sequence of G, by reporting $\mathsf{extension}(v)$ for every node v. However, the reported sequences may not be maximal and the same sequence may be reported multiple times.

Because of the correspondence between cutnodes and dominators explained in Section 2, the extension of a node can also be expressed as suitable paths in dominator trees. We will use this fact in the rest of the paper.

▶ Remark 6. In an s-t DAG G, $\mathsf{extension}(v)$ is the same as the path from s to v in the s-dominator tree of G concatenated with the path from v to t in the t-dominator tree of G.

3.2 Properties of s- and t-dominator trees

We start with a lemma showing an interplay between s-domination and t-domination on general s-t directed graphs.

▶ **Lemma 7.** *Let $G = (N, A)$ be an s-t directed graph, let $u, v \in N$ be nodes, and let $k \in \mathbb{N}^+$. If u is the k-th ancestor of v in the s-dominator tree, then $dom_t(u, k)$ t-dominates v, and v is not a t-dominator of u unless $v = dom_t(u, k)$.*

Proof. Note that $u \neq t$ since u is an s-dominator. Let $w = dom_t(u, k)$ and let $y_1, \ldots, y_{k-1}$ be the nodes between u and v in the s-dominator tree. We can assume $w \neq v$.

Observe that $w \neq y_i$ for all $i \in \{1, \ldots, k-1\}$. Suppose otherwise. Then $w \neq t$ because t is not an s-dominator. Hence, by definition of dom_t, there are $k-1$ nodes $x_1, \ldots, x_{k-1}$ between w and u in the t-dominator tree. Some x_j is not between u and w in the s-dominator tree, and hence there is a u-w path avoiding x_j. Further, since x_j is strictly t-dominated by w there is a w-t path avoiding x_j. Concatenating these paths at w results in a u-t path avoiding x_j, contradicting the fact that x_j t-dominates u.

Suppose now for a contradiction that v is not strictly t-dominated by w. From the point above, there is a u-v path avoiding w. This path concatenated with a v-t path avoiding w gives a u-t path also avoiding w, contradicting the fact that w t-dominates u.

It remains to show that v cannot t-dominate u. Suppose otherwise. Then we have $v = x_i$ for some $i \in \{1, \ldots, k-1\}$, since v is strictly t-dominated by w and t-dominates u. Thus, some y_j is not between v and u in the t-dominator tree, implying that there is a u-v path avoiding y_j. Since y_j is s-dominated by u, there is an s-u path avoiding y_j, which concatenated with the previous path gives an s-v path also avoiding y_j, contradicting the fact that y_j s-dominates v. ◀

Lemma 7 can be interpreted for the special case of $k = 1$ as follows: if u immediately s-dominates v then the immediate t-dominator of u t-dominates v (see Figure 2). Another consequence of this result is that if u is an s-dominator, then at most one node immediately s-dominated by u is in the path from t to u in the dominator tree of t, which must be the immediate t-dominator of u.

Observe now in Figure 1 that c is the immediate s-dominator only of d and that d is the immediate t-dominator only of c, and likewise for d and e, and so $\mathsf{extension}(c) = \mathsf{extension}(d) = \mathsf{extension}(e)$. This redundancy and asymmetry introduced by unitary paths is undesirable in the characterization of maximal safe sequences. This discussion motivates the following result.

▶ **Lemma 8.** *Let $G = (N, A)$ be a compressed s-t DAG and let $u, v \in N$ be distinct nodes. If u immediately s-dominates v and v immediately t-dominates u then u immediately s-dominates some node different from v and v immediately t-dominates some node different from u.*

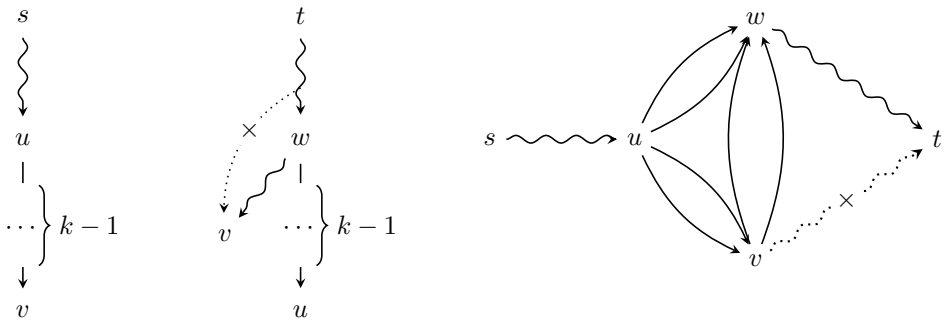


Figure 2 Illustration of the proof of Lemma 7. The two dominator trees discussed in the proof are shown on the left. On the right, an abstract graph for the special case of $k = 1$ is shown; node v cannot reach t without w, without contradicting the fact that $dom_t(u, 1) = w$.

Proof. Assume without loss of generality that u immediately s-dominates only node v.

We claim that $N_v^- = \{u\}$. Suppose otherwise. If N_v^- is empty then $v = s$, and so v is not a t-dominator, a contradiction. So there is an arc wv with $w \neq u$. Since u s-dominates v, it must also s-dominate every in-neighbour of v, namely w. But u immediately s-dominates only v, thus, v must s-dominate w, implying that there is a v-w path. Since $wv \in A$, G contains a cycle, a contradiction.

Since v is a t-dominator, by Lemma 5 there is a node whose unique out-neighbour is v. If this node is u, we contradict the fact that the graph is compressed by Lemma 4. Therefore, v must have at least two distinct in-neighbors, a contradiction. ◀

3.3 Characterization and enumeration of maximal safe sequences

Using Lemmas 7 and 8, we can give a precise characterization of maximal safe sequences.

▶ **Theorem 9** (Characterization of maximal safe sequences)**.** *Let $G = (N, A)$ be a compressed s-t DAG and let $u \in N$ be a node. Then* extension(u) *is a maximal safe sequence if and only if u is a leaf in both the s- and t-dominator trees of G.*

Proof. ($\Leftarrow$) If u is a leaf in both dominator trees then extension(u) is maximal: no other node can be added to this sequence by the equivalence of safe sequences and paths in the dominator trees given by Remark 6 and Theorem 1.

($\Rightarrow$) Suppose that extension(u) is a maximal safe sequence but u is not a leaf in both dominator trees. Without loss of generality suppose that u is not a leaf in the s-dominator tree. Note that $u \neq t$ since u is an s-dominator. Let v be a node immediately s-dominated by u and let w denote the immediate t-dominator of u.

First we argue that there is a node z that is immediately s-dominated by u and that is strictly t-dominated by w. We consider two cases. If $v \neq w$ then we can take $z = v$, because by Lemma 7 for $k = 1$ we can conclude that w t-dominates v, and since $v \neq w$, it strictly t-dominates v. If $v = w$ then we can take $z = v'$, since by Lemma 8 there is a node $v' \neq v$ that is immediately s-dominated by u; applying Lemma 7 to u and v' with $k = 1$, we conclude that w t-dominates v, and since $w = v \neq v$, w strictly t-dominates v'.

Now we show that extension(z) properly contains extension(u). Consider extension(u), which by definition consists of the sequence of s-u cutnodes (call it Y_1) concatenated with the sequence of u-t cutnodes (call it Y_2). In extension(u), by definition, the double occurrence

of u is removed, and thus let Y_2' be the sequence of w-t cutnodes. That is, $\mathsf{extension}(u)$ can be written as Y_1Y_2'. Consider also $\mathsf{extension}(z)$, which can be written analogously as X_1X_2'. Since u is the parent of z in the s-dominator tree, we have that Y_1 is *strictly* contained in X_1, and since z is a proper descendant of w in the t-dominator tree, we have that Y_2' is contained in X_2' (possibly $Y_2' = X_2'$). Therefore, $\mathsf{extension}(u)$ is a proper subsequence of $\mathsf{extension}(z)$, contradicting our assumption that $\mathsf{extension}(u)$ is a maximal safe sequence. ◀

The idea in the above theorem of targeting special nodes or arcs in order to characterize every maximal safe object is a recurring theme in safety problems. For instance, [31, 2] proposed the notion of *representative arc*, [48] proposed that of *core*, and [10] found a special class of walks, coined *macrotigs*, from where every maximal safe walk can be built. Each of these constructs was used in a different safety problem.

While the above characterization uses dominator trees, we can derive an equivalent characterization of dominator-tree leaves just in terms of in- and out-neighborhoods via Lemma 5, since a node is a leaf if and only if it is not a dominator. Computing the extension of every node satisfying such condition leads to a simple $O(mn)$ time algorithm outputting all and only maximal safe sequences, without needing to compute the dominator trees, from which Theorem 2 follows.

Since dominator trees can be built in $O(m+n)$ time (see, e.g., the algorithm by Alstrup et al. [6], Buchsbaum et al. [7]), or Fraczak et al. [20]), we can prove Theorem 3.

Proof of Theorem 3. Let G' be the compressed graph of G. Build the dominator trees of G' and compute all the leaves common to both trees. These three steps take $O(m+n)$ time.

1: For every node that is a leaf in both dominator trees, output $\mathsf{extension}(v)$. By Theorem 9 we find every maximal safe sequence during this process. Further, no duplicate safe sequence is computed since different leaves common to both trees have different extensions. Therefore the algorithm runs in time equal to the length of all the maximal safe sequences plus the time required to build two dominator trees over G', so $O(m+o)$ altogether.

2: With the equivalence given by Theorem 9, we have that every maximal safe sequence is the result of $\mathsf{extension}(v)$ for every node v of G' that is a leaf in both dominator trees. The sequence $\mathsf{extension}(v)$ is encoded in the dominator trees by Remark 6. ◀

4 Experiments

The experiments were performed on an AMD 32-core machine with 512GB RAM. The source code of our project is publicly available on Github[5]. All our algorithms are implemented in Python3, as are the scripts to produce the experimental results. To solve ILPs, we use Gurobi's Python API (version 11) under an academic license. For every graph, we run Gurobi with 4 threads and set up a timeout of 300 seconds in Gurobi's execution time.

Implementation. As described in Section 1.2, we study experimentally the potential of applying safe sequences in two arc-path covering problems, LeastSquares and MinPathError.

We did not use an external algorithm to compute arc-dominator trees because these are mostly tailored for node-dominators. Instead, we compute the immediate s-dominator and t-dominator of all arcs with an $O(m^2)$ time procedure via the algorithm of [9], appropriately handling the reversed graph of G for the immediate s-domination relation. The classical

[5] `https://github.com/algbio/safe-sequences`

algorithm of Aho and Ulman [3] to build dominator trees has the same running time as our procedure. We remark that even with a suboptimal algorithm to compute the dominator trees, our entire safety-preprocessing step takes less than 0.1 seconds on average.

With the dominator trees, we can compute all the maximal safe sequences with respect to arbitrary subsets $C \subseteq A$ of arcs to be covered, all in time proportional to the total length of all the maximal safe sequences, completely analogously to the discussion in Section 3.

The ILPs were optimized as in [25] by fixing x_{uvi} variables to 1 based on the maximal safe sequences of the input DAG, which reduces the search space of the linear program and hence speeds-up the solver. Let $X_1, \ldots, X_t$ be sequences in the input DAG such that: (a) each X_i, $i \in \{1, \ldots, t\}$, must appear in some solution path of the ILP, and (b) there exists no path in the DAG containing two distinct X_i and X_j. In the case that every X_i is a path, Grigorjew et al. [25] noticed that if the two mentioned properties hold, then $X_1, \ldots, X_t$ must appear in t distinct paths of among the k solution paths of the ILP (see the three solid-colored sequences shown in Figure 1 which must appear in different paths of any path cover). As such, without loss of generality, one can set $x_{uvi} = 1$ for each arc uv of X_i, for all $i \in \{1, \ldots, t\}$. When using sequences to set binary variables, we can proceed in a completely analogous manner. To have a large number of binary variables set to 1, we follow the approach of [25]. Namely, for every arc uv, we define its weight as the length of the longest safe sequence containing uv. Then we find a maximum-weight antichain of arcs in the DAG using the min-flow reduction of [39]. Then, for each a_i in the maximum-weight antichain, we consider a longest safe sequence X_i containing a_i to fix variables to 1. Note that a path covering two such sequences X_i and X_j would witness that a_i and a_j cannot be in an antichain. Finally, we set k to the arc-width of the graph and run Gurobi to solve the ILP.

Note that any solution to MinPathError is also a path cover, since each arc weight is positive, and thus the simplified ILP has equivalent optimal solutions. This may not be the case for LeastSquares, since some arcs may not be covered by any solution path. To handle such scenarios we propose using maximal safe sequences for path covers where only a subset C of nodes or arcs can be trusted to appear in any optimal solution. As an illustration of this approach, we perform experiments by choosing the subset C as those arcs whose weight is at least as large as the 25-th percentile of the weights of all arcs, since we can heuristically infer that the solution paths of LeastSquares cover the arcs of large enough weight.

Datasets. We experiment with seven datasets. The first four contain splice graphs with erroneous weights created directly from gene annotation by [16].[6] These were created starting from the well-known dataset of Shao and Kingsford [49], which was also used in several other studies benchmarking the speed of minimum flow decomposition solvers [17, 32, 54]. The original splice graphs were created in [49] from gene annotation from **Human**, **Mouse** and **Zebrafish**. This dataset also contains a set of graphs (**SRR020730**) from human gene annotation, but with flow values coming from a real sequencing sample from the Sequence Read Archive (SRR020730), quantified using the tool Salmon [42]. To mimic splice graphs constructed from real read abundances, [16] added noise to the flow value of each arc, according to a Poisson distribution. The last three datasets contain graphs constructed by a state-of-the-art tool for RNA transcript assembly, IsoQuant [44]. Mouse annotated transcript expression was simulated according to a Gamma distribution that resembles real expression profile [44]. From these expressed transcripts, PacBio reads were simulated using

[6] Available at `https://doi.org/10.5281/zenodo.10775004`

IsoSeqSim [1] and fed to IsoQuant for graph creation (we call this **Mouse PacBio**), and ONT reads were simulated using Trans-Nanosim [28], which was modified to perform a more realistic read truncation (see [44]), and fed to IsoQuant for graph creation (we call this **Mouse ONT**). For the latter one, there is also a simpler version where no read truncation was introduced (**Mouse ONT.tr**). These are available at [43].

Discussion. We group the graphs in each dataset by width (column "w"). For each width-interval we show different metrics, namely the number of graphs "#g", the average number of arcs "avg m" (and the maximum number of arcs in parenthesis), the time taken in seconds by the safety preprocessing step in "prep", the percentage of fixed arc-variables x_{uvi} of the ILP out of all mk variables in "vars", the number of solved instances by the ILP with and without safety in "#solved" (and in parenthesis the average time taken in seconds to solve those instances), and in the column "×" we show the average speedup obtained by advising the linear program with safety. The speed-ups are computed by considering for each graph the ratio between the time taken by the original ILP (or 300 sec. if this timed-out) divided by the time taken by the optimized ILP, and averaging these ratios. In the "prep" and "vars" column, a dash is written only if no instance was solved with safety information. In the remaining columns a dash means not applicable.

In Table 1 we show the experimental results of the safety-optimized ILPs for MinPathError.(As explained before, for this problem it is correct to select $C = A$.) When the width of the graph is small, the speed-ups are also small. However, such graphs are fast to solve without safety in the first place. The challenging cases are the graphs of larger widths and more arcs (indeed, the solver with no safety information performs poorly and it becomes slower as the width increases). The speed-up of the solver significantly increases as the width of the graph increases, becoming more than 200× in some datasets. Moreover, these speed-up are obtained almost for free, in the sense that the time needed to compute safe sequences is negligible for all datasets, i.e. below 0.1 seconds, and often even much faster. We highlight the notable difference between the number of instances solved with no safety and with safety. In Table 2 for LeastSquares we observe the same phenomena, obtaining in some datasets speed-ups of 300× on graphs of larger widths, and solving always more instances with safety than without safety. In the extended version results for $C \subsetneq A$ in the MinPathError and for $C = A$ in the LeastSquares can be found.

References

1 Isoseqsim. URL: `https://github.com/yunhaowang/IsoSeqSim` [cited April 23, 2025].

2 Bashar Ahmed, Siddharth Singh Rana, Shahbaz Khan, et al. Evaluating Optimal Safe Flows Decomposition for RNA Assembly. *arXiv preprint arXiv:2409.13279*, 2024.

3 Alfred V Aho and Jeffrey D Ullman. *The theory of parsing, translation, and compiling*, volume 1. Prentice-Hall Englewood Cliffs, NJ, 1973.

4 Ravindra K Ahuja, Thomas L Magnanti, James B Orlin, et al. *Network flows: theory, algorithms, and applications*, volume 1. Prentice hall Englewood Cliffs, NJ, 1993.

5 Frances E Allen. Control flow analysis. *ACM Sigplan Notices*, 5(7):1–19, 1970. `doi:10.1145/800028.808479`.

6 Stephen Alstrup, Dov Harel, Peter W Lauridsen, and Mikkel Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–2132, 1999. `doi:10.1137/S0097539797317263`.

7 Adam L Buchsbaum, Loukas Georgiadis, Haim Kaplan, Anne Rogers, Robert E Tarjan, and Jeffery R Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008. `doi:10.1137/070693217`.

Table 1 Experimental results of MinPathError. Safety is with respect to path covers of $C = A$.

	w	#g	avg m (max m)	prep (s)	vars (%)	#solved (avg time (s))		×
						no safety	safety	
Zebrafish	1-3	15405	14 (210)	0.003	87.5	15405 (0.011)	15405 (0.011)	1.2
	4-6	239	35 (156)	0.007	29.5	239 (1.927)	239 (0.095)	14.6
	7-9	6	70 (158)	0.015	15.3	4 (71.070)	6 (0.645)	244.3
	10+	1	81 (81)	0.016	13.1	0 (-)	1 (3.421)	88.1
Human	1-3	10729	15 (331)	0.003	78.9	10729 (0.020)	10729 (0.014)	1.4
	4-6	947	35 (163)	0.008	24.7	944 (4.031)	947 (0.147)	19.0
	7-9	92	54 (105)	0.012	14.1	63 (63.346)	92 (3.924)	116.6
	10+	12	80 (104)	0.019	11.7	2 (189.233)	9 (32.264)	35.0
Mouse	1-3	12280	14 (128)	0.003	84.4	12280 (0.014)	12280 (0.011)	1.3
	4-6	749	35 (126)	0.007	27.4	749 (2.295)	749 (0.121)	15.0
	7-9	60	58 (231)	0.015	17.5	43 (52.717)	60 (4.955)	73.3
	10+	9	79 (121)	0.014	15.2	1 (262.851)	7 (34.163)	69.2
SRR020730	1-3	35069	9 (186)	0.002	84.7	35069 (0.015)	35069 (0.010)	1.4
	4-6	4497	33 (171)	0.007	20.4	4496 (2.043)	4497 (0.124)	12.7
	7-9	1008	52 (131)	0.010	11.9	844 (42.962)	1007 (1.692)	117.1
	10+	296	75 (184)	0.015	7.9	80 (63.971)	289 (22.945)	142.5
Mouse ONT	1-3	18527	11 (129)	0.003	81.4	18527 (0.013)	18527 (0.011)	1.3
	4-6	3083	31 (387)	0.007	30.2	3083 (0.203)	3083 (0.041)	4.8
	7-9	762	48 (157)	0.010	18.1	755 (2.495)	762 (0.180)	38.7
	10+	442	87 (589)	0.021	11.1	372 (18.194)	437 (0.611)	266.0
Mouse ONT.tr	1-3	18029	12 (131)	0.003	81.1	18029 (0.011)	18029 (0.011)	1.1
	4-6	3028	32 (187)	0.007	30.6	3028 (0.187)	3028 (0.042)	3.8
	7-9	803	50 (419)	0.012	19.0	796 (2.344)	803 (0.182)	21.9
	10+	476	89 (549)	0.022	11.5	403 (17.204)	472 (0.982)	162.1
Mouse PacBio	1-3	14256	14 (382)	0.003	83.5	14256 (0.010)	14256 (0.011)	1.2
	4-6	1376	33 (187)	0.006	33.3	1376 (0.196)	1376 (0.040)	5.2
	7-9	182	49 (159)	0.009	20.5	181 (4.083)	182 (0.144)	48.1
	10+	63	109 (1178)	0.026	13.3	53 (22.188)	62 (2.557)	210.3

8 Benjamin Merlin Bumpus, Bart M. P. Jansen, and Jari J. H. de Kroon. Search-space reduction via essential vertices. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 30:1–30:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.30.

9 Massimo Cairo, Shahbaz Khan, Romeo Rizzi, Sebastian Schmidt, Alexandru I. Tomescu, and Elia C. Zirondelli. A simplified algorithm computing all s-t bridges and articulation points. *Discrete Applied Mathematics*, 305:103–108, 2021. doi:10.1016/j.dam.2021.08.026.

10 Massimo Cairo, Romeo Rizzi, Alexandru I. Tomescu, and Elia C. Zirondelli. Genome Assembly, from Practice to Theory: Safe, Complete and Linear-Time. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.43.

Table 2 Experimental results of LeastSquares. Safety is with respect to path covers of the subset C of arcs whose weight is at least as large as the 25-th percentile of the weights of all arcs.

	w	#g	avg m (max m)	prep (s)	vars (%)	#solved (avg time (s))		×
						no safety	safety	
Zebrafish	1-3	15405	14 (210)	0.003	83.1	15405 (0.031)	15405 (0.023)	1.5
	4-6	239	35 (156)	0.007	18.1	224 (13.295)	238 (3.061)	15.9
	7-9	6	70 (158)	0.011	8.4	0 (-)	2 (62.840)	4.8
	10+	1	81 (81)	-	-	0 (-)	0 (-)	-
Human	1-3	10729	15 (331)	0.003	74.6	10729 (0.046)	10729 (0.031)	1.8
	4-6	947	35 (163)	0.007	16.3	862 (17.153)	930 (6.334)	12.9
	7-9	92	54 (105)	0.011	9.6	4 (69.195)	53 (71.207)	29.1
	10+	12	80 (104)	-	-	0 (-)	0 (-)	-
Mouse	1-3	12280	14 (128)	0.003	80.0	12280 (0.035)	12280 (0.026)	1.6
	4-6	749	35 (126)	0.007	16.9	696 (14.851)	744 (5.917)	12.4
	7-9	60	58 (231)	0.015	9.9	0 (-)	16 (64.253)	121.9
	10+	9	79 (121)	-	-	0 (-)	0 (-)	-
Mouse SRR020730	1-3	35069	9 (186)	0.002	83.2	35069 (0.026)	35069 (0.017)	1.7
	4-6	4497	33 (171)	0.007	15.5	4428 (13.082)	4494 (1.275)	16.7
	7-9	1008	52 (131)	0.011	8.9	269 (127.595)	899 (25.515)	82.7
	10+	296	75 (184)	0.015	6.8	1 (7.363)	104 (61.880)	68.0
Mouse ONT	1-3	18527	11 (129)	0.003	75.9	18527 (0.020)	18527 (0.017)	1.6
	4-6	3083	31 (387)	0.007	22.5	3077 (1.448)	3082 (0.310)	7.6
	7-9	762	48 (157)	0.010	13.5	693 (15.761)	754 (1.602)	116.2
	10+	442	87 (589)	0.018	8.5	222 (37.135)	405 (11.143)	301.0
Mouse ONT.tr	1-3	18029	12 (131)	0.003	74.7	18029 (0.020)	18029 (0.017)	1.5
	4-6	3028	32 (187)	0.007	21.7	3012 (1.693)	3022 (0.426)	7.1
	7-9	803	50 (419)	0.011	13.5	727 (14.958)	794 (2.087)	76.2
	10+	476	89 (549)	0.021	8.8	251 (32.631)	423 (8.422)	183.2
Mouse PacBio	1-3	14256	14 (382)	0.003	74.4	14256 (0.034)	14256 (0.025)	1.7
	4-6	1376	33 (187)	0.007	22.1	1356 (2.622)	1372 (1.189)	12.3
	7-9	182	49 (159)	0.010	14.0	146 (37.428)	174 (6.627)	137.8
	10+	63	109 (1178)	0.015	10.3	15 (110.413)	46 (17.775)	370.6

11 Massimo Cairo, Romeo Rizzi, Alexandru I. Tomescu, and Elia C. Zirondelli. Genome assembly, from practice to theory: Safe, complete and *Linear-Time*. *ACM Trans. Algorithms*, 20(1):4:1–4:26, 2024. doi:10.1145/3632176.

12 Stefan Canzar, Sandro Andreotti, David Weese, Knut Reinert, and Gunnar W Klau. CIDANE: comprehensive isoform discovery and abundance estimation. *Genome Biology*, 17:1–18, 2016.

13 Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.

14 Keith D Cooper, Timothy J Harvey, and Ken Kennedy. A simple, fast dominance algorithm. *Software Practice & Experience*, 4(1-10):1–8, 2001.

15 Fernando H C Dias, Manuel Cáceres, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. A safety framework for flow decomposition problems via integer linear programming. *Bioinformatics*, 39(11):btad640, October 2023. doi:10.1093/bioinformatics/btad640.

16 Fernando H. C. Dias and Alexandru I. Tomescu. Accurate flow decomposition via robust integer linear programming. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–14, 2024. doi:10.1109/TCBB.2024.3433523.

17 Fernando HC Dias, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. Fast, flexible, and exact minimum flow decompositions via ILP. In *RECOMB 2022 – International Conference on Research in Computational Molecular Biology*, pages 230–245. Springer, 2022.

18 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? *CoRR*, abs/1811.09429, 2018. arXiv:1811.09429.

19 Jianxing Feng, Wei Li, and Tao Jiang. Inference of isoforms from short sequence reads. *Journal of Computational Biology*, 18(3):305–321, 2011. doi:10.1089/CMB.2010.0243.

20 Wojciech Fraczak, Loukas Georgiadis, Andrew Miller, and Robert E Tarjan. Finding dominators via disjoint set union. *Journal of Discrete Algorithms*, 23:2–20, 2013. doi:10.1016/J.JDA.2013.10.003.

21 Thomas Gatter and Peter F Stadler. Ryūtō: network-flow based transcriptome reconstruction. *BMC bioinformatics*, 20:1–14, 2019. doi:10.1186/S12859-019-2786-5.

22 Loukas Georgiadis, Luigi Laura, Nikos Parotsidis, and Robert E Tarjan. Loop nesting forests, dominators, and applications. In *Experimental Algorithms: 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29–July 1, 2014. Proceedings 13*, pages 174–186. Springer, 2014.

23 Loukas Georgiadis and Nikos Parotsidis. Dominators in directed graphs: a survey of recent results, applications, and open problems. *Proc. of ISCIM*, 13:15–20, 2013.

24 Loukas Georgiadis, Robert Tarjan, and Renato Werneck. Finding dominators in practice. *Journal of Graph Algorithms and Applications*, 10(1):69–94, 2006. doi:10.7155/JGAA.00119.

25 Andreas Grigorjew, Fernando H. C. Dias, Andrea Cracco, Romeo Rizzi, and Alexandru I. Tomescu. Accelerating ILP solvers for minimum flow decompositions through search space and dimensionality reductions. In Leo Liberti, editor, *22nd International Symposium on Experimental Algorithms, SEA 2024, July 23-26, 2024, Vienna, Austria*, volume 301 of *LIPIcs*, pages 14:1–14:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SEA.2024.14.

26 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007. doi:10.1145/1233481.1233493.

27 Rajiv Gupta. Generalized dominators and post-dominators. In *Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 246–257, 1992. doi:10.1145/143165.143216.

28 Saber Hafezqorani, Chen Yang, Theodora Lo, Ka Ming Nip, René L Warren, and Inanc Birol. Trans-NanoSim characterizes and simulates nanopore RNA-sequencing data. *GigaScience*, 9(6):giaa061, June 2020. doi:10.1093/gigascience/giaa061.

29 Bart M. P. Jansen and Ruben F. A. Verhaegh. Search-space reduction via essential vertices revisited: Vertex multicut and cograph deletion. In Hans L. Bodlaender, editor, *19th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2024, June 12-14, 2024, Helsinki, Finland*, volume 294 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SWAT.2024.28.

30 Shahbaz Khan, Milla Kortelainen, Manuel Cáceres, Lucia Williams, and Alexandru I. Tomescu. Improving RNA assembly via safety and completeness in flow decompositions. *J. Comput. Biol.*, 29(12):1270–1287, 2022. doi:10.1089/CMB.2022.0261.

31 Shahbaz Khan and Alexandru I. Tomescu. Optimizing Safe Flow Decompositions in DAGs. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 72:1–72:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.ESA.2022.72.

32 Kyle Kloster, Philipp Kuinke, Michael P O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. A practical FPT algorithm for flow decomposition and transcript assembly. In *ALENEX 2018 – Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments*, pages 75–86. SIAM, 2018. doi:10.1137/1.9781611975055.7.

33 Jingyi Jessica Li, Ci-Ren Jiang, James B Brown, Haiyan Huang, and Peter J Bickel. Sparse linear modeling of next-generation mRNA sequencing (RNA-Seq) data for isoform discovery and abundance estimation. *Proceedings of the National Academy of Sciences*, 108(50):19867–19872, 2011.

34 Wei Li, Jianxing Feng, and Tao Jiang. IsoLasso: a LASSO regression approach to RNA-Seq based transcriptome assembly. In *RECOMB 2011 – International Conference on Research in Computational Molecular Biology*, pages 168–188. Springer, 2011.

35 Yen-Yi Lin, Phuong Dao, Faraz Hach, Marzieh Bakhshi, Fan Mo, Anna Lapuk, Colin Collins, and S Cenk Sahinalp. CLIIQ: Accurate comparative detection and quantification of expressed isoforms in a population. In *WABI 2012 – 12th International Workshop on Algorithms in Bioinformatics*, pages 178–189. Springer, 2012.

36 Juntao Liu, Ting Yu, Zengchao Mu, and Guojun Li. TransLiG: a de novo transcriptome assembler that uses line graph iteration. *Genome Biology*, 20:1–9, 2019.

37 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization - preprocessing with a guarantee. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 129–161. Springer, 2012. doi:10.1007/978-3-642-30891-8_10.

38 Aziz M Mezlini, Eric JM Smith, Marc Fiume, Orion Buske, Gleb L Savich, Sohrab Shah, Sam Aparicio, Derek Y Chiang, Anna Goldenberg, and Michael Brudno. iReckon: simultaneous isoform discovery and abundance estimation from RNA-seq data. *Genome Research*, 23(3):519–529, 2013.

39 Ralf Möhring. Algorithmic Aspects of Comparability Graphs and Interval Graphs. In Ivan Rival, editor, *Graphs and Order: the role of graphs in the theory of ordered sets and its applications*. D. Reidel Publishing Company, 1984.

40 Nidia Obscura Acosta, Veli Mäkinen, and Alexandru I Tomescu. A safe and complete algorithm for metagenomic assembly. *Algorithms for Molecular Biology*, 13:1–12, 2018. doi:10.1186/S13015-018-0122-7.

41 Renata Ochranová. Finding dominators. In *Foundations of Computation Theory: Proceedings of the 1983 International FCT-Conference Borgholm, Sweden, August 21–27, 1983 4*, pages 328–334. Springer, 1983. doi:10.1007/3-540-12689-9_115.

42 Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419, 2017. doi:10.1038/nmeth.4197.

43 Andrey Prjibelski. IsoQuant graphs from PacBio and ONT Mouse sequencing data, October 2024. doi:10.5281/zenodo.13987688.

44 Andrey D. Prjibelski, Alla Mikheenko, Anoushka Joglekar, Alexander Smetanin, Julien Jarroux, Alla L. Lapidus, and Hagen U. Tilgner. Accurate isoform discovery with IsoQuant using long reads. *Nature Biotechnology*, 41(7):915–918, 2023. doi:10.1038/s41587-022-01565-y.

45 Amatur Rahman and Paul Medvedev. Assembler artifacts include misassembly because of unsafe unitigs and underassembly because of bidirected graphs. *Genome Research*, 32(9):1746–1753, 2022.

46 Zhaleh Safikhani, Mehdi Sadeghi, Hamid Pezeshk, and Changiz Eslahchi. SSP: An interval integer linear programming for de novo transcriptome assembly and isoform discovery of RNA-seq reads. *Genomics*, 102(5-6):507–514, 2013.

47 Palash Sashittal, Chuanyi Zhang, Jian Peng, and Mohammed El-Kebir. Jumper enables discontinuous transcript assembly in coronaviruses. *Nature Communications*, 12(1):6728, 2021.

48 Sebastian Schmidt, Santeri Toivonen, Paul Medvedev, and Alexandru I. Tomescu. Applying the Safe-And-Complete Framework to Practical Genome Assembly. In Solon P. Pissis and Wing-Kin Sung, editors, *24th International Workshop on Algorithms in Bioinformatics (WABI 2024)*, volume 312 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.WABI.2024.8.

49 Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(2):658–670, 2017.

50 Li Song, Sarven Sabunciyan, and Liliana Florea. CLASS2: accurate and efficient splice variant annotation from RNA-seq reads. *Nucleic Acids Research*, 44(10):e98–e98, 2016.

51 Alexandru I. Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. *BMC Bioinformatics*, 14(5):S15, 2013. doi:10.1186/1471-2105-14-S5-S15.

52 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly via omnitigs. In Mona Singh, editor, *Research in Computational Molecular Biology*, pages 152–163, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-31957-5_11.

53 Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008. doi:10.1016/J.EJOR.2006.05.043.

54 Lucia Williams, Alexandru I. Tomescu, and Brendan Mumey. Flow decomposition with subpath constraints. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 20(1):360–370, 2023. doi:10.1109/TCBB.2022.3147697.

55 Zheng Xia, Jianguo Wen, Chung-Che Chang, and Xiaobo Zhou. NSMAP: a method for spliced isoforms identification and quantification from RNA-Seq. *BMC Bioinformatics*, 12:1–13, 2011.

56 Jin Zhao, Haodi Feng, Daming Zhu, and Yu Lin. MultiTrans: an algorithm for path extraction through Mixed Integer Linear Programming for transcriptome assembly. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19(1):48–56, 2021. doi:10.1109/TCBB.2021.3083277.

57 Hongyu Zheng, Cong Ma, and Carl Kingsford. Deriving ranges of optimal estimated transcript expression due to nonidentifiability. *Journal of Computational Biology*, 29(2):121–139, 2022. Proceedings paper of RECOMB 2021. doi:10.1089/CMB.2021.0444.

PLS-Completeness of String Permutations

Dominik Scheder
TU Chemnitz, Germany

Johannes Tantow
TU Chemnitz, Germany

Abstract

Bitstrings can be permuted via permutations and compared via the lexicographic order. In this paper we study the complexity of finding a minimum of a bitstring via given permutations. As finding a global optimum is known to be NP-complete [1], we study the local optima via the class PLS[8] and show hardness for PLS. Additionally, we show that even for one permutation the global optimization problem is NP-complete and give a formula that has these permutation as its symmetries. This answers an open question inspired from Kołodziejczyk and Thapen [9] and stated at the *SAT and interactions* seminar in Dagstuhl [14].

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases PLS, total search problems, local search, permutation groups, symmetry

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.56

Related Version *Full Version*: https://arxiv.org/abs/2505.02622

Acknowledgements We want to thank Neil Thapen for introducing the problem to us.

1 Introduction

Given a string $\mathbf{x} \in \{0,1\}^n$ and a couple of permutations in S_n, we can apply a permutation to $\mathbf{x}$ and obtain a new bit string. What is the lexicographically smallest string we can obtain this way? This problem is known [1] to be NP-hard. What about finding a local minimum, i.e., arriving at a bit string that cannot be further improved by a single application of a permutation? In this paper, we show that this problem is PLS-complete.
To be more formal, we are given a string $\mathbf{x} \in \{0,1\}^n$ and permutations $\pi_1, \dots, \pi_m$ on the set $[n]$. When we view $\mathbf{x}$ as a function $[n] \to \{0,1\}$, the notation $\mathbf{x} \circ \pi$ makes sense and is the string obtained from $\mathbf{x}$ by permuting the coordinates according to π. By $\langle \pi_1, \dots, \pi_m \rangle$ we denote the subgroup of $S_{[n]}$ generated by the π_i. The problem k-Permutation Global Orbit Minimum asks for the $\pi \in \langle \pi_1, \dots, \pi_k \rangle$ such that $\mathbf{x} \circ \pi$ is lexicographically minimal. Babai and Luks [1] showed that this is NP-hard even for $k = 2$. In fact, we will see that it is NP-hard even for $k = 1$, i.e., a single permutation.

k-Permutation Local Orbit Minimum asks for a local minimum. That is, an element $\pi \in \langle \pi_1, \dots, \pi_k \rangle$ such that

$$\mathbf{x} \circ \pi \preceq_{\text{lex}} \mathbf{x} \circ \pi \circ \pi_i \quad \text{for all } 1 \leq i \leq k \ ,$$

i.e., a single application of a permutation π_i cannot further improve the string $\mathbf{x} \circ \pi$.
A local optimum always exists and hence this is an instance of a total search problem. Total search problems where solutions are recognizable in polynomial time form the class TFNP. Total search problems that can be stated as finding a local optimum with respect to a certain *cost function* and a *neighborhood relation* constitute the subclass PLS (polynomial local search). Known hard problems for PLS include finding a pure Nash-equilibrium in a congestion game[4] or finding a locally optimal max cut (LocalMaxCut) [13]. Almost all known PLS-complete problems require quite involved cost functions. Our problem

© Dominik Scheder and Johannes Tantow;
licensed under Creative Commons License CC-BY 4.0

k-Permutation Local Orbit Minimum has the benefit of using the possibly simplest cost function - the lexicographic ordering. The only other PLS-complete problem using a lexicographic cost function that we know of is FLIP, which asks to minimize the m-bit output of a circuit C, where the solutions are all n-bit inputs and the neighborhood relation is defined by flipping a single bit.

Thus, our result unifies two desirable properties - our PLS-complete problem is very combinatorial in nature (in contrast to FLIP) and uses a very simple cost function (in contrast to LocalMaxCut)

1.1 SAT Solving and Symmetry Breaking

When encoding a combinatorial problem as a CNF formula F (think of "is there a k-Ramsey graph on n vertices?"), the formula will often contain many symmetries. To make the problem easier for SAT solvers, one can take the statement

> *The satisfying assignment α should be a local lexicographical minimum with respect to those symmetries,*

encode it as a CNF formula G and feed $F \wedge G$ to the SAT solver. Clearly, $F \wedge G$ is satisfiable if and only if F is. In case that $F \wedge G$ is unsatisfiable, SAT solvers are often expected to produce a proof of unsatisfiability. A popular proof system used in this context is DRAT [15]. However, it is not known to what extend DRAT can handle symmetry breaking [7], that is, whether a short DRAT-refutation of $F \wedge G$ can be transformed into a short DRAT-refutation of F. In this context, Thapen [14] asked whether there exists a polynomial algorithm that, given a CNF formula F, a handful of symmetries thereof, and a satisfying assignment α, finds a satisfying assignment β that is a local lexicographical minimum with respect to those symmetries. In this paper, we show that this problem is PLS-complete, which is evidence that such a polynomial time algorithm might not exist.

2 Preliminaries

2.1 Total search problems and PLS

The class FNP is the functional correspondent of NP. TFNP $\subset$ FNP is the subset of *total* search problems, i.e., problems that always have a solution. As this is a semantic class, TFNP has no known complete problems, and thus it is usually studied via its subclasses. These subclasses are based on the combinatorial principle that proves the existence of a solution. These principles include the existence of sinks in directed acyclic graphs (PLS)[8], the parity argument for directed and undirected graphs (PPAD, PPA) or the pigeonhole principle (PPP) (all introduced in [11]). Nonetheless, not all problems in TFNP can be categorized in one of the known subclasses, Factoring being a prime example.

By the above characterization, PLS requires finding a sink of a directed acyclic graph G. This would be possible in polynomial time if G was given explicitly. Instead, G is always given implicitly via a circuit that computes the successor list of a given node. To make sure that G is acyclic, we have a second circuit computing a topological ordering, that is, a "cost" function that is strictly decreasing along the edges. A solution to the problem is a sink of G or an edge (u, v) with $\text{cost}(u) \leq \text{cost}(v)$, i.e., violating the decreasing cost condition. This guarantees the totality of the problem.

An alternative definition is the following. For a PLS problem P we have a set of instances I. Each instance $i \in I$ has a set of feasible solutions S (e.g. for the Euclidean traveling salesman problem the solutions are exactly the Hamilton cycles of K_n). Additionally, we require the following polynomial-time computable algorithms (usually given as circuits):

1. an algorithm that decides whether a given s is a feasible solution.
2. an algorithm that computes a starting solution $s \in S$
3. an algorithm that computes for a feasible solution s the neighborhood $N(s)$
4. an algorithm that computes for a solution s the cost $\text{cost}(s)$

A feasible solution s is a *local minimum* if $\text{cost}(s) \leq \text{cost}(s')$ for all $s' \in N(s)$. The definition is given for a minimization problem, but can be also defined in terms of a maximization problem.

We can easily transform this into a directed acyclic graph by keeping only those neighbors in $N(s)$ having strictly smaller cost–or even keeping only the one neighbor $s' \in N(s)$ of minimal cost (breaking ties arbitrarily).

Similar to NP, there is also a hardness structure in PLS. Let P and Q be two problems in PLS. P reduces to Q via a *PLS-reduction* (f, g) for functions f and g such that f maps an instance I from P to an instance $f(I)$ of Q and g maps a solution s of $f(I)$ to a solution $g(I, s)$ of P so that if s is a local minimum in $f(I)$ then also $g(I, s)$ is a local minimum in I. This was defined in [8] and the first natural PLS-complete problem is FLIP. There solutions are n-bit strings, the cost is calculated by a given circuit C and the neighborhood are all n-bit strings with a Hamming distance of 1.

The obvious greedy algorithm to find a solution for a PLS-problem is as follows: Use the given algorithms to compute the start solution and always select the best neighbor until there is no better solution. This solution is called the *standard solution* and the algorithm the *standard algorithm*. For the problem FLIP, finding the standard solution for a given start solution is PSPACE-complete[12, Lemma 4].

Reductions that preserve the PSPACE-completeness are called tight[13]. For this, we consider the transition graph $TG(I)$ of an instance I of the problem P, that has a directed edge from each feasible solution x to all of its neighbors $N(x)$.

A PLS-reduction (f, g) from P to Q is called *tight* if for every instance I of P there exists a set $\mathcal{R}$ of feasible solutions for $f(I)$ such that

1. $\mathcal{R}$ contains all local optima of $f(I)$
2. For every solution s of I it is possible to construct in polynomial time a feasible solution $t \in \mathcal{R}$ such that $g(I, t) = s$
3. If the transition graph of $TG(f(I))$ contains a path from q to q' such that both q and q' are in $\mathcal{R}$ and all other intermediate nodes are not in $\mathcal{R}$, let $p = g(I, q)$ and $p' = g(I, q')$ be the corresponding solutions in P. Then either $p = p'$ or there is an arc from p to p' in $TG(I)$.

An interesting subclass of PLS is CLS that is supposed to capture continuous local search problems. Recently, it was shown that CLS = PLS ∩ PPAD[5].

2.2 Permutation groups

A permutation of a set V is a bijection $\pi : V \to V$. For permutations $\pi_1, \ldots, \pi_k$, we denote by $\langle \pi_1, \ldots, \pi_k \rangle$ the subgroup of S_V generated by the π_i. Checking membership $\pi \in \langle \pi_1, \ldots, \pi_k \rangle$ is non-trivial but can be done in polynomial time [6, Section 1].

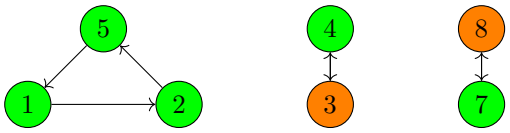


Figure 1 An example of the scenario for one permutation π.

3 Related Work

Many problems are known to be PLS-complete, whose reductions mostly start from FLIP. An influential reduction technique was used by Krentel [10] to show that finding a local minimum of a weighted CNF-formula is PLS-complete. The idea is to have multiple copies of the circuit, so called *test circuits* that precompute the effects of a flip. This idea is pushed further in [13] in order to show that finding a local minimum for weighted positive NAE(not all equal) 3-SAT is PLS-complete, and this is used to show that other problems as finding a LocalMaxCut or finding a stable configurations of Hopfield networks are PLS-complete.

Other direct reductions from FLIP are used in [4] to show that finding a pure Nash equilibrium in an asymmetric network congestion game is PLS-complete and in [3] to show that maximum constraint assignment, a generalization of CNF-SAT, is PLS-complete. These reductions are especially of interest to us as FLIP is to the best of our knowledge the only problem in PLS with lexicographical weights and hence needed for our reduction.

Numerous optimization problems on permutation groups given via generators $\pi_1, \ldots, \pi_k$ are studied in [2]. These include finding a $\pi \in \langle \pi_1, \ldots, \pi_k \rangle$ that minimizes $\Sigma_{i \in V} c(i, \pi(i))$ for some cost function $c : V^2 \to \mathbb{R}$. All these problems are shown to be NP-complete via a reduction from finding a fixed-point free permutation, which is shown to be NP-complete.

Our problem has previously been studied in [1], but there the interest was in *global* optima. This was proven to be NP-hard even for abelian groups.

4 The one permutation problem

We start our investigation by studying the special case that $k = 1$, i.e., we have only one generator π_1 and our subgroup is $\langle \pi_1 \rangle$. In this case one can efficiently find a local optimum[9, Section 8.2]. We show this with a different algorithm again. In contrast, we will show that surprisingly finding a global optimum is NP-complete. To the best of our knowledge, this has only been known for *two* permutations [1].

4.1 Finding a local optimum

The problem is efficiently solvable when $k = 1$, i.e., we are only given a single permutation π. We describe a different way of solving it in contrast to [9]. We can transform π into the cycle notation and annotate each element with the bit that it is mapped to by the string $\mathbf{x} \in \{0, 1\}^n$. We call a cycle *interesting* if $\mathbf{x}$ is non-constant on it. Additionally, we identify each cycle with its smallest member.

Consider the permutation (1 2 5)(3 4)(7 8) and the string 001001 depicted in Figure 1. We color an element green if it is mapped to zero and orange if it is mapped to one by the string. The left cycle is the cycle of 1 and not interesting whereas the other two are which are identified by 3 and 7.

The permutation has no effect on elements on non-interesting cycles. Due to the cost function, lower indices are more costly than higher indices. We look for the interesting cycle that contains the position with the least index among all interesting cycles and let l be the

smallest index on it. There are indices i and $j := \pi(i)$ on this cycle with $x_i = 0$ and $x_j = 1$. Let k be such that $\pi^k(l) = i$. Then $\mathbf{x} \circ \pi^k$ has a 0 at position i but $\mathbf{x} \circ \pi^{k+1}$ has a 1. In other words, $\mathbf{x} \circ \pi^k$ is a local optimum

In example in Figure 1 we have $x = 3$, $j = 4$ and $d = 1$. The local optimal permutation is hence π.

4.2 Finding a global optima

▶ **Theorem 1** ([1]). *2-Permutation Global Orbit Minimum is NP-hard.*

This follows via a reduction from Independent Set where we encode a graph as a bit string, one bit per potential edge, and the permutations basically allow us to move the vertices of the independent set I to the front, generating a large prefix of $\binom{|I|}{2}$ many 0's.

Since 1-Permutation Local Orbit Minimum was so clearly solvable in polynomial time (even by the greedy algorithm), it comes as a surprise that global optimization, even for *one* permutation, is NP-hard:

▶ **Theorem 2.** *1-Permutation Global Orbit Minimum is NP-hard.*

We will define an intermediate NP-complete problem called Disjunctive Chinese Remainder, short DCR. For two numbers $t, m \in \mathbb{N}$ and a set $S \subseteq \mathbb{N}$, we write

$$t \notin S \mod m$$

to state that $t \not\equiv s \mod m$ for all $s \in S$. Now in the DCR decision problem, we are given moduli $m_1, \ldots, m_l$ and sets of "forbidden remainders" $S_1, \ldots, S_l$ with $S_i \subseteq \mathbb{Z}_{m_i} := \{0, 1, \ldots, m_i - 1\}$. All numbers are given in unary and the moduli are not required to be pairwise co-prime. DCR asks for a solution $t \in \mathbb{N}$ of the system

$$t \notin S_i \mod m_i \qquad \forall i = 1, \ldots, l\ .$$

This is clearly in NP: if there is a solution $x \in \mathbb{N}$, then there is one with $0 \leq t \leq \operatorname{lcm}(m_1, m_2, \ldots m_l)$ and thus t has polynomially many bits in binary. Verifying that this t is a solution can now be done by division with remainder.

▶ **Lemma 3.** *DCR is NP-complete.*

Proof. We reduce from 3-Colorability. Given a graph $G = (V, E)$ with $|V| = n$, we let $3 = p_1, p_2, \ldots, p_n$ be the first n prime numbers greater than 2. By the prime number theorem, p_n is polynomial in n, thus the p_i can be found in polynomial time by a brute force search using naive prime number testing.

We let $N = p_1 \cdot p_2 \cdot \cdots \cdot p_n$ and define the following function from $\mathbb{Z}_N$ to 3-colorings of the vertices V: given a number $0 \leq t \leq N - 1$, the corresponding coloring $c_t : V \to \{r, g, b\}$ is defined by

$$c_t(v_i) = \begin{cases} r & \text{if } t \equiv 0 \mod p_i \\ g & \text{if } t \equiv 1 \mod p_i \\ b & \text{else.} \end{cases}$$

By the Chinese Remainder Theorem, this is a surjective function and thus every 3-coloring can be encoded by one single number $0 \leq x \leq N-1$. For an edge $e = \{u, v\}$, we write a constraint that makes sure that u and v receive different colors. For each pair $(a, b) \in \mathbb{Z}_{p_u} \times \mathbb{Z}_{p_v}$ with

$(a,b) = (0,0)$ or $(a,b) = (1,1)$ or $a \geq 2, b \geq 2$, we compute the unique number $c \in \mathbb{Z}_{p_u q_v}$ with $c \equiv a \mod p_u$ and $c \equiv b \mod q_v$. Let S_e be the set of all numbers c thus constructed, set $m_e := p_u \cdot p_v$ and write the constraint

$$x \notin S_e \mod m_e \ . \tag{1}$$

If e is the i^{th} edge of the graph, we set $m_i = p_u p_v$ and $S_i = S_e$. We see that x satisfies (1) if and only if x encodes a properly colors edge e. ◀

▶ **Lemma 4.** *DCR reduces to* 1-Permutation Global Orbit Minimum.

Proof. Given an instance of DCR we set $M = m_1 + m_2 + \cdots + m_l$ and define a permutation π on $[M]$ that has l disjoint cycles, one of length m_i for each i. We label the elements of the i^{th} cycle consecutively with the numbers $0, \ldots, m_i - 1$. We define a bit string $\mathbf{x} \in \{0,1\}^M$ that has exactly one 1 in each cycle, placed on the element that has label 0. The orbit element $\mathbf{x} \circ \pi^t$ still has exactly one 1 in cycle i, but now at the vertex labeled $t \bmod m_i$.

For each cycle i, let F_i be the elements on it whose labels are in the set S_i of forbidden remainders and let $F := F_1 \cup \cdots \cup F_l$. We order the elements of $[M]$ such that the elements of F come first. There exists a solution $t \in \mathbb{N}$ to the DCR instance if and only if the string $\mathbf{x} \circ \pi^t$ has no 1 in any position in F. ◀

5 PLS-Hardness

We now state and prove our main result:

▶ **Theorem 5.** *k-Permutation Local Orbit Minimum is PLS-complete.*

In order to view it as a PLS-problem we have an instance I consisting of the permutations $\pi_1, .., \pi_k$ and the string $s \in \{0,1\}^N$. Solutions are permutations from $\langle \pi_1, \ldots \pi_k \rangle$ as we can efficiently recognize whether permutations are in the group generated by $\pi_1, .., \pi_k$. The start solution is the identity. Neighbors of π are permutations $\pi \circ \pi_i$ for $i \in \{1, \ldots k\}$. The cost of a permutation π is $\sum_{i=1}^{N} s(\pi(i)) 2^{N-i}$, where $s(\pi(i))$ is the digit of the position that i is mapped to by π. All these can be computed in polynomial time, hence the problem is in PLS.

A solution π is cheaper than a solution σ for a string s if there is an integer i such that for all $j < i$ we have that $s(\sigma(j)) = s(\pi(j))$ and $s(\pi(i)) < s(\sigma(i))$ as the cost is a geometric sum.

We can turn the minimization problem into a maximization problem by inverting the string.

5.1 High-level idea

We reduce from the PLS-complete problem FLIP. This problem is especially suitable since it uses a lexicographic cost function, too. Formally FLIP is defined as follows:

▶ **Definition 6.** *An instance of FLIP consists of a circuit C with n inputs and m outputs. Feasible solutions are all input assignments, i.e., the set $\{0,1\}^n$. The cost of a solution $\mathbf{x}$ is the output of $C(\mathbf{x}) \in \{0,1\}^m$. Two solutions are neighbors if they differ in a single bit. The cost function is defined by reading the output as a number in binary; in other words, by the lexicographic order on $\{0,1\}^m$. We are asked to find a solution whose cost is minimal among all its neighbors.*

We use an idea by Krentel[10] and have $n+1$ copies $C_0, C_1, \ldots, C_n$ of the circuit C, where C_0 is fed as an input $\mathbf{x}$ and C_i is fed as an input $\mathbf{x} \oplus \mathbf{e}_i$, i.e., $\mathbf{x}$ with the i-th bit flipped. The setup is depicted in Figure 2.

The permutation group consists of two types of permutations. The first kind π_i^j simulates flipping the output of gate i in circuit j; we allow for gates and hence circuits to be temporarily evaluated incorrectly. The input and output of a gate are not saved anywhere but syntactically built into the permutations and string. An exception is the input and output of the circuit, which we save for later usage. We use some positions as very important control bits to ensure that the correct evaluation is always possible.

The second type of permutation σ_j swaps the circuit 0 with the circuit j and flips the j-th input bit for all other circuits. This simulates a step in the FLIP-problem.

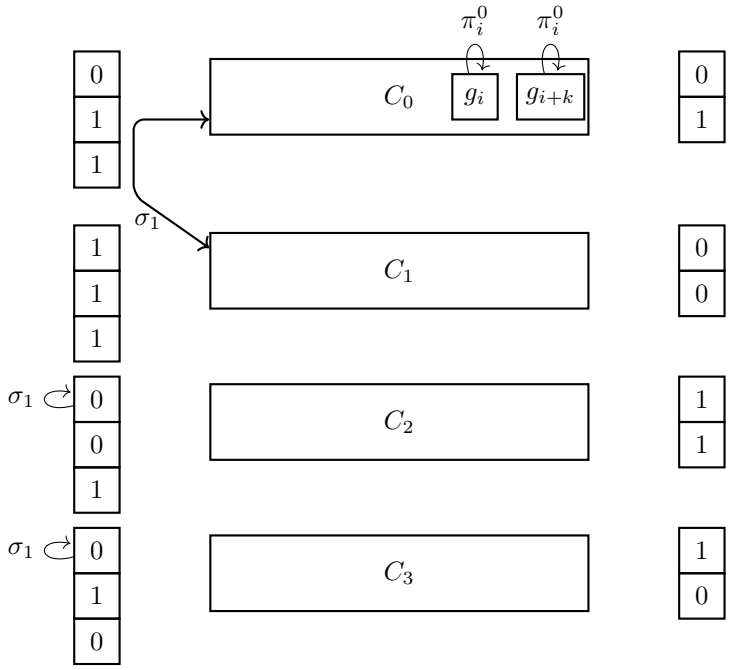


Figure 2 A high level overview of the reduction.

5.2 Definition of the reduction

We assume without loss of generality that the circuit C consists only of NAND-gates. Additionally, we assume that no input is directly passed to an output, i.e. on every path from an input to an output, there is at least one gate.

We will now describe the set V of *positions*, the permutations in S_V, and how strings in $\{0,1\}^V$, called *assignments*, correspond to the circuits $C_0, \ldots, C_n$ computing their values on an input $\mathbf{x}$ and its neighbors $\mathbf{x} \oplus \mathbf{e}_i$. We face two main challenges:

1. We need an operation of the form "flip bit i", but permutations can only permute positions, not flip bits. We solve this by replacing position i by two positions i_0, i_1 and encoding $[y_i = 0]$ by $[y_{i_0} = 0, y_{i_1} = 1]$ and $[y_i = 1]$ by $[y_{i_0} = 1, y_{i_1} = 0]$. Flipping bit i then corresponds to the permutation that swaps i_0 and i_1, i.e., the transposition (i_0, i_1). We call the one-position-per-bit view the *condensed view* and the two-positions-per-bit view the *expanded view*. We will give details in the full version due to space restrictions. For now, we phrase things in the condensed view.

2. Usually, when we flip an input x_i to a circuit C, we imagine the change propagating instantaneously through the circuit C, potentially changing its output. Here, we need to allow for a way for this change to proceed gradually; therefore, we allow gates to be temporarily in an incorrect state.

When we have a position $i \in V$ and an assignment $\mathbf{y} \in \{0,1\}^V$ and $y_i = b$, we sometimes say *i is assigned value b* and sometimes *the label of i is b*.

State of a gate. The *state of a gate* is a triple (x, y, b) where x, y are the two input bits and b is the output bit. If $b = \neg(x \wedge y)$ we call (x, y, b) *correct*, because b is what it is supposed to be: the NAND of x and y; otherwise, we call it *incorrect*. A gate g is represented by a gadget of four positions $g_{00}, g_{01}, g_{10}, g_{11}$, ordered as a 2×2 square, as shown in Figure 3. A state (x, y, b) is encoded as follows: position g_{xy} is labeled b, the three remaining ones are labeled with $\neg b$. Note that not all labelings of the gadget correspond to a gate state, only those where the number of positions labeled 1s is one or three.

▶ **Observation 7.** *The triple (x, y, b) is correct if and only if the position g_{11} in its representation is labeled* 0.

This is the core reason why we use this representation: we can determine correctness of a gate by reading just one bit. This will be important later when defining a cost function: having a 1 at those *control positions* is bad.

Input and output variables. Let $x_i^{(j)}$ be the i-th input variable to the j-th circuit (so $1 \le i \le n$ and $0 \le j \le n$). We introduce one position for each $x_i^{(j)}$. Similarly, let $c_k^{(j)}$ be the k-th output value of the j-th circuit; we introduce one position for each $c_k^{(j)}$.

A central definition is that of a well-behaved assignment. Basically, it formalizes when an assignment encodes the partial evaluation of the inputs by the $n + 1$ circuits.

▶ **Definition 8.** *An assignment $\mathbf{y} \in \{0,1\}^V$ is called* well-behaved *if the following hold:*

- *For each gate g, the four positions $g_{00}, g_{01}, g_{10}, g_{11}$ are the encoding of a gate state $(a_1, a_2, b) \in \{0,1\}^3$.*
- *If the output of a gate g is the l-th input of some gate h, then the corresponding input and output values agree, i.e., if the state of h is (a'_1, a'_2, b') then $b = a'_l$.*
- *If the l-th input of g is an input variable $x_i^{(j)}$, then $x_i^{(j)}$ is assigned the value a_l.*
- *If g is the k-th output gate of circuit j, then its output value b is the same as the label of $c_k^{(j)}$.*
- *The labels of the input values $x_i^{(j)}$ equal $x_i^{(0)}$ if $i \neq j$, and are unequal if $i = j$; in words, if the labels of the input positions of C_0 form a vector $\mathbf{x} \in \{0,1\}^n$, then those of C_j form the vector $\mathbf{x} \oplus e_j$.*

Next, we describe the permutations on the positions. They come in two types: (1) flipping a gate and (2) swapping two circuits.

Flipping a gate. If g is in state (a_1, a_2, b), then flipping g means replacing b by $\neg b$, and for each gate h (in state (a'_1, a'_2, b')) into which g feeds as l-th input, flipping a'_l. Since our permutations do not work on the *state* of a gate but on its representation in the four-position gadget (Figure 3), we work as follows: we flip the labels in all positions of g, i.e., $g_{00}, g_{01}, g_{10}, g_{11}$; if g is the first input of h, we swap positions of h horizontally: swap h_{00} with h_{10} and h_{01} with h_{11}; if g is the second input of h, we perform a vertical swap: h_{00}

with h_{01} and h_{10} with h_{11}. This operation changes the state of g from incorrect to correct (or vice versa) and may also change correctness of h. If g happens to be the k-th output bit of circuit j, then this operation also flips position $c_k^{(j)}$. See Figure 4 for an example of two consecutive gates being flipped. We call this permutation π_g. If g is the i-th gate in circuit j, we may also call it π_i^j. Note that if $\mathbf{y}$ is a well-behaved then $\mathbf{y} \circ \pi_g$ is well-behaved, too.

Swapping two circuits. We want a permutation that simulates flipping the i-th input bit to C, the circuit in the instance of FLIP. We achieve this by swapping C_0 with C_i – that is, swapping every position (input values, values in gate gadgets, output values) in C_0 with its corresponding position in C_i, and simultaneously flipping the i-th input bit $x_i^{(j)}$ for every circuit C_j with $j \in \{1, \dots, n\} \setminus \{i\}$; naturally, if this $x_i^{(j)}$ is the first input to a gate g in C_j, we have to perform the "horizontal swap" at g outlined above, and if it is the second input to g, a "vertical swap" at g. We call this permutation σ_i. Again, if $\mathbf{y}$ is well-behaved then $\mathbf{y} \circ \sigma_i$ is well-behaved, too.

The starting string $\mathbf{y_{start}} \in \{0,1\}^V$. This is the assignment where C_0 has input $\mathbf{0} \in \{0,1\}^n$ and C_i has input $\mathbf{e}_i \in \{0,1\}^n$ and all gates have output 0 (whether correctly or incorrectly). This is certainly well-behaved (or rather, can be made well-behaved by making sure that input to gate h matches output of gate g should they be connected).

We now have a set V of positions, an assignment $\mathbf{y}_{\text{start}} \in \{0,1\}^V$, and a set of permutations-gate-flippers π_g for each gate g and circuit-swappers σ_i for each $i \in [n]$. They generate a subgroup G of S_V. It is clear that the orbit of $\mathbf{y}_{\text{start}}$ under G is the set of well-behaved assignments.

5.3 The cost function

We have promised to use a lexicographic ordering as a cost function. That is, if $\mathbf{y}, \mathbf{y}' \in \{0,1\}^V$ are two assignments, then $\mathbf{y}$ is better than $\mathbf{y}'$ (meaning lower cost) if $\mathbf{y} \prec_{\text{lex}} \mathbf{y}'$. Thus, to define the cost function it suffices to specify an ordering on the positions in V.

- Positions in C_0 come before positions in C_1 and so on.
- Within a circuit, most important are the control positions g_{11} of the gates, followed by the output gates, followed by all remaining positions.
- Within control positions in the same circuit, the order follows the topological ordering of the circuit, i.e., if g feeds into h, then g's control position comes before h's.

5.4 Proof of Correctness

In this section we now prove the correctness and tightness of our reduction. In order to show the correctness we have to show that if $\pi \in G$ is a local optimum of the k-Permutation Local Orbit Minimum instance, i.e., if the assignment $\mathbf{y} := \mathbf{y}_{\text{start}} \circ \pi$ cannot be further improved by applying a π_g or a σ_j, then the input to C_0 encoded in $\mathbf{y}$ corresponds to a local optimum of FLIP.

Well-behaved permutations for a string s and a circuit C are interesting, because they realize the evaluation of the circuit somehow. The only problem is that the gate does not have the correct output in the current permutation with the string s compared to C.

▶ **Lemma 9.** *Let $\mathbf{y}$ be a well-behaved assignment and suppose that $y(g_{11}) = 0$ for every gate – every control position is labeled 0. Then all output positions are mapped to correct results according to C. That is, if $\mathbf{x} \in \{0,1\}^n$ is the vector to which the positions of circuit C_0 are mapped under $\mathbf{y}$, then*

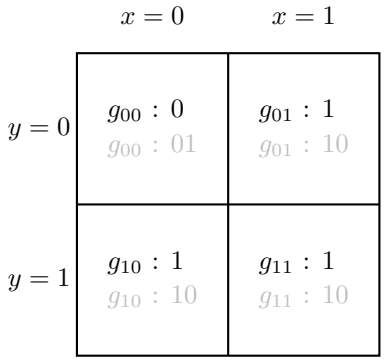
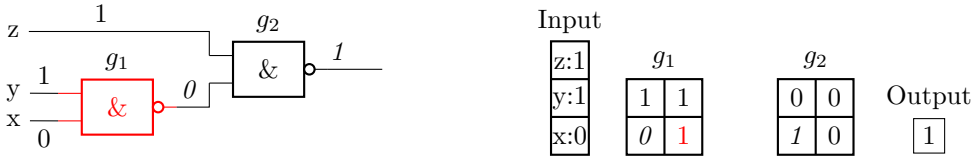
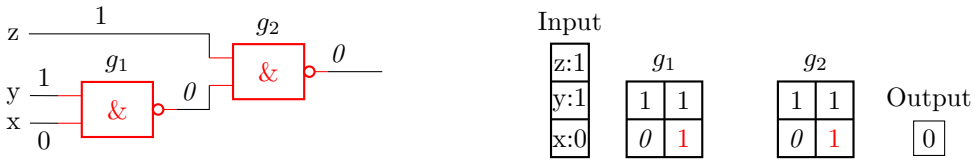


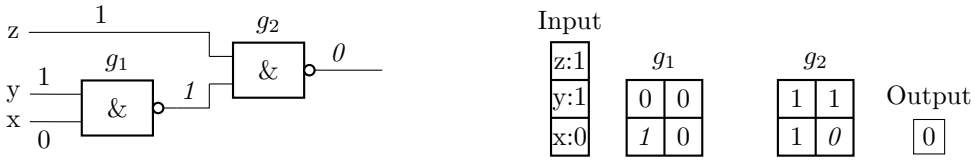
Figure 3 An example gate configuration with the input $x = 0$ and $y = 0$ and the output 0. This is incorrect for a NAND-gate as indicated by the control bit. In light grey we denote the expanded encoding of the positions.



(a) Step 1: Initial state. Gate g_1 is evaluated incorrectly, g_2 correctly. Still, we call this a *well-behaved* state.



(b) Step 2: Application of the permutation π_{g_2} which leads to g_2 being now incorrect as well. This is not a local improvement step. While it minimizes the output, the more expensive control bit of g_2 is now active. Still the relationship between the gate state and the string under the current permutation holds, so this is well-behaved as well.



(c) Step 3: Application of the permutation π_{g_1}. Due to the inversion, g_1 is now correct and the change of its output is reflected in g_2 which is now also correct since the one was swapped out of the control position. This is now a local optima.

Figure 4 Step-by-step evaluation of the circuit and reduction process. Each step shows the circuit state (left) and the reduction state (right). Currently incorrect gates are marked red and the output of a gate is identifiable via italics.

1. *the input positions of C_i are mapped to $\mathbf{x} \oplus \mathbf{e}_i$ (this actually holds for every well-behaved $\mathbf{y}$, control positions being 0 or 1),*
2. *the output positions of C_0 are mapped to $C(\mathbf{x})$;*
3. *the output positions of C_j are mapped to $C(\mathbf{x} \oplus \mathbf{e}_j)$*

Proof. This follows from Observation 7 and induction over the sequence of gates. ◀

The previous lemma tells us that all control positions should be mapped to zero in any local optimum so that all output positions are mapped to the correct output of C given the input. We show that we can always apply a permutation to achieve this.

▶ **Lemma 10.** *Let π be a permutation from G. We consider a gate g_i in the circuit j. The control position of g_i is mapped to position of the form $g^j_{i,k}$ by π*

Proof. This can be proven by induction on the structure of π in the permutation group. Any generator preserves this property as it either does not affect this position at all $\pi^{j'}_{i'}$ for $i \neq i'$ or $j \neq j'$. If both i' and j' are equal to i and j, then the position is simply inverted which preserves this property. Additionally, any permutation σ_j either maps a gate to itself or swaps the gate, so that the claim holds here as well. ◀

▶ **Lemma 11.** *If $\mathbf{y}$ is well-behaved and some control position g_{11} is 1 under $\mathbf{y}$, then $\mathbf{y}$ is not a local optimum.*

Proof. Among all control positions assigned 1, let g_{11} be the highest-ranking (i.e., of smallest index). Now apply π_g, i.e., flip gate g, which inverts the bit by the previous lemma. Under $\mathbf{y} \circ \pi_g$, the control position g_{11} is now correct; successor gates h in the same circuit might now become incorrect, but their control positions have lower rank by our ordering; gates in other circuits are not affected. Thus, $\mathbf{y} \circ \pi_g \prec_{\text{lex}} \mathbf{y}$, and $\mathbf{y}$ is not a local optimum. ◀

▶ **Corollary 12.** *In any local optimum all sub circuits are correctly evaluated.*

The previous lemmas suffice to show the main result.

▶ **Lemma 13.** *Let $\mathbf{y} = \mathbf{y}_{\text{start}} \circ \pi \in \{0,1\}^V$ be a local optimum an instance of* k-Permutation Local Orbit Minimum. *Suppose the input variables of C_0 are mapped to some $\mathbf{x} \in \{0,1\}^n$. Then $\mathbf{x}$ is a local optimum of the FLIP instance.*

Proof. Since $\mathbf{y}$ is well-behaved, the input variables of C_i are mapped to $\mathbf{x} \oplus \mathbf{e}_i$. Since $\mathbf{y}$ is a local minimum, by the corollary, the output values of C_j are in fact mapped to the correct output $C(\mathbf{x} \oplus \mathbf{e}_j)$.

Suppose now that the mapped input is not a local optimum for the FLIP instance. Then there must be neighbor with a better output, i.e., $C(\mathbf{x} \oplus e_j) \prec_{\text{lex}} C(\mathbf{x})$. Now consider $\mathbf{y}' = \mathbf{y} \circ \sigma_j$. Under $\mathbf{y}'$, the control positions of C_0 are all mapped to 0 (because those of C_j were under $\mathbf{y}$); the output of C_0 under $\mathbf{y}'$ is better than under $\mathbf{y}$ because $C(\mathbf{x} \oplus e_j) \prec_{\text{lex}} C(\mathbf{x})$. Control positions in C_i with $i \geq 1$ might now be 1, but their priority is less than that of C_0's output. Thus, $\mathbf{y}'$ is better than $\mathbf{y}$, and $\mathbf{y}$ is not a local optimum. ◀

This concludes the proof of Theorem 5. Every permutation used in the reduction has an order of two, so it is an involution. Still, these permutations do not form a commutative group due to the σ_i permutations. This is to be expected as even finding a global optimum for the lexicographical leader of a string under an abelian permutation group where every element has an order of two is polynomial time solvable[1, Section 3.1].

We additionally note that the reduction is tight and hence finding a local optimal bitstring under permutations via the standard algorithm is PSPACE-complete.

▶ **Theorem 14.** *The given reduction is tight.*

Proof. Let I be an instance of FLIP and (f, g) the previously defined reduction. We use as $\mathcal{R}$ simply the set of all permutations in the group of the generators which necessarily contains all local optima. We can find for any solution s of I in polynomial time a solution π with

$g(I, \pi) = s$ in $\mathcal{R}$ by applying the σ_j permutations to construct the needed input string. This requires applying at most n permutations which can be done in polynomial time. Finally, we see that any path where only the endpoints are in $\mathcal{R}$ must be an edge. If the edge is due to an π_i^j permutation the input does not change and hence both endpoints are mapped to the same solution. If the edge is alternatively due to a σ_j permutation this changes the input in one variable and is hence an edge in $TG(I)$ as it corresponds directly to a flip there. ◀

6 Realizing the permutations in propositional formula

In the problem Locally Minimal Solution we are given a CNF formula F over some variables V, a satisfying assignment $\alpha : V \to \{0, 1\}$, and a list of permutations $\pi_1, \ldots, \pi_k$ on V such that F is invariant under π_i (that is, when applying π_i to each variable occurrence in F, the resulting formula F' is equal to F up to a re-ordering of the clauses and the literals therein). The task now is to find a satisfying assignment β of F that such that $\beta \circ \pi_i \succeq_{\text{lex}} \beta$. This is clearly in PLS: whether F is invariant under the π_i and whether β satisfies F are both easy to check. Note that it is not required that β be in the orbit of α under $\langle \pi_1, \ldots, \pi_k \rangle$.

▶ **Theorem 15.** *Locally Minimal Solution is PLS-complete.*

Proof. We will define a formula F whose satisfying assignments correspond exactly the well-behaved assignments to the positions, as defined in Definition 8. We first describe how to encode one circuit of the total $n + 1$ circuits. We have input variables $x_1, \ldots, x_n$ to the circuit; we introduce one *gate output variable* g_{out} for each gate; and four *gate control variables* $g_{00}, g_{01}, g_{10}, g_{11}$ for the four positions in the square-representation of that gate, as in Figure 3. For each such variable u we introduce its twin $\tilde{u}$ and add $(u \leftrightarrow \neg\tilde{u})$. In other words, the (positive) literal $\tilde{u}$ simulates the negative literal $\bar{u}$. Take a gate g, let u, v be its inputs and w its output. The following formula F_g ensures that the gate control variables g_{ab} are set correctly as required for a well-behaved assignment:

$$
\begin{aligned}
&(u \wedge v \wedge w) \to (\tilde{g}_{00} \wedge \tilde{g}_{01} \wedge \tilde{g}_{10} \wedge g_{11})\\
&(u \wedge v \wedge \tilde{w}) \to (g_{00} \wedge g_{01} \wedge g_{10} \wedge \tilde{g}_{11})\\
&(u \wedge \tilde{v} \wedge w) \to (\tilde{g}_{00} \wedge \tilde{g}_{01} \wedge g_{10} \wedge \tilde{g}_{11})\\
&(u \wedge \tilde{v} \wedge \tilde{w}) \to (g_{00} \wedge g_{01} \wedge \tilde{g}_{10} \wedge g_{11})\\
&(\tilde{u} \wedge v \wedge w) \to (\tilde{g}_{00} \wedge g_{01} \wedge \tilde{g}_{10} \wedge \tilde{g}_{11})\\
&(\tilde{u} \wedge v \wedge \tilde{w}) \to (g_{00} \wedge \tilde{g}_{01} \wedge g_{10} \wedge g_{11})\\
&(\tilde{u} \wedge \tilde{v} \wedge w) \to (g_{00} \wedge \tilde{g}_{01} \wedge \tilde{g}_{10} \wedge \tilde{g}_{11})\\
&(\tilde{u} \wedge \tilde{v} \wedge \tilde{w}) \to (\tilde{g}_{00} \wedge g_{01} \wedge g_{10} \wedge g_{11})
\end{aligned}
$$

F_g can easily be written as a CNF formula. The permutation π_g amounts to flipping the output of gate g and flipping the corresponding inputs at those gates h that g's output feeds into. Thus, π_g is

$$(w\tilde{w})(g_{00}\tilde{g}_{00})(g_{01}\tilde{g}_{01})(g_{10}\tilde{g}_{10})(g_{11}\tilde{g}_{11}) \circ (\text{stuff at successor gates}) \tag{2}$$

F_g is invariant under π_g (even when we write it as a CNF). Next, there is a permutation σ that flips the input u of g. This swaps the two rows of the square representation of g:

$$(\text{stuff at predecessor gates}) \circ (u\tilde{u})(g_{00}g_{10})(\tilde{g}_{00}\tilde{g}_{10})(g_{01}g_{10})(\tilde{g}_{01}\tilde{g}_{11}) \tag{3}$$

If u is the output of some gate h, then σ is π_h and "stuff at predecessor gate" is what we describe in (2); otherwise u is an input variable x_i and "stuff at predecessor gate" does nothing – for now.

This formula describes well-behaved assignments in one of the $n+1$ circuits $C_0, \dots, C_n$. We now create $n+1$ copies of this formula, introducing a fresh version of each variable, so the j^{th} input variable x_j becomes $x_j^{(i)}$ in C_i, and g_{00} becomes $g_{00}^{(i)}$ and so on. We create a formula H to ensure that $x_j^{(i)}$ and $x_j^{(0)}$ differ if and only if $i = j$:

$$H := \bigwedge_{\{i_1,i_2\} \in \binom{\{0,\dots,n\}}{2}} \bigwedge_{j=1}^{n} \left\{ \begin{array}{ll} \left(x_j^{(i_1)} \leftrightarrow x_j^{(i_2)}\right) \wedge \left(\tilde{x}_j^{(i_1)} \leftrightarrow \tilde{x}_j^{(i_2)}\right) & \text{if } j \notin \{i_1, i_2\} \\ \left(x_j^{(i_1)} \leftrightarrow \tilde{x}_j^{(i_2)}\right) \wedge \left(\tilde{x}_j^{(i_1)} \leftrightarrow x_j^{(i_2)}\right) & \text{if } j \in \{i_1, i_2\} \ . \end{array} \right\} \tag{4}$$

The permutation σ_i flipping circuit C_i and C_0 and inverting $x_i^{(i')}$ for all other i' can be written as

$$\begin{aligned} &\left(u_j^{(i)} u_j^{(0)}\right)\left(\tilde{u}_j^{(i)} \tilde{u}_j^{(0)}\right) && \text{(for each input and gate output and control variable } u) \\ \circ &\left(x_i^{(i')} \tilde{x}_i^{(i')}\right) \circ \Big(\text{do (3) at gates having } x_i^{(i')} \text{ as input}\Big) && \text{(for all } i' \in \{1,\dots,n\} \setminus \{i\}) \end{aligned}$$

This forms the final formula $F = H \wedge \bigwedge_{i=0}^{n} \bigwedge_{\text{gate } g} F_g^{(i)}$. Its satisfying assignments correspond exactly to the well-behaved assignments described above and that each π_g and each σ_i is indeed a symmetry of F. The order of the variables is as in the previous proof: of highest priority are the control variables $g_{11}^{(0)}$ (following the topological order of the gates in the circuit); then the output variables of C_0; then to the control variables of the other circuits; then all the rest. It is easy to provide an "initial" satisfying assignment $\alpha \in \mathrm{SAT}(F)$. Finally, if β is some satisfying assignment of F that is locally minimal, i.e., cannot be improved by applying any π_g or σ_i, then β represents a configuration in which all circuits $C_0, \dots, C_n$ are correctly evaluated and no C_i outputs something better than C_0; in other words, a local optimum of FLIP. This shows that Locally Minimal Solution is PLS-complete. ◀

7 Conclusion and open questions

We have shown that the problem is PLS-hard in general and hence a polynomial time algorithm is unlikely unless P=PLS. In the theory of PLS-complete problems we thereby demonstrated another example of a hard problem with lexicographic weights. The used permutations are realistic in the sense that they can occur in an actual CNF formula.

Our above reduction requires polynomially many permutations. There is an efficient algorithm for the case of *one* permutation. What about when we are given a constant number of permutations?

What about if the given permutations form an Abelian group? Is it still PLS-hard to find a local minimum?

The neighborhood of a solution π is in our work defined as $\pi \circ \pi_i$ for some generator π_i. An alternative neighborhood would be $\pi_i \circ \pi$, which would place the generator between the string and the current permutation. While the results of the one permutation case trivially hold again, the hardness proof does not transform to this formulation.

References

1. László Babai and Eugene M. Luks. Canonical labeling of graphs. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 171–183. ACM, 1983. doi:10.1145/800061.808746.
2. Christoph Buchheim and Michael Jünger. Linear optimization over permutation groups. *Discret. Optim.*, 2(4):308–319, 2005. doi:10.1016/J.DISOPT.2005.08.005.
3. Dominic Dumrauf and Burkhard Monien. On the *PLS*-complexity of maximum constraint assignment. *Theor. Comput. Sci.*, 469:24–52, 2013. doi:10.1016/J.TCS.2012.10.044.
4. Alex Fabrikant, Christos H. Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 604–612. ACM, 2004. doi:10.1145/1007352.1007445.
5. John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: CLS = PPAD ∩ PLS. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 46–59. ACM, 2021. doi:10.1145/3406325.3451052.
6. Merrick L. Furst, John E. Hopcroft, and Eugene M. Luks. Polynomial-time algorithms for permutation groups. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 36–41. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.34.
7. Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Expressing symmetry breaking in DRAT proofs. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction – CADE-25 – 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 591–606. Springer, 2015. doi:10.1007/978-3-319-21401-6_40.
8. David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.
9. Leszek Aleksander Kolodziejczyk and Neil Thapen. The strength of the dominance rule. In Supratik Chakraborty and Jie-Hong Roland Jiang, editors, *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India*, volume 305 of *LIPIcs*, pages 20:1–20:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SAT.2024.20.
10. Mark W. Krentel. On finding locally optimal solutions. In *Proceedings: Fourth Annual Structure in Complexity Theory Conference, University of Oregon, Eugene, Oregon, USA, June 19-22, 1989*, pages 132–137. IEEE Computer Society, 1989. doi:10.1109/SCT.1989.41819.
11. Christos H. Papadimitriou. On graph-theoretic lemmata and complexity classes (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 794–801. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89602.
12. Christos H. Papadimitriou, Alejandro A. Schäffer, and Mihalis Yannakakis. On the complexity of local search (extended abstract). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 438–445. ACM, 1990. doi:10.1145/100216.100274.
13. Alejandro A. Schäffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM J. Comput.*, 20(1):56–87, 1991. doi:10.1137/0220004.
14. Neil Thapen. personal communication.
15. Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. Drat-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014 – 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014. doi:10.1007/978-3-319-09284-3_31.

Improved Hardness-Of-Approximation for Token-Swapping

Sam Hiken
Department of Computer Science and Engineering, University of Michigan, Ann Arbor, MI, USA

Nicole Wein
Department of Computer Science and Engineering, University of Michigan, Ann Arbor, MI, USA

Abstract

We study the token swapping problem, in which we are given a graph with an initial assignment of one distinct token to each vertex, and a final desired assignment (again with one token per vertex). The goal is to find the minimum length sequence of *swaps* of adjacent tokens required to get from the initial to the final assignment.

The token swapping problem is known to be NP-complete. It is also known to have a polynomial-time 4-approximation algorithm. From the hardness-of-approximation side, it is known to be NP-hard to approximate with a ratio better than 1001/1000.

Our main result is an improvement of the approximation ratio of the lower bound: We show that it is NP-hard to approximate with ratio better than 14/13.

We then turn our attention to the *0/1-weighted* version, in which every token has a weight of either 0 or 1, and the cost of a swap is the sum of the weights of the two participating tokens. Unlike standard token swapping, no constant-factor approximation is known for this version, and we provide an explanation. We prove that 0/1-weighted token swapping is NP-hard to approximate with ratio better than $(1-\varepsilon)\ln(n)$ for any constant $\epsilon > 0$.

Lastly, we prove two barrier results for the standard (unweighted) token swapping problem. We show that one cannot beat the current best known approximation ratio of 4 using a large class of algorithms which includes all known algorithms, nor can one beat it using a common analysis framework.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Graph algorithms analysis

Keywords and phrases algorithms, token-swapping, hardness-of-approximation, lower-bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.57

Related Version *Full Version*: `https://arxiv.org/abs/2410.19638`

Funding *Sam Hiken*: NSF grant CNS-2150186, Paglia Post-Baccalaureate Fellowship.

Acknowledgements We would like to thank anonymous reviewers for their helpful suggestions.

1 Introduction

In the *token swapping* problem [1, 8, 2, 25, 30, 31, 32, 35, 18, 9, 10, 16, 20, 17, 36, 21, 29], we are given an undirected n-vertex graph $G = (V, E)$, n distinct tokens, and two one-to-one assignments of tokens to vertices: a *starting* assignment, and a *target* assignment. A swap along an edge $(u, v) \in E$ switches the locations of the tokens on u and v. The token swapping problem asks how many swaps are needed to arrive at the target configuration from the starting configuration.

The token swapping problem is a fundamental and well-studied problem, and one of the central problems in the area of *reconfiguration* algorithms. It has also found relevance in a number of disparate areas including network engineering [2], robot motion planning [12, 28], and game theory [14]. Token swapping (mainly its parallel variant [3, 6, 12, 17, 37]) also


© Sam Hiken and Nicole Wein;
licensed under Creative Commons License CC-BY 4.0



has an extensively studied application to qubit routing (e.g. [5, 26, 19, 7, 11, 15, 24, 33]). Algorithms and heuristics for the token swapping problem have also undergone experimental evaluation [23, 27].

The token swapping problem is NP-complete [18], even when the underlying graph is a tree [1]. As a result, the research literature has focused on approximation algorithms. Miltzow, Narins, Okamoto, Rote, Thomas, and Uno gave a 4-approximation [18], which remains the best known. For the special case of trees, there is a 2-approximation, which was independently discovered 3 times using different algorithms [2, 32, 35].

From the hardness side, the above work [18] proved that token swapping is APX-hard (on general graphs). Thus, unless P = NP, token swapping does not admit a PTAS, and instead there exists some positive constant c for which there is no c-approximation. Regarding this constant c, they state *"We want to point out that a crude estimate for the constant c in [the NP-hardness result] is $c \approx 1 + 1/1000$. We do not believe that it is worth to compute c exactly. Instead, we hope that future research might find reductions with better constants."* This question is the main focus of our work.

We also consider the *0/1-weighted* version of token swapping, in which each token has a weight of either 0 or 1, and the cost of a swap is the sum of the weights of the two tokens. Despite being studied in prior work [8, 1], there is no known approximation algorithm for 0/1-weighted token swapping with a non-trivial approximation ratio. Furthermore, there is no known separation between 0/1-weighted token swapping and standard token swapping.

See the introductions of [8] and [1] for a more detailed survey of related work, including a sizable body of work on token swapping for a number of special classes of graphs.

1.1 Our Results

Our main result is the first hardness of approximation result for token swapping with an explicit approximation ratio of "reasonable" magnitude. Specifically, we show that it is NP-hard to obtain better than a 14/13-approximation:

▶ **Theorem 1.** *For any constant $\epsilon > 0$, it is NP-hard to approximate token swapping on graphs within a factor of $14/13 - \epsilon$.*

Our result is via a reduction from the label cover problem. We note that our result does not rely on the Unique Games Conjecture, and is instead a gap-preserving reduction from a regime of the label cover problem known to be NP-complete.

For 0/1-weighted token swapping, our next result explains the aforementioned gap in the literature by showing a large separation between the 0/1-weighted and unweighted versions. Specifically, we show that it is NP-hard to obtain better than an $(\ln n)$-approximation for 0/1-weighted token swapping:

▶ **Theorem 2.** *For any constant $\epsilon > 0$, it is NP-hard to approximate weighted token swapping with $\{0, 1\}$ weights on n vertices within a factor of $(1 - \epsilon) \cdot \ln n$.*

Our reduction uses a completely different technique from our main result, and is a much simpler reduction, from the set cover problem. This is notable in comparison to the known reductions in the token swapping literature, which tend to be quite complicated [1, 8, 18, 9].

Our final two results are barrier results regarding the algorithm and analysis techniques that could possibly achieve better than the current best-known approximation ratio of 4 (for standard unweighted token swapping). All known token swapping algorithms have the

natural property of *local optimality*: they never perform a swap that brings *both* tokens farther from their destinations.[1] We show that, strangely, this property must be violated to achieve any approximation ratio better than 4.

▶ **Theorem 3.** *For any $\delta > 0$, there exists a token swapping instance K so that for any locally optimal swap sequence of length k, $(4-\delta) \cdot OPT(K) \leq k$.*

In our second barrier result, we show that a proof technique used by all known analyses of approximation algorithms for token swapping, cannot yield better than a 4-approximation. In particular, all known approximation algorithms for token swapping on general graphs and trees [18, 35, 2] use the following approach to bound the approximation factor. Given a Token Swapping instance K, consider the quantity $total(K) = \sum_{t \in T} \text{dist}(f_1(t), f_2(t))$. We observe that $\frac{1}{2} total(K) \leq OPT(K)$, as each swap moves at most two tokens closer to their destinations. Proofs of correctness for approximation algorithms then show that they yield swap sequences of length at most $c \cdot total(K)$, which implies that the given algorithm is a $2c$-approximation. A natural approach to obtaining a $(4-\epsilon)$-approximation for token swapping on graphs, then, is to show that on any input an algorithm yields a swap sequence of length at most $c \cdot total(K)$, for some $c < 2$. Below, we show that this approach will not work.

▶ **Theorem 4.** *For any $\delta > 0$, there exists a token swapping instance K so that $OPT(K) \geq (2-\delta) \cdot total(K)$.*

The proof is given in the full version.

Lastly, in the full version we provide an alternative algorithm that gives a 4-approximation for token swapping (in addition to the known algorithm [18]). While the algorithm of [18] is an extension of the 2-approximation "happy swap algorithm" for trees [2], our algorithm is an extension of the 2-approximation "cycle algorithm" for trees [35].

1.2 Additional Related Work

Token swapping and its variants have been studied from many angles. Exponential-time algorithms and hardness under the Exponential Time Hypothesis (ETH) were studied in [18]. Parameterized complexity was studied in [9], where the authors show that token swapping is W[1]-hard when the parameter is the number of swaps, and show further hardness under ETH. Token swapping has also been studied on a variety of special classes of graphs including cycles [16], stars [21, 20], brooms [17, 8, 31], complete bipartite graphs [35], complete split graphs [36], and cliques [10] (dating back to Cayley in 1849). *Colored* token swapping has also been studied, where each token has a color and same-colored tokens are indistinguishable [18, 8, 9, 34, 34]. There are also several other models of token movement on graphs such as token sliding, token rotation, and token permutation (see e.g. [28]). See also the introductions of [8] and [1] for more details on the wealth of related work.

[1] We compare the notion of a locally optimal algorithm to that of an *ℓ-straying* algorithm, introduced in prior work on token swapping for *trees* [1]. These two notions are incomparable: a swap sequence can have either property without having the other. However, any algorithm that solves token swapping on general graphs must generate an $\Omega(n)$-straying swap sequence: consider the example of a cycle where each token wants to shift over by 1. For this reason, we do not consider ℓ-straying algorithms for general graphs, and focus instead on locally optimal algorithms.

2 Preliminaries

▶ **Definition 5.** *A* Token Swapping *instance* $K = (G, T, f_1, f_2)$ *consists of a graph* $G = (V, E)$*, a set of tokens* T *where* $|T| = |V|$*, and two one-to-one assignments* $f_1, f_2 : T \to V$*. We call* f_1 *and* f_2 *the* starting *and* target *configurations respectively. A swap along an edge* $(u, v) \in E$ *switches the locations of the tokens on* u *and* v*. The token swapping problem asks how many swaps are needed to arrive at the target configuration from the starting configuration.*

For a token swapping instance K, we denote by $OPT(K)$ the length of the shortest swap sequence. For a vertex v, we denote by $f_1^{-1}(v)$ and $f_2^{-1}(v)$ the tokens that begin and end on v. When a token t lies on the vertex v_1 of the path $p = v_1, v_2, ..., v_k$, we use the phrase *bubbling* t *across* p to denote the sequence of swaps $(v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k)$.

We also consider the following weighted variant of Token Swapping:

▶ **Definition 6** (Weighted Token Swapping)**.** *An instance of* Weighted Token Swapping $W = (G, T, w, f_1, f_2)$ *consists of a graph* $G = (V, E)$*, together with a set of tokens* T *of size* $|V|$*. The weight function* $w : T \to \mathbb{R}^{\geq 0}$ *assigns to each token a non-negative real weight. As in standard* Token Swapping*, the one-to-one functions* $f_1, f_2 : T \to V$ *map each token to a unique vertex, giving the starting and target configurations of* W*. The weight of an edge swap is the sum of the weights of the tokens being swapped. The weight of a swap sequence is the sum of the weights of the constitutive edge swaps. The output to* W *is the weight of the lowest-weight swap sequence from the starting to the target configuration.*

3 Hardness for standard token swapping

In this section, we prove Theorem 1.

▶ **Theorem 1.** *For any constant* $\epsilon > 0$*, it is NP-hard to approximate token swapping on graphs within a factor of* $14/13 - \epsilon$*.*

This improves on the lower bound presented in [18], which is roughly 1001/1000. We obtain our lower bound via a gap-preserving reduction from Label-Cover, defined below.

▶ **Definition 7** (Label Cover)**.** *An instance of* Label-Cover $\Phi = (X, Y, E, \Sigma, \Pi)$ *consists of a bipartite graph with vertex set* $X \cup Y$ *and edge set* E*, together with a finite alphabet* Σ*, and a set of constraints* $\Pi = \{\Pi_e : e \in E\}$*. Each constraint is a function* $\Pi_e : \Sigma \to \Sigma$*.*

A labelling of $X \cup Y$ *is a function* $\lambda : X \cup Y \to \Sigma$ *which assigns a label to each vertex. For* $x \in X$*,* $y \in Y$*, a constraint* $\Pi_{(x,y)}$ *is satisfied if* $\Pi_{(x,y)}(\lambda(x)) = \lambda(y)$*.*

We denote by $OPT(\Phi)$ the maximum fraction of constraints in Φ which can be satisfied by any labelling. An instance of the promise problem GapLabel-Cover$_{1,\gamma}(\Sigma)$ consists of a label-cover instance Φ with alphabet Σ, together with the guarantee that either $OPT(\Phi) = 1$ or $OPT(\Phi) < \gamma$. Here, γ is a positive constant close to 0. The following is a seminal result in the theory of hardness of approximation.

▶ **Theorem 8.** *For any constant* $\gamma > 0$*, there exists a sufficiently large constant* $|\Sigma|$ *(dependent only on* γ*) such that* GapLabel-Cover$_{1,\gamma}(\Sigma)$ *is NP-hard.*

Arora and Lund [4] describe a reduction from 3-SAT which, together with Raz's Parallel Repetition Theorem [22], implies that Theorem 8 remains true even on instances where the underlying graph is regular. Moreover, we may take the degree to be at least any arbitrarily

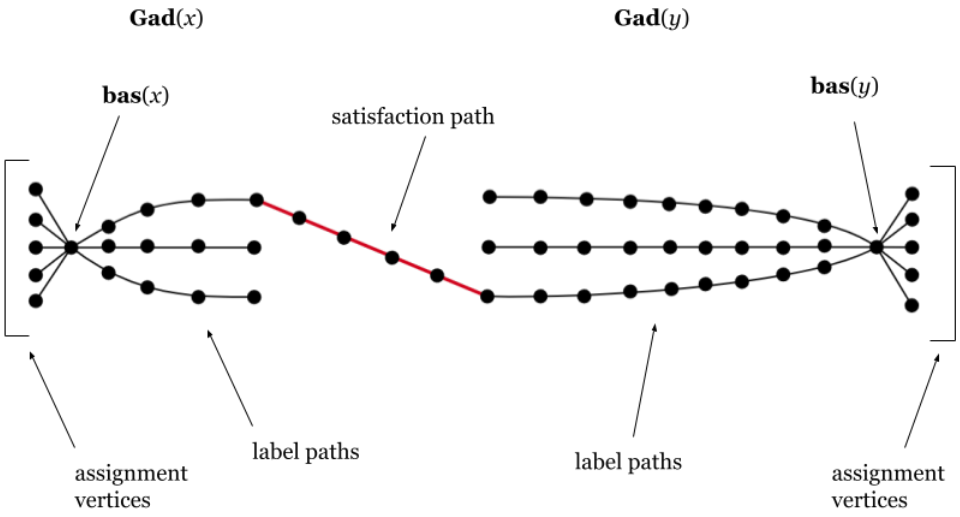


Figure 1 A left and right gadget, joined by a satisfaction path; $d = 5$, $|\Sigma| = 3$.

large constant by creating many copies of our graph $X \cup Y$ and adding a copy of the constraint $\Pi_{(x,y)}$ between every copy of x and every copy of y. Note that in such an instance $|X| = |Y|$. We use a gap-preserving reduction from label-cover to prove Theorem 1.

3.1 Construction

Our reduction maps a label cover instance $\Phi = (X, Y, E, \Sigma, \Pi)$, where the base graph has degree d, to a token swapping instance $K(\Phi) = (G, T, f_1, f_2)$.

We construct G by building a gadget $\mathbf{Gad}(v)$ for each $v \in X \cup Y$ as follows. Our construction for $\mathbf{Gad}(v)$ starts with a base vertex $\mathbf{bas}(v)$. We add d additional vertices adjacent to $\mathbf{bas}(v)$, which we call *assignment vertices* (recall that d is the degree in the label-cover instance). To each assignment vertex in $\mathbf{Gad}(v)$ we identify a unique edge $(v, w) \in E$ (where E is the label cover edge set) incident to v; we call this assignment vertex $\mathbf{asg}(v, w)$. We then create $|\Sigma|$ paths, each with $\mathbf{bas}(v)$ as an endpoint. We call these paths *label paths*. If $v \in X$, then each label path in $\mathbf{Gad}(v)$ contains $d - 1$ edges; if $v \in Y$, then each label path in $\mathbf{Gad}(v)$ contains $2d - 1$ edges. To each label path in $\mathbf{Gad}(v)$ we associate a unique $\sigma \in \Sigma$, and denote this label path by $\mathbf{lab}(v, \sigma)$.

If $v \in X$, then we call $\mathbf{Gad}(v)$ a *left gadget*; otherwise, we call $\mathbf{Gad}(v)$ a *right gadget*. We call an assignment vertex a left (resp. right) assignment vertex if it is in a left (resp. right) gadget. We denote the set of left gadgets by $\mathbf{L}$ and the set of right gadgets by $\mathbf{R}$.

Whenever it is the case that the label pair (σ_x, σ_y) satisfies the constraint $\Pi_{(x,y)}$, we add a path of d edges connecting the endpoint of $\mathbf{lab}(x, \sigma_x)$ opposite $\mathbf{bas}(x)$ with the endpoint of $\mathbf{lab}(y, \sigma_y)$ opposite $\mathbf{bas}(y)$. We call such a path a *satisfaction path*, and denote it by $\mathbf{sat}(x, \sigma_x, y, \sigma_y)$.

This gives our construction for G. An illustration of a left and right gadget connected by a satisfaction path given in Figure 1. We construct f_1 and f_2 as follows. For each $(x, y) \in E$, the tokens on $\mathbf{asg}(x, y)$ and $\mathbf{asg}(y, x)$ wish to swap positions with each other. That is, $f_2(f_1^{-1}(\mathbf{asg}(x, y))) = \mathbf{asg}(y, x)$, and $f_2(f_1^{-1}(\mathbf{asg}(y, x))) = \mathbf{asg}(x, y)$.

We will refer to such a token as an *assignment token*. We will call an assignment token t where $f_1(t)$ is a left (resp. right) assignment vertex a left (resp. right) assignment token. For $t \in T$ that are not assignment tokens, $f_1(t) = f_2(t)$. That is, t does not wish to move from its starting position.

The intuition behind the reduction is as follows: if there is a labelling λ satisfying every constraint in Π, then all the assignment tokens that start in $\mathbf{Gad}(v)$ can move to their destination along the single label path associated with $\lambda(v)$. Then, any assignment token seeking to *enter* $\mathbf{Gad}(v)$ can travel along this path, and will in the process swap with many of the assignment tokens leaving $\mathbf{Gad}(v)$, bringing them closer to their destination. If, on the other hand, no good labelling exists, then having so many efficient swaps becomes impossible.

We prove the following lemma.

▶ **Lemma 9.** *Let $\Phi = (X, Y, \Sigma, \Pi)$ be a label-cover instance whose graph is d-regular. The following two claims hold:*

- *If $OPT(\Phi) = 1$, then $OPT(K(\Phi)) \leq \left(\frac{13}{4}d^2 + d\right)|X \cup Y|$.*
- *If $OPT(\Phi) < \gamma$, then $OPT(K(\Phi)) > \left(\frac{7-\gamma}{2}d^2 - 5d\right)|X \cup Y|$.*

Given a label-cover instance Φ, the token swapping instance $K(\Phi)$ can be computed in polynomial time. We set d to be a large constant and γ a small constant, and the hardness result of Theorem 1 follows from Lemma 9. In Section 3.2, we briefly discuss the proof of the first half of Lemma 9 (completeness), with a proof given in the full version. In Section 3.3, we prove the second half (soundness).

3.2 Completeness

Let Φ be a label-cover instance as defined in Lemma 9 such that $OPT(\Phi) = 1$. We prove the first half of Lemma 9 by providing a swap sequence for $K(\Phi)$ using $(\frac{13}{4}d^2 + d)|X \cup Y|$ swaps. We describe the procedure in detail, and prove the first half of Lemma 9 in the full version.

3.3 Soundness

In this section, we will prove the second half of Lemma 9: if $OPT(\Phi) < \gamma$, then $OPT(K(\Phi)) > \left(\frac{7-\gamma}{2}d^2 - 5d\right)|X \cup Y|$.

Let Φ be a label-cover instance such that $OPT(\Phi) < \gamma$, and let S_{OPT} denote the optimal sequence of swaps on $K(\Phi)$. For a token $t \in T$, consider the subsequence of S_{OPT} in which t is one of the tokens being swapped. This subsequence traces out a walk in G. Consider the path from $f_1(t)$ to $f_2(t)$ obtained by removing all the closed sub-walks from this walk. We denote this path $\mathbf{Swap}(t)$.

We now partition the assignment tokens into two types: *detour tokens* and *non-detour* (assignment) tokens.

▶ **Definition 10** (Detour token)**.** *A token t is a* detour token *if:*

- *$f_1(t)$ is an assignment vertex.*
- ***Swap**(t) has more than $4d$ edges.*

We will refer to all other assignment tokens as *non-detour tokens*. This allows us to introduce the following notation: det_L (resp. det_R) is the number of detour tokens t such that $f_1(t)$ is a left (resp. right) assignment vertex. We will denote by Det the set of detour tokens, and $Ndet$ the set of non-detour tokens.

We obtain lower bounds on two quantities: B_{sat}, the number of swaps occurring within satisfaction paths, and B_{Gad}, the number of swaps occurring within gadgets.

$$B_{sat} \geq (d^2 - 3d)|X \cup Y| + \frac{d}{2}(det_L + det_R) \tag{1}$$

$$B_{Gad} > \left(\frac{5-\gamma}{2}d^2 - 2d\right)|X \cup Y| - \frac{d}{2}(det_L + det_R) \tag{2}$$

Combining these inequalities proves the second half of Lemma 9.

3.3.1 Swaps on satisfaction paths

First, we prove inequality 1.

▶ **Observation 11.** *If t is a detour token, then **Swap**(t) contains at least three satisfaction paths as subpaths.*

Observation 11 follows from the construction of G; any path of length greater than $4d$ with a left assignment vertex and a right assignment vertex as endpoints traverses at least three satisfaction paths. Therefore, we can associate to each detour token t a sequence of at least three satisfaction paths. We refer to the path(s) in this sequence which are not the first or last path as the *intermediate path(s)* of t. Each detour token has at least one intermediate path.

▷ Claim 12. S_{OPT} contains at least $\frac{d}{2}(det_L + det_R)$ swaps where at least one token is a detour token visiting one of its intermediate paths.

Proof. There are $det_L + det_R$ detour tokens, each of which participates in at least d swaps on intermediate paths. Each swap involves at most 2 tokens, hence the bound of $\frac{d}{2}(det_L + det_R)$. ◁

Furthermore, let $ext(t)$ denote the number of swaps that t participates in on satisfaction paths that are not intermediate paths. For $t \in Det$, $ext(t) \geq 2d$. For $t \in Ndet$, $ext(t) \geq d$. Unfortunately, we cannot simply add these quantities together to obtain a lower bound on the number of swaps on satisfaction paths. This is because a swap may be between two different assignment tokens, so adding these quantities would risk double-counting. However, we can obtain an upper-bound on the amount of double-counting that can take place, using the following definition.

▶ **Definition 13** (Efficient satisfaction swap)**.** *A swap between assignment tokens t_1 and t_2 is an* efficient satisfaction swap *if one of the following is the case:*

1. *neither t_1 nor t_2 is a detour token,*
2. *one of t_1 or t_2 is not a detour token, the other is a detour token not on an intermediate path, and the swap is contained in **Swap**(t), where t is the detour token, or*
3. *t_1 and t_2 are detour tokens, neither is on an intermediate path, and the swap is contained in **Swap**(t), where t is t_1 or t_2.*

Let k_{sat} denote the number of efficient satisfaction swaps in S_{OPT}. Combining Definition 13 with Claim 12, we obtain the following inequality:

$$B_{sat} \geq \frac{d}{2}(det_L + det_R) + \sum_{t \in Det} ext(t) + \sum_{t \in Ndet} ext(t) - k_{sat}$$

which implies that

$$B_{sat} \geq \frac{d}{2}\left(det_L + det_R\right) + 2d(det_L + det_R) + d\Big(d|X \cup Y| - (det_L + det_R)\Big) - k_{sat}. \quad (3)$$

We give an upper bound on k_{sat}. We do so by proving individual upper bounds on each of the three types of efficient satisfaction swaps outlined in Definition 13.

Swaps of type 1: *neither t_1 not t_2 is a detour token.* Because the label-cover graph is simple, each edge $(x, y) \in E$ is mapped by our reduction to a unique pair of assignment vertices, those being $\mathbf{asg}(x, y)$ and $\mathbf{asg}(y, x)$. Therefore, any satisfaction path $\mathbf{sat}(x, \sigma_x, y, \sigma_y)$ is traversed by at most two non-detour assignment tokens, those being $f_1^{-1}(\mathbf{asg}(x, y))$ and $f_1^{-1}(\mathbf{asg}(y, x))$. It follows that any non-detour token can swap on a satisfaction path with at most one other non-detour token Because there are $d|X \cup Y| - (det_L + det_R)$ non-detour assignment tokens, we obtain the following upper bound on swaps of type 1:

$$\frac{1}{2}\Big(d|X \cup Y| - (det_L + det_R)\Big).$$

Swaps of type 2: *one of t_1 or t_2 is not a detour token, and the other is a detour token not on an intermediate path.* Again, any satisfaction path $\mathbf{sat}(x, \sigma_x, y, \sigma_y)$ is traversed by at most two non-detour assignment tokens, those being $f_1^{-1}(\mathbf{asg}(x, y))$ and $f_1^{-1}(\mathbf{asg}(y, x))$. A detour token moving across $\mathbf{sat}(x, \sigma_x, y, \sigma_y)$ can swap with only one, depending on the direction in which it is moving. Because each detour token traverses exactly two non-intermediate satisfaction paths, the number of swaps of type 2 is at most:

$$2(det_L + det_R)$$

Swaps of type 3: *t_1 and t_2 are detour tokens, but neither is on an intermediate path.* Each detour token participates in at most a combined $2d$ swaps on the first and last satisfaction paths that it visits. Each of these swaps involves at most 2 such tokens, implying that the number of swaps of type 3 is at most:

$$d(det_L + det_R).$$

Combining these bounds with inequality 3, we are left with the following lower bound on B_{sat}:

$$\begin{aligned}
B_{sat} &\geq \frac{d}{2}\left(det_L + det_R\right) + d\Big(d|X \cup Y| - (det_L + det_R)\Big) + 2d(det_L + det_R) \\
&\quad - \frac{1}{2}\Big(d|X \cup Y| - (det_L + det_R)\Big) - 2(det_L + det_R) - d(det_L + det_R) \\
&= d^2|X \cup Y| + \frac{3d}{2}(det_L + det_R) - \frac{d}{2}|X \cup Y| - \left(d + \frac{5}{2}\right)(det_L + det_R) \\
&= \left(d^2 - \frac{d}{2}\right)|X \cup Y| + \left(\frac{d-5}{2}\right)(det_L + det_R) \\
&\geq \left(d^2 - 3d\right)|X \cup Y| + \frac{d}{2}(det_L + det_R)
\end{aligned}$$

as desired.

3.3.2 Swaps on gadgets

Next, we prove inequality 2:

$$B_{Gad} > \left(\frac{5-\gamma}{2}d^2 - 2d\right)|X \cup Y| - \frac{d}{2}(det_L + det_R).$$

Any assignment token t participates in at least $3d-2$ swaps within label paths: $d-1$ swaps on a left label path and $2d-1$ swaps on a right label path. Moreover, we can identify the two label paths, one in a left gadget and one in a right gadget, which come at either end of $\mathbf{Swap}(t)$. We refer to these paths as the *first* and *last legs* of t. Note that the first leg of t comes in the gadget containing t's start vertex, and the last leg comes in the gadget containing t's target vertex.

A swap which corresponds to an edge in the first leg of one token t_1 and the last leg of another token t_2 we will call an *efficient gadget swap*. Note that such a swap appears in both $\mathbf{Swap}(t_1)$ and $\mathbf{Swap}(t_2)$. We denote the number of efficient gadget swaps in S_{OPT} by k_{Gad}, and we denote the number of efficient gadget swaps occurring within the gadget $\mathbf{Gad}(v)$ by $k_{Gad}(v)$. We observe that

$$B_{Gad} \geq (3d^2 - 2d)|X \cup Y| - k_{Gad} \tag{4}$$

and we seek an upper bound on k_{Gad} accordingly.

To this end, we define the *outflow* of a label path p to be the number of tokens whose first leg is p, and the *inflow* of p to be the number of tokens whose last leg is p. For $\mathbf{Gad}(v) \in \mathbf{L} \cup \mathbf{R}$, we call the label path in $\mathbf{Gad}(v)$ with the maximum outflow (resp. inflow) the *out-dominant* (resp. *in-dominant*) label path of $\mathbf{Gad}(v)$. We denote by $out(\mathbf{Gad}(v))$ (resp. $in(\mathbf{Gad}(v))$) the outflow (resp. inflow) of the out-dominant (resp. in-dominant) label path in $\mathbf{Gad}(v)$. We obtain two inequalities.

▷ Claim 14. For any gadget $\mathbf{Gad}(v)$,
- $k_{Gad}(v) \leq d(out(\mathbf{Gad}(v)))$
- $k_{Gad}(v) \leq d(in(\mathbf{Gad}(v)))$.

Proof. We prove the first inequality. Let $t \in T$ be an assignment token. Suppose $f_2(t)$ is in $\mathbf{Gad}(v)$. Then t participates in at most $out(\mathbf{Gad}(v))$ efficient gadget swaps on $\mathbf{Gad}(v)$, as at most $out(\mathbf{Gad}(v))$ such swaps can occur on the label path taken by t. There are d tokens t where $f_2(t)$ is an assignment vertex in $\mathbf{Gad}(v)$, hence the bound of $d(out(\mathbf{Gad}(v)))$. The proof of the second inequality is symmetric. ◁

▷ Claim 15. If $OPT(\Phi) < \gamma$, then

$$\sum_{x \in X} out(\mathbf{Gad}(x)) + \sum_{y \in Y} in(\mathbf{Gad}(y)) < \left(\frac{1+\gamma}{2}\right) d|X \cup Y| + det_L$$

$$\sum_{x \in X} in(\mathbf{Gad}(x)) + \sum_{y \in Y} out(\mathbf{Gad}(y)) < \left(\frac{1+\gamma}{2}\right) d|X \cup Y| + det_R$$

Proof. We prove the first half of the claim; the proof of the second half is symmetric. Suppose for contradiction the first inequality does not hold. The left side of the inequality counts two quantities: (a) the total outflow from the out-dominant label path of each left gadget, plus (b) the total inflow to the in-dominant label path of each right gadget. Each left assignment token can contribute at most 1 to (a) and at most 1 to (b). Because $d|X \cup Y|$ is the total number of assignment tokens and the graph is regular (so $|X| = |Y|$), there are exactly $(d/2)|X \cup Y|$ left assignment tokens. As a result, even if the maximum number of left assignment tokens are contributing to only one of (a) or (b), at least $\frac{\gamma}{2}(d|X \cup Y|) + det_L$ contribute to both. Therefore, there are at least $\frac{\gamma}{2}(d|X \cup Y|)$ non-detour tokens starting in left gadgets which have an out-dominant label path as a first leg and an in-dominant label path as a last leg. Let t be such a token. Because t is a non-detour token, there is a

satisfaction path between t's first and last legs. By our construction, the constraint in the label cover instance associated with t is satisfied by the labelling corresponding to t's first and last legs. Consider a labelling of Φ which assigns to $x \in X$ the label corresponding to the out-dominant label path of $\mathbf{Gad}(x)$, and to $y \in Y$ the label corresponding to the in-dominant label path of $\mathbf{Gad}(y)$. There are at least $\gamma|E|$ constraints satisfied by this labelling, which is a contradiction. ◁

We average the inequalities in Claim 14 and rearrange some terms to obtain the new bound for any $\mathbf{Gad}(v)$:

$$\frac{k_{Gad}(v)}{d} \leq \frac{out(\mathbf{Gad}(v)) + in(\mathbf{Gad}(v))}{2}$$

and we average the two inequalities in Claim 15 to obtain

$$\sum_{v \in X \cup Y} \frac{out(\mathbf{Gad}(v)) + in(\mathbf{Gad}(v))}{2} < \left(\frac{1+\gamma}{2}\right) d|X \cup Y| + \frac{det_L + det_R}{2}.$$

Combining these bounds, we arrive at the following inequality:

$$\sum_{v \in X \cup Y} \frac{k_{Gad}(v)}{d} < \frac{1+\gamma}{2} d|X \cup Y| + \frac{det_L + det_R}{2}$$

which implies

$$k_{Gad} < \frac{1+\gamma}{2} d^2|X \cup Y| + \frac{d}{2}(det_L + det_R)$$

which, combined with inequality 4:

$$B_{Gad} \geq (3d^2 - 2d)|X \cup Y| - k_{Gad}$$

implies inequality 2:

$$B_{Gad} > \left(\frac{5-\gamma}{2} d^2 - 2d\right)|X \cup Y| - \frac{d}{2}(det_L + det_R).$$

4 Hardness for 0/1-Weighted Token Swapping

We prove the following theorem:

▶ **Theorem 2.** *For any constant $\epsilon > 0$, it is NP-hard to approximate weighted token swapping with $\{0, 1\}$ weights on n vertices within a factor of $(1 - \epsilon) \cdot \ln n$.*

We proceed via a reduction from the Set Cover problem, defined below.

▶ **Definition 16** (Set Cover). *An instance of Set-Cover $\Phi = (U, \{S_i\}_{1 \leq i \leq k})$ consists of a finite universe set U, together with subsets $S_1, ..., S_k \subseteq U$. We seek to find the size of the smallest collection of subsets in $\{S_1, ..., S_k\}$ so that each element of U is in at least one subset.*

Dinur and Steurer [13] prove the following theorem:

▶ **Theorem 17.** *For any constant $\delta > 0$, it is NP-hard to approximate Set Cover within a factor of $(1 - \delta) \cdot \ln(|U| + k)$.*

We give a gap-preserving reduction from Set Cover to Weighted Token Swapping. We describe the reduction in the full version, proving the desired lower-bound.

5 Additional barriers to improved algorithms

In this section, we discuss two barriers to obtaining a $(4-\epsilon)$-approximation for unweighted token swapping on graphs. The first barrier (Theorem 3 is against a broad class of algorithms that encompasses all known approximation algorithms for token swapping. The second barrier (Theorem 4) shows that the technique used to prove approximation factors for existing algorithms cannot be used to prove an approximation factor of $4-\epsilon$ for any $\epsilon > 0$. The proof of Theorem 4 is given in the full version of the paper.

5.1 A barrier against a general class of algorithms

We define the following property for swap sequences on a Token Swapping instance.

▶ **Definition 18** (Local Optimality). *A swap sequence is* locally optimal *if every swap between tokens t_1 and t_2 does not move both t_1 and t_2 further from their destinations.*

All known algorithms for token swapping on trees or on general graphs work by returning the length of a swap sequence that is locally optimal.

▶ **Theorem 3.** *For any $\delta > 0$, there exists a token swapping instance K so that for any locally optimal swap sequence of length k, $(4-\delta) \cdot OPT(K) \leq k$.*

We construct a token swapping instance K as follows. See Figure 2. Let p and q be parameters to be decided later. We can assume p is even. We begin with a cycle C^{out} of length $p \cdot q$. Starting with an arbitrary vertex, we label the vertices $v_0, ..., v_{p \cdot q-1}$ in one direction (say, clockwise) around the cycle. We partition the vertices of C^{out} into p consecutive "segments". For any $0 \leq i \leq p-1$, the vertices $\{v_{iq}, ..., v_{iq+(q-1)}\}$ form the i-th segment. If i is even, we call such a segment an *even segment*; else it is an *odd segment*. For any j so that v_j is in an even segment, the token starting on v_j has target vertex $v_{j+2q} \pmod{pq}$. If v_j is in an odd segment, the token starting on v_j has target vertex $v_{j-2q} \pmod{pq}$. That is, each token in an even segment wants to move to the corresponding vertex in the next even segment in the clockwise direction, while each token in an odd segment wants to move to the next odd segment over in the counter-clockwise direction. For each $0 \leq j \leq pq$, we add a path P_j of $2q-2$ edges between v_j and v_{j+2q}. All the vertices on this path (except the endpoints in C^{out}) wish to stay on their start vertices. We call these paths *inner paths*. We call the vertices in C^{out} *outer vertices*, and all other vertices *inner vertices*. Tokens that begin on outer vertices (which also have destinations on outer vertices) we call *outer tokens*; other tokens are *inner tokens*. Outer tokens with start and target vertices in even segments we will call *even tokens*; other outer tokens we will call *odd tokens*. Note that each outer token begins on a vertex connected to its target vertex by an inner cycle. The outer cycle, together with a single inner cycle, is depicted in Figure 2.

▷ Claim 19. $OPT(K) \leq pq^2$.

Proof. Here is a swap sequence bringing every token to its destination vertex:

1. For $j = 0, ..., pq-1$:
   a. If $f_1^{-1}(v_j)$ is an odd token, bubble it counter-clockwise around C^{out} across q edges.
2. Again, for $j = 0, ..., pq-1$:
   a. If $f_1^{-1}(v_j)$ is an odd token, bubble it counter-clockwise around C^{out} across q edges.

We check that the above swap sequence brings every token to its target vertex. No swap occurs on an edge in an inner cycle, so none of the inner tokens are displaced. Suppose $f^{-1}(v_j)$ (the token starting on v_j) is an odd outer token. It is not difficult to see that when

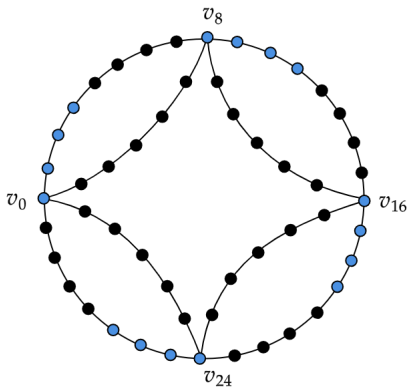


Figure 2 An (incomplete) graph for a token swapping instance where $p = 8$, $q = 4$. The diagram depicts the outer cycle and one inner cycle, but leaves out the remaining inner cycles for legibility. In the outer cycle, vertices in even segments are colored blue, while those in odd segments are colored black. The token starting on v_0 has target v_8, the token starting on v_8 has target v_{16}, the token starting on v_{16} has target v_{24}, and the token starting on v_{24} has target v_0.

$f^{-1}(v_j)$ is bubbled q edges counter-clockwise, it only swaps with even tokens (this is because any odd token from the same segment was already bubbled q edges counter-clockwise during a previous iteration). Thus, each odd token is bubbled counter-clockwise $2q$ times, and each even token is bubbled clockwise $2q$ times. This brings every token to its target node. ◁

▷ Claim 20. Let S be the shortest locally optimal swap sequence bringing every token to its destination, and let k be the length of S. Then $4pq^2 - 5pq - 8q^2 \leq k$.

We observe that for $0 \leq j \leq 2q - 1$, the inner paths $P_j, P_{j+2q}, P_{j+4q}, ..., P_{j+q(p-1)}$ form a cycle, which we will call the j-th *inner cycle*, denoted C^{in}_j. Moreover, every token begins on a vertex in the same inner cycle as its target vertex. Claim 20 will follow from the next two claims.

▷ Claim 21. Let $a, b \in V$ be in the same inner cycle C^{in}_j, and let $P_{a,b}$ be a path from a to b containing at least one edge not in C^{in}_j. Then $\text{dist}(a, b) + 2 \leq |P_{a,b}|$, where $|P_{a,b}|$ is the number of edges in $P_{a,b}$.

Proof. We will prove the claim in two steps. First, we will modify $P_{a,b}$ to obtain a path $P'_{a,b}$ which is the same length as $P_{a,b}$ and which does not contain any edges from inner cycles other than C^{in}_j, but which contains at least one edge from C^{out}. Then, we will show that any path from a to b which contains edges from (and only from) both C^{in}_j and C^{out} has length at least $\text{dist}(a, b) + 2$.

Suppose $P_{a,b}$ contains edges outside of C^{in}_j. If $P_{a,b}$ does not contain edges from a different inner cycle, then we set $P'_{a,b}$ to $P_{a,b}$ and move on to the argument in the next paragraph. Otherwise, $P_{a,b}$ contains an edge e from a different inner cycle. Let ℓ be such that C^{in}_ℓ is the inner cycle containing e. Suppose e is, in particular, the first edge to appear in $P_{a,b}$ which is in an inner cycle other than C^{in}_j. Therefore, e is incident to a vertex $v_{\ell'}$ so that $\ell' \equiv \ell \pmod{2q}$, and e is contained in either $P_{\ell'}$ or $P_{\ell'-2q}$. We can assume by symmetry that e is contained in $P_{\ell'}$. Because $P_{a,b}$ is a shortest path, it is simple. By the construction of the graph then, $P_{a,b}$ contains the entirety of $P_{\ell'}$. Let $v_{j'}$ be the last vertex in C^{in}_j preceding $v_{\ell'}$ in $P_{a,b}$. Because e is incident to an outer vertex, the path between $v_{j'}$ and $v_{\ell'}$ only has edges in C^{out}. The length of the sub-path of $P_{a,b}$ from the appearance of $v_{j'}$ to $v_{\ell'+2q}$ is

then $|\ell' - j'| + 2q - 2$. We replace this sub-path with the following path: the shortest path in C^{in}_j from $v_{j'}$ to $v_{j'+2q}$, then the shortest path in C^{out} from $v_{j'+2q}$ to $v_{\ell'+2q}$. The length of this path is at most $|\ell' - j'| + 2q - 2$, so we have not increased the length of $P_{a,b}$, but we have decreased the number of edges from an inner cycle other than C^{in}_j. We repeat this procedure until we obtain a path $P'_{a,b}$ which does not contain any edges from inner cycles other than C^{in}_j.

We have a simple path $P'_{a,b}$ which contains edges only from C^{in}_j and C^{out}, and at least one edge from C^{out}. Then there exists a consecutive sub-path in $P'_{a,b}$ of edges from C^{out} of length $2q$, as this is the number of edges needed to go from one vertex in C^{in}_j to another while traversing C^{out}. However, this sub-path can be replaced by the sub-path of length $2q - 2$ between its endpoints using only edges from C^{in}_j. Therefore, $\text{dist}(a, b) + 2 \leq |P'_{a,b}| = |P_{a,b}|$. ◁

For a, b in C^{in}_j, and for c not in C^{in}_j which is a neighbor of a, it is the case that $\text{dist}(a, b) < \text{dist}(c, b)$. Otherwise, there would be a path from a to b containing the edge (a, c) of length at most $\text{dist}(a, b) + 1$, even though (a, c) is not in C^{in}_j, which contradicts Claim 21. Because each token starts in the same inner cycle as its target vertex, this implies that any swap across an edge in C^{out} is not locally optimal. We proceed by proving a lower bound on the number of swaps needed to bring all the tokens in C^{in}_j to their target vertex without leaving C^{in}_j.

▷ Claim 22. For any j, the number of swaps needed to bring all of the tokens on C^{in}_j to their target vertices while swapping only along edges in C^{in}_j is greater than $2pq - 5p/2 - 4q$.

Proof. Suppose without loss of generality that v_j is in an even segment; by our construction, all the other outer vertices in C^{in}_j are also in even segments. Moreover, each outer vertex in C^{in}_j begins with a token which wants to move to the closest outer vertex in C^{in}_j in the clockwise direction, of distance $2q - 2$ away. All inner tokens want to stay on their start vertices. Note that C^{in}_j has length $p(q - 1)$, and contains $p/2$ outer vertices.

In a given swap, one token in C^{in}_j is moved clockwise, and the other counter-clockwise. For a token t, let $\mathit{clock}(t)$ be the number of swaps in the optimal swap sequence on C^{in}_j in which t is moved clockwise, and $\mathit{counter}(t)$ the number of times its moved counter-clockwise.

Let t_{in} be an inner token in C^{in}_j. Because t_{in} has the same start and target, $\mathit{clock}(t_{in}) - \mathit{counter}(t_{in}) \equiv 0 \pmod{p(q-1)}$. If t_{out} is an outer vertex in C^{in}_j, then $\mathit{clock}(t_{out}) - \mathit{counter}(t_{out}) \equiv 2q - 2 \pmod{p(q-1)}$. Because each swap moves one token clockwise and one token counter-clockwise, $\sum_{t_{in} \in C^{in}_j} \mathit{clock}(t_{in}) - \mathit{counter}(t_{in}) = 0$. It follows there is a token $t_{counter} \in C^j_{in}$ where $\mathit{counter}(t_{counter}) > \mathit{clock}(t_{counter})$ (else, each token in C^j_{in} would end on its start vertex). By our construction, the distance from $t_{counter}$'s start vertex to its target in the counter-clockwise direction is at least $p(q-1) - 2q + 2$, so $\mathit{counter}(t_{counter}) - \mathit{clock}(t_{counter}) \geq p(q-1) - 2q + 2$.

If there is an additional token $t'_{counter}$ so that $p(q-1) - 2q + 2 \leq \mathit{counter}(t')$, then there are at least $2(p(q-1) - 2q) > 2pq - 5p/2 - 4q$ swaps in the optimal swap sequence, as desired. Otherwise, there are at least $p/2 - 1$ outer tokens in C^{in}_j which are not moved counter-clockwise at least $p(q-1) - 2q$ times. Each is involved in at least $2q - 2$ swaps where it is the token being moved clockwise. Moreover, at most 1 of these swaps is with $t_{counter}$, because if a pair of tokens swap at least twice, then both swaps can be removed to yield an equivalent swap sequence. This results in a total of at least

$$\begin{aligned}(p/2-1)(2q-3) + \mathit{counter}(t_{counter}) &\geq (p/2-1)(2q-3) + p(q-1) - 2q \\ &> 2pq - 5p/2 - 4q\end{aligned}$$

swaps, as desired. ◁

Because there are $2q$ inner cycles, the total number of swaps taken by a sequence with only locally optimal swaps is at least $2q(2pq - 5p/2 - 4q) = 4pq^2 - 5pq - 8q^2$, proving Claim 20. We set p, q to be large enough so that $4pq^2$ overtakes the smaller-order terms, so that for the parameter δ,

$$4 - \delta < (4pq^2 - 5pq - 8q^2)/pq^2 < 4.$$

This proves Theorem 3.

References

1 Oswin Aichholzer, Erik D. Demaine, Matias Korman, Anna Lubiw, Jayson Lynch, Zuzana Masárová, Mikhail Rudoy, Virginia Vassilevska Williams, and Nicole Wein. Hardness of token swapping on trees. In *30th annual European Symposium on Algorithms*, volume 244 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 3, 15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.3.

2 S.B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, 1989. doi:10.1109/12.21148.

3 Noga Alon, F. R. K. Chung, and R. L. Graham. Routing permutations on graphs via matchings. *SIAM J. Discrete Math.*, 7(3):513–530, 1994. doi:10.1137/S0895480192236628.

4 Sanjeev Arora and Carsten Lund. Hardness of approximation. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 10, pages 399–446. PWS Publishing, Boston, 2004.

5 Avah Banerjee, Xin Liang, and Rod Tohid. Locality-aware qubit routing for the grid architecture. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 607–613. IEEE, 2022.

6 Indranil Banerjee and Dana Richards. New results on routing via matchings on graphs. In *Fundamentals of computation theory*, volume 10472 of *Lecture Notes in Comput. Sci.*, pages 69–81. Springer, Berlin, 2017. doi:10.1007/978-3-662-55751-8_7.

7 Aniruddha Bapat, Andrew M Childs, Alexey V Gorshkov, and Eddie Schoute. Advantages and limitations of quantum routing. *PRX Quantum*, 4(1):010313, 2023.

8 Ahmad Biniaz, Kshitij Jain, Anna Lubiw, Zuzana Masárová, Tillmann Miltzow, Debajyoti Mondal, Anurag Murty Naredla, Josef Tkadlec, and Alexi Turcotte. Token swapping on trees. *Discrete Math. Theor. Comput. Sci.*, 24(2):Paper No. 9, 37, 2022. doi:10.46298/DMTCS.8383.

9 Édouard Bonnet, Tillmann Miltzow, and Paweł Rzążewski. Complexity of token swapping and its variants. *Algorithmica*, 80(9):2656–2682, 2018. doi:10.1007/s00453-017-0387-0.

10 Arthur Cayley. Lxxvii. note on the theory of permutations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 34(232):527–529, 1849.

11 Andrew M. Childs, Eddie Schoute, and Cem M. Unsal. Circuit transformations for quantum architectures. In *14th Conference on the Theory of Quantum Computation, Communication and Cryptography*, volume 135 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 3, 24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

12 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: reconfiguring a swarm of labeled robots with bounded stretch. *SIAM J. Comput.*, 48(6):1727–1762, 2019. doi:10.1137/18M1194341.

13 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 624–633, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591884.

14 Laurent Gourvès, Julien Lesca, and Anaëlle Wilczynski. Object allocation via swaps along a social network. In *26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 213–219, 2017. doi:10.24963/IJCAI.2017/31.

15 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Algorithmic theory of qubit routing. In *Algorithms and data structures*, volume 14079 of *Lecture Notes in Comput. Sci.*, pages 533–546. Springer, Cham, 2023. doi:10.1007/978-3-031-38906-1_35.

16 Mark R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoret. Comput. Sci.*, 36(2-3):265–289, 1985. doi:10.1016/0304-3975(85)90047-7.

17 Jun Kawahara, Toshiki Saitoh, and Ryo Yoshinaka. The time complexity of permutation routing via matching, token swapping and a variant. *J. Graph Algorithms Appl.*, 23(1):29–70, 2019. doi:10.7155/jgaa.00483.

18 Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and hardness of token swapping. In *24th Annual European Symposium on Algorithms (ESA)*, volume 57 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 66, 15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

19 Abtin Molavi, Amanda Xu, Martin Diges, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. Qubit mapping and routing via maxsat. In *2022 55th IEEE/ACM international symposium on Microarchitecture (MICRO)*, pages 1078–1091. IEEE, 2022. doi:10.1109/MICRO56248.2022.00077.

20 Igor Pak. Reduced decompositions of permutations in terms of star transpositions, generalized Catalan numbers and k-ary trees. *Discrete Math.*, 204(1-3):329–335, 1999. doi:10.1016/S0012-365X(98)00377-X.

21 Frederick J Portier and Theresa P Vaughan. Whitney numbers of the second kind for the star poset. *European Journal of Combinatorics*, 11(3):277–288, 1990. doi:10.1016/S0195-6698(13)80127-8.

22 Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998. doi:10.1137/S0097539795280895.

23 Bruno Schmitt, Mathias Soeken, and Giovanni De Micheli. Symbolic algorithms for token swapping. In *2020 IEEE 50th International Symposium on Multiple-Valued Logic—ISMVL 2020*, pages 28–33. IEEE Computer Soc., Los Alamitos, CA, 2020. doi:10.1109/ISMVL49045.2020.00-34.

24 Asim Sharma and Avah Banerjee. Noise-aware token swapping for qubit routing. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 82–88. IEEE, 2023. doi:10.1109/QCE57702.2023.00018.

25 Zhizhang Shen and Ke Qiu. On the Whitney numbers of the second kind for the star poset: comment on [European J. Combin. **11** (1990), no. 3, 277–288; mr1059558] by F. J. Portier and T. P. Vaughan. *European J. Combin.*, 29(7):1585–1586, 2008. doi:10.1016/j.ejc.2007.11.026.

26 Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. Qubit allocation as a combination of subgraph isomorphism and token swapping. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–29, 2019. doi:10.1145/3360546.

27 Pavel Surynek. Finding optimal solutions to token swapping by conflict-based search and reduction to sat. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 592–599. IEEE, 2018. doi:10.1109/ICTAI.2018.00096.

28 Pavel Surynek. Multi-agent path finding with generalized conflicts: An experimental study. In *International Conference on Agents and Artificial Intelligence*, pages 118–142. Springer, 2019. doi:10.1007/978-3-030-37494-5_7.

29 Caio Henrique Segawa Tonetti, Vinicius Fernandes dos Santos, and Sebastián Urrutia. Polynomial time algorithms for the token swapping problem on cographs. *RAIRO Oper. Res.*, 58(1):441–455, 2024. doi:10.1051/ro/2023134.

30 Theresa P. Vaughan. Bounds for the rank of a permutation on a tree. *J. Combin. Math. Combin. Comput.*, 10:65–81, 1991.

31 Theresa P. Vaughan. Factoring a permutation on a broom. *J. Combin. Math. Combin. Comput.*, 30:129–148, 1999.

32 Theresa P. Vaughan and Frederick J. Portier. An algorithm for the factorization of permutations on a tree. *J. Combin. Math. Combin. Comput.*, 18:11–31, 1995.

33 Friedrich Wagner, Andreas Bärmann, Frauke Liers, and Markus Weissenbäck. Improving quantum computation by optimized qubit routing. *J. Optim. Theory Appl.*, 197(3):1161–1194, 2023. doi:10.1007/s10957-023-02229-w.

34 Katsuhisa Yamanaka, Erik D. Demaine, Takashi Horiyama, Akitoshi Kawamura, Shin-ichi Nakano, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Ryuhei Uehara, and Takeaki Uno. Sequentially swapping colored tokens on graphs. *J. Graph Algorithms Appl.*, 23(1):3–27, 2019. doi:10.7155/jgaa.00482.

35 Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theoret. Comput. Sci.*, 586:81–94, 2015. doi:10.1016/J.TCS.2015.01.052.

36 Gaku Yasui, Kouta Abe, Katsuhisa Yamanaka, and Takashi Hirayama. Swapping labeled tokens on complete split graphs. *Inf. Process. Soc. Japan. SIG Tech. Rep*, 14:1–4, 2015.

37 Louxin Zhang. Optimal bounds for matching routing on trees. *SIAM J. Discrete Math.*, 12(1):64–77, 1999. doi:10.1137/S0895480197323159.

Streaming Diameter of High-Dimensional Points

Magnús M. Halldórsson
Department of Computer Science, Reykjavik University, Iceland

Nicolaos Matsakis
Department of Computer Science, Reykjavik University, Iceland

Pavel Veselý
Computer Science Institute of Charles University, Prague, Czech Republic

Abstract

We improve the space bound for streaming approximation of Diameter but also of Farthest Neighbor queries, Minimum Enclosing Ball and its Coreset, in high-dimensional Euclidean spaces. In particular, our deterministic streaming algorithms store $\mathcal{O}(\varepsilon^{-2}\log(\frac{1}{\varepsilon}))$ points. This improves by a factor of ε^{-1} the previous space bound of Agarwal and Sharathkumar (SODA 2010), while retaining the state-of-the-art approximation guarantees, such as $\sqrt{2}+\varepsilon$ for Diameter or Farthest Neighbor queries, and also offering a simpler and more complete argument. Moreover, we show that storing $\Omega(\varepsilon^{-1})$ points is necessary for a streaming $(\sqrt{2}+\varepsilon)$-approximation of Farthest Pair and Farthest Neighbor queries.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases streaming algorithm, farthest pair, diameter, minimum enclosing ball, coreset

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.58

Related Version *Full Version*: `https://arxiv.org/abs/2505.16720`

Funding *Magnús M. Halldórsson*: Partially supported by Icelandic Research Fund grant 2511609.
Nicolaos Matsakis: Partially supported by Icelandic Research Fund grant 2511609 and by Czech Science Foundation project 22-22997S.
Pavel Veselý: Partially supported by Czech Science Foundation project 22-22997S, by ERC-CZ project LL2406 of the Ministry of Education of Czech Republic, and by Center for Foundations of Modern Computer Science (Charles Univ. project UNCE 24/SCI/008).

Acknowledgements We would like to thank Martin Tancer for helpful discussions on high-dimensional geometry.

1 Introduction

In the streaming model, the input data is assumed to be vast and must be processed using limited memory in one or a few passes. Therefore, streaming algorithms "sketch" the input, yielding a small data structure that still accurately preserves desired properties. The research on streaming algorithms has been remarkably fruitful and we now have optimal or near-optimal algorithms for counting distinct elements, frequency moments, quantiles and a plethora of other problems (see [14] for a comprehensive exposition).

We focus on high-dimensional geometric streams, where the input S consists of points in $\mathbb{R}^d$ for d large, a topic of recent interest, e.g., [28, 17, 26, 15, 25, 11, 10, 12, 23]. Extent measures, such as the Diameter or the Minimum Enclosing Ball, are fundamental statistics of a set of points, having a body of work in both streaming and non-streaming settings [2, 22, 19, 3, 8, 7].

In an influential work, Agarwal and Sharathkumar [5] gave a streaming algorithmic framework for several high-dimensional extent problems. Their Blurred Ball Cover data structure maintains a collection of $\mathcal{O}(\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$ balls, whose union approximately covers the


© Magnús M. Halldórsson, Nicolaos Matsakis, and Pavel Veselý;
licensed under Creative Commons License CC-BY 4.0



input S. It was then used to approximate a number of high-dimensional extent problems. The claim is that each ball is represented by a coreset of $\mathcal{O}(\frac{1}{\varepsilon})$ points of S, for a total space of $\mathcal{O}(\frac{1}{\varepsilon^3}\log(\frac{1}{\varepsilon}))$ points. It appears though that a somewhat higher space bound is needed for the claimed approximations; see Section A.

We give a modified data structure, Guarded Ball Cover, building on [5]. It allows for both a simpler and more complete treatment, and also results in the smaller space bound of $\mathcal{O}(\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$ points. The improved space bound extends to all four applications: approximate Farthest Neighbor queries and for maintaining approximate Farthest Pair (providing an estimate for Diameter), Minimum Enclosing Ball, and Coreset for Minimum Enclosing Ball. This is feasible by storing only a single point per ball, along with its center and radius. Correctness arguments are simplified by also storing the first point of S as a proxy for all points deleted later from memory. We retain the approximation guarantees of all four applications, as in [5]: $1.22+\varepsilon$ for Minimum Enclosing Ball and $\sqrt{2}+\varepsilon$ for each of Diameter, Farthest Neighbor Queries, and Coreset for Minimum Enclosing Ball.

We also show that $\Omega(\varepsilon^{-1})$ points need to be stored for a comparable $(\sqrt{2}+\varepsilon)$-approximation of Farthest Neighbor queries and also for maintaining $(\sqrt{2}+\varepsilon)$-approximate Farthest Pair. This applies to a computational model where the algorithm must return an input point upon a query and the space is determined by the number of points stored; crucially, once a point is deleted from memory, it cannot be retrieved.

2 Preliminaries

Let S be a multiset of points in $\mathbb{R}^d$ that arrive sequentially in a stream. Upon arrival, each point $p \in S$ is either stored in memory (and, possibly, deleted later) or irrevocably discarded. We assume one-pass streaming algorithms in the insertion-only setting. By $\varepsilon \in (0,1]$ we denote an error parameter and by $\alpha > 1$ an approximation guarantee.

An extent measure of a set of points computes certain statistics of either this set or a geometric shape enclosing it [2]. Let $\|pq\|$ denote the Euclidean distance between points $p \in \mathbb{R}^d$ and $q \in \mathbb{R}^d$. The *Diameter* is the maximum Euclidean distance between any pair of points in S and the *Farthest Pair* $\mathsf{FP}(S)$ is a pair of points of S having Euclidean distance equal to the Diameter. The *Farthest Neighbor* of a point q is a point p of the largest Euclidean distance from q. An α-farthest-neighbor α-$\mathsf{FN}(q)$ of a query point $q \in \mathbb{R}^d$ is a point $x \in S$ such that for every $p \in S$ it is $\|qp\| \leq \alpha \cdot \|xq\|$.

By $B(c(B), r(B))$ we denote a ball centered at point $c(B)$ with radius $r(B)$. The $(1+\varepsilon)$-expansion of $B(c,r)$ is defined as $B(c,(1+\varepsilon)r)$. The *Minimum Enclosing Ball* $\mathsf{MEB}(S)$ is the ball of minimum radius containing all points of S. A ball B is α-$\mathsf{MEB}(S)$ if $r(B) \leq \alpha r(\mathsf{MEB}(S))$ and each point of S is within Euclidean distance $r(B)$ from $c(B)$.

For a set of points S, a *coreset* is a set $S' \subseteq S$ preserving a geometric property of S [21]. A set $S' \subseteq S$ is α-$\mathsf{coreset}(S)$ for MEB if each point of S is contained in the α-expansion of $\mathsf{MEB}(S')$.

We focus on computing $(\sqrt{2}+\varepsilon)$-$\mathsf{FN}(q)$ for any query $q \in \mathbb{R}^d$ and maintaining $(1.22+\varepsilon)$-$\mathsf{MEB}(S)$, $(\sqrt{2}+\varepsilon)$-$\mathsf{coreset}(S)$ and $(\sqrt{2}+\varepsilon)$-$\mathsf{FP}(S)$; these ratios are the same as in [5].

2.1 Related Work

The streaming algorithm of Gonzalez [20] computes a 2-MEB by storing the first point p_1 of S and its farthest neighbor q; the enclosing ball is simply $B(p_1, \|p_1 q\|)$. Zarrabi-Zadeh and Chan [29] improved the guarantee of α-MEB to $\alpha = 1.5$ by giving a one-pass streaming algorithm that stores one ball. They also gave a lower bound of $\frac{\sqrt{2}+1}{2} \approx 1.207$ for the guarantee of any deterministic algorithm for α-MEB that stores only one ball.

Badŏiu et al. [8] showed that the number of coreset points approximating $(1+\varepsilon)$-MEB(S') for a set S' in $\mathbb{R}^d$ does not depend on d. Improved algorithms were given in [24, 6, 7]; however, these algorithms do not work in a streaming fashion.

Preceding the work of Agarwal and Sharathkumar [4], a simple $(1/\sqrt{3})$-approximate two-pass algorithm for the Diameter was given by Egecioglu and Kalantari [16], working in space $\mathcal{O}(d)$.

Following the conference result of [4] that maintains a $(\frac{1+\sqrt{3}}{2}+\varepsilon)$-MEB$(S)$, Chan and Pathak [9] improved this guarantee to $\alpha = 1.22+\varepsilon$ by employing a detailed analysis to this algorithm. Subsequently, Agarwal and Sharathkumar in their journal paper [5] observed that the guarantee of their algorithm is slightly greater than $\frac{\sqrt{2}+1}{2} \approx 1.207$ by presenting an input for $d = 3$.

On the negative side, any randomized streaming algorithm that maintains α-FP(S), α-MEB(S) or α-coreset(S) with probability at least $2/3$ requires $\Omega(\min\{n, \exp(d^{1/3})\})$ space for certain values of α. These values are $\alpha < \sqrt{2}(1-2/d^{1/3})$ for α-FP(S) and α-coreset(S) and $a < (1+\sqrt{2})(1/2-1/d^{1/3})$ for α-MEB(S), as shown by Agarwal and Sharathkumar [5].

For low d, such as $d = O(1)$ or $d = O(\log\log n)$, the lower bounds do not apply as it is possible to maintain $(1+\varepsilon)$-FP(S) or answer $(1+\varepsilon)$-FN(x) queries in a poly-logarithmic space, using an optimized version of the sampling approach of [18]; this applies also to dynamic streams where points may be deleted. For high-dimensional dynamic streams, the best streaming algorithm follows from asymmetric embedding techniques of Indyk [22] but provides $O(1)$-approximation only at the cost of using space polynomial in n.

3 The Guarded Ball Cover

The Guarded Ball Cover is a collection $\mathcal{B}$ of balls that approximately cover all points of S. We represent each ball of $\mathcal{B}$ by a triplet $B = (c, r, q)$, where c is the center of B (possibly $c \notin S$), r is its radius, and q is a point of S inside B. The point q is referred to as the *guard* of B. Our algorithm maintains a coreset Q that consists of the guard points. We treat the first point $p_1 \in S$ specially by always having $p_1 \in Q$.

Let $(1+\varepsilon)\mathcal{B} = \{(c, (1+\varepsilon)r, q) : (c, r, q) \in \mathcal{B}\}$ be the collection of the $(1+\varepsilon)$-expansions of the balls in $\mathcal{B}$. If the arriving point $p \in S$ belongs to $(1+\varepsilon)\mathcal{B}$, then it is discarded. Otherwise, a new ball is added to $\mathcal{B}$. Finally, all balls of too small radius are removed from $\mathcal{B}$.

As the space bound is our primary measure, we assume an exact MEB algorithm, but a good approximation also suffices, specifically within a factor of $1+\varepsilon^2/16$ (using the algorithm of [6] as a subroutine).

Algorithm 1 Algorithm for processing a new point $p \in S$ (excluding the first point p_1).

1: **if** $p \notin (1+\varepsilon)\mathcal{B}$ **then** ▷ If p is outside of the expansions of all guarded balls
2: $(c, r) \leftarrow$ MEB$(Q \cup \{p\})$ ▷ Compute new MEB
3: $\mathcal{B} \leftarrow \{(c, r, p)\} \cup \{(c', r', p') \in \mathcal{B} : r' \geq \varepsilon^2 r/80\}$ ▷ Add new ball, delete small balls
4: $Q \leftarrow \{p_1\} \cup \bigcup_{(c'', r'', q) \in \mathcal{B}} \{q\}$ ▷ Update coreset

The following lemma holds for every MEB computed in line 2 of Algorithm 1:

▶ **Lemma 1** (Lemma 2.2 in [8]). *If $B = B(c, r)$ is the MEB of a set X of points, then any closed half-space containing c also contains a point of X on the boundary of B.*

To analyze the algorithm, we first observe that deleted balls are "guarded" by p_1.

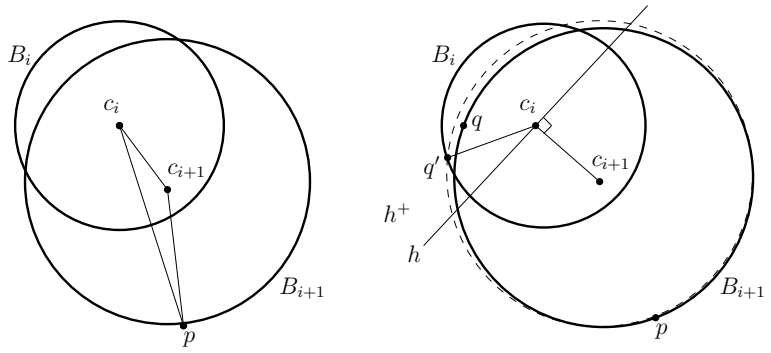


Figure 1 Left: Case $\|c_ic_{i+1}\| \leq 5\varepsilon r_i/6$, Right: Case $\|c_ic_{i+1}\| > 5\varepsilon r_i/6$ of Lemma 3. The ball with dashed boundaries is the MEB of all points, assuming that q' is deleted before B_{i+1} is created.

▶ **Lemma 2.** *Suppose ball B is deleted when a ball of radius r is added to $\mathcal{B}$. Then, B is contained in a ball of radius $\varepsilon^2 r/40$ centered at p_1. Consequently, for any guard point q' that has been deleted up to the current time, it holds that $\|q'p_1\| \leq \varepsilon^2 r/40$.*

Proof. B contains p_1 and when deleted in line 3, it has radius at most $\varepsilon^2 r/80$. Since the guard of B (which is also evicted from memory) is inside B, it is within distance at most $\varepsilon^2 r/40$ from p_1. ◀

The main technical part is to show that the radii of new balls increase exponentially:

▶ **Lemma 3.** *If $B_{i+1} = (c_{i+1}, r_{i+1}, p)$ is added to $\mathcal{B}$ following $B_i = (c_i, r_i, p_i)$, then it holds that $r_{i+1} \geq (1 + \varepsilon^2/8) \cdot r_i$.*

Proof. The proof is similar to that of Lemma 2 in [5] and Claim 2.4 in [8]. We first assume that the exact MEB is computed in line 2 of Algorithm 1. Let Q, $\hat{Q}_D$ be the point sets such that $B_{i+1} = \mathsf{MEB}(Q \cup \{p\})$ and $B_i = \mathsf{MEB}(Q \cup \hat{Q}_D)$, i.e., Q is the coreset just before computing B_{i+1}. Consider two cases:

If $\|c_ic_{i+1}\| \leq 5\varepsilon r_i/6$ then $r_{i+1} \geq \|c_{i+1}p\| \geq \|c_ip\| - \|c_ic_{i+1}\| \geq (1+\varepsilon)r_i - 5\varepsilon r_i/6 \geq (1+\varepsilon^2/6)r_i$ (Figure 1, left), using the triangle inequality and that $p \notin (1+\varepsilon)B_i$ (by line 1).

Otherwise $\|c_ic_{i+1}\| > 5\varepsilon r_i/6$. Then let h be the hyperplane passing through c_i with direction c_ic_{i+1} as its normal and let h^+ be the halfspace bounded by h that does not contain c_{i+1}. There is a point $q' \in (Q \cup \hat{Q}_D) \bigcap h^+$ at Euclidean distance r_i from c_i, by Lemma 1. Then, $\|q'c_{i+1}\| \geq (\|c_ic_{i+1}\|^2 + \|q'c_i\|^2)^{1/2} \geq ((5\varepsilon r_i/6)^2 + r_i^2)^{1/2} \geq (1+\varepsilon^2/4)r_i$ (Figure 1, right), where the first inequality follows from the cosine law. By Lemma 2, there is a point $q \in Q$ such that $\|qq'\| \leq (\varepsilon^2/40)r_i$ (if $q' \in Q$ then $q = q'$, and $q = p_1$ otherwise). Hence, $r_{i+1} \geq \|qc_{i+1}\| \geq \|q'c_{i+1}\| - \|qq'\| \geq (1+\varepsilon^2/5)r_i$.

Finally, if we use a $(1+\varepsilon^2/16)$-approximate MEB, then we still have that $r_{i+1} \geq (1+\varepsilon^2/5)r_i/(1+\varepsilon^2/16) \geq (1+\varepsilon^2/8)r_i$. ◀

Finally, we show that at any time, the $(1+\varepsilon)$-expansion of any deleted ball is contained in the $(1+\varepsilon)$-expansion of each ball created after the deletion of the former ball.

▶ **Lemma 4.** *If ball $\hat{B}$ was deleted then $(1+\varepsilon)\hat{B} \subset (1+\varepsilon)B_i$, for each $B_i \in \mathcal{B}$ created after the deletion of $\hat{B}$.*

Proof. Let $B_i \in \mathcal{B}$. By Lemmas 2 and 3, it is $\hat{B} \subset B(p_1, \varepsilon^2 r(B_i)/40)$; therefore, we have $(1+\varepsilon)\hat{B} \subset B(p_1, \varepsilon r(B_i)/20)$ as $\varepsilon \leq 1$. Since p_1 is in B, it follows that $(1+\varepsilon)\hat{B} \subset (1+\varepsilon/20)B_i$. ◀

Our main result follows from the preceding lemmas:

▶ **Theorem 5.** *$\mathcal{B}$ contains $\mathcal{O}((1/\varepsilon^2)\log(1/\varepsilon))$ balls and $S \subset (1+\varepsilon)\mathcal{B}$.*

Proof. The first claim follows from Lemma 3 and line 3 of Algorithm 1. For the second claim, let $p \in S$. By construction, p is in the $(1+\varepsilon)$-expansion of a ball B that entered $\mathcal{B}$. By Lemma 4, $(1+\varepsilon)B \subset \cup_{B' \in \mathcal{B}}(1+\varepsilon)B'$, whether B was deleted or not. ◀

Differences to the Blurred Ball Cover

The Blurred Ball Cover [5], similarly to the Guarded Ball Cover, initiates the computation of a new ball when an arriving point is not contained in any $(1+\epsilon)$-expansion of a stored ball. The key difference is that in the Blurred Ball Cover, the coreset on the boundary of the new ball, which is returned by the MEB computation and comprises up to $\lceil 1/\epsilon \rceil$ points, is explicitly stored in memory for each ball. (In fact, $\Omega(1/\epsilon^2)$ points may need to be stored for each ball in the Blurred Ball Cover as we discuss in Appendix A.) Coresets belonging to sufficiently small balls are also deleted from the Blurred Ball Cover, though the radius threshold for this deletion is $\approx \varepsilon r$ instead of $\approx \varepsilon^2 r$ in our algorithm. Finally, the first point $p_1 \in S$ is never deleted by the Guarded Ball Cover, contrary to the Blurred Ball Cover which treats p_1 as any other point of S.

Update time

The worst-case update time of our algorithm is $\mathcal{O}(d \cdot \text{poly}(1/\varepsilon))$ when an approximate MEB computation is used [7], which is comparable to that of [5]; the precise update times primarily depend on the MEB computation. (Note that the update times of [5] are also affected by the possible issue mentioned in Appendix A.) We remark that the Blurred Ball Cover of [5] allows for better *amortized* update time as it is possible to perform batched updates, i.e., storing incoming points in a buffer and only running the update procedure when the buffer gets full (see also the second paragraph in Appendix A). This is not possible in our case as we require storing one guard per ball; that is, if we run an update on our sketch with a batch of size C, we may need to store up to C guard points for B_{i+1} if they all end up in the coreset of B_{i+1} (and thus on its boundary).

4 Applications

Farthest Neighbor Queries and Diameter

We largely follow the analysis of [5]. For a query point $x \in \mathbb{R}^d$, the algorithm computes and returns the farthest point in Q: $q' = \arg\max_{q \in Q}\|qx\|$. We show that $\|p'x\| \le (\sqrt{2}+2\varepsilon)\|q'x\|$, where $p' = \arg\max_{p \in S}\|px\|$ is one of the (optimal) farthest points from x.

Let $B_i = (c_i, r_i)$ be the ball in $\mathcal{B}$ of greatest radius that contains p' in its $(1+\varepsilon)$-expansion, which exists by Theorem 5. Applying the triangle inequality, followed by the inequality $x + y \le \sqrt{2(x^2+y^2)}$, we have that

$$\|xp'\| \le \|xc_i\| + \|c_ip'\| \le \|xc_i\| + (1+\varepsilon)r_i \le \sqrt{2}(\|xc_i\|^2 + r_i^2)^{1/2} + \varepsilon r_i \ . \tag{1}$$

By Lemma 1 (for half-space with direction c_ix as normal, c_i on the boundary, and not containing x), when B_i was created, there was a guard z such that: i) $\|zc_i\| = r_i$, ii) $\|xz\| \ge r_i$, and iii) $\angle zc_ix \ge 90°$. By i), iii), and the cosine law, we have

$$\|xz\| \ge (\|xc_i\|^2 + r_i^2)^{1/2} \ . \tag{2}$$

Combining (1) and (2) and using ii), we get that

$$\|xp'\| \leq \sqrt{2}\|xz\| + \varepsilon r_i \leq (\sqrt{2}+\varepsilon)\|xz\| \ . \tag{3}$$

Note that $\|q'x\| \geq r_m/2$, where r_m is the radius of the largest ball in $\mathcal{B}$, as otherwise there is a ball containing Q of radius less than r_m. Also, by Lemma 2, there is a point $w \in Q$ (z or p_1) of distance at most $\varepsilon^2 r_m/80 \leq \varepsilon^2\|q'x\|/40$ from z. By definition of q', $\|wx\| \leq \|q'x\|$. Thus, $\|xz\| \leq \|wx\| + \|wz\| \leq (1+\varepsilon^2/40)\|q'x\|$. Combining this with (3) gives that q' is a $(\sqrt{2}+\varepsilon')$-$\mathsf{FN}(x)$, for $\varepsilon' = 2\varepsilon$.

Finally, for the closely related problem of Diameter (Farthest Pair), we return $\mathsf{FN}(p)$ for each point $p \in S$. If $\bar{p} = \mathsf{FN}(p)$ and $\|p\bar{p}\|$ exceeds the stored value for Diameter, then we replace the old Farthest Pair with the new pair $(p, \bar{p})$.

▶ **Corollary 6.** *For a stream S of points in $\mathbb{R}^d$, the Guarded Ball Cover of $\mathcal{O}((1/\varepsilon)^2 \log(1/\varepsilon))$ stored points answers $(\sqrt{2}+\varepsilon)$-FN(x) for any query $x \in \mathbb{R}^d$, and maintains $(\sqrt{2}+\varepsilon)$-FP(S).*

Minimum Enclosing Ball

The following theorem was shown by Chan and Pathak [9] and improved the guarantee of the approximate MEB algorithm of Agarwal and Sharathkumar to $1.22+\varepsilon$ (see [5], p. 91):

▶ **Theorem 7** (Theorem 1 in [9], Theorem 1 in [27])**.** *Let $K_1, ..., K_u$ be subsets of a point set S in $\mathbb{R}^d$, with $B_i =$ MEB(K_i) such that: i) $r(B_i)$ is increasing over i, and ii) $K_i \subset (1+\varepsilon)B_j$, for each $i < j$. Then, $r(B) \leq (1.22+\varepsilon) \cdot r(MEB(S))$, where $B =$ MEB$(\bigcup_{i=1}^{u} B_i)$.*

In our case, K_i is the coreset on the boundary of the MEB computed in line 2 of Algorithm 1. Therefore, the first requirement of Theorem 7 holds by Lemma 3. The second requirement follows immediately for points in Q and by Lemma 4 for points deleted from Q.

▶ **Corollary 8.** *For a stream S of points in $\mathbb{R}^d$, the Guarded Ball Cover of $\mathcal{O}((1/\varepsilon^2)\log(1/\varepsilon))$ stored points maintains $(1.22+\varepsilon)$-MEB(S).*

Coreset for Minimum Enclosing Ball

We mostly follow the analysis of [5]. Let $B = (c, r_m)$ be the most recently created ball added to $\mathcal{B}$ and let Q be the coreset at the time of computing B, thus $B = \mathsf{MEB}(Q)$. Note that $\mathsf{MEB}(S)$ has radius at least r_m, since B is an MEB of a subset of S. We claim that $(\sqrt{2}+\varepsilon)B$ contains all points in S, which implies that Q forms a $(\sqrt{2}+\varepsilon)$-coreset(S). Namely, we show that each point $y \in S$ has $\|yc\| \leq (\sqrt{2}+\varepsilon')r_m$, for $\varepsilon' = 2\varepsilon$.

Consider a point $y \in S$ that is farthest from c. Let $B_i = (c_i, r_i)$ be a guarded ball that has not been deleted and whose $(1+\varepsilon)$-expansion contains y (B_i is well-defined by Theorem 5). By the triangle inequality and the definition of B_i, $\|yc\| \leq \|cc_i\| + \|yc_i\| \leq \|cc_i\| + (1+\varepsilon)r_i$. Let h be the hyperplane passing through c_i with direction cc_i as normal and let h^+ be the halfspace bounded by h that does not contain c. By Lemma 1 there is a guard g in h^+ with $\|gc_i\| = r_i$, and by the cosine law it is $\|gc\| \geq \sqrt{\|cc_i\|^2 + r_i^2}$. Then (using the inequality $a+b \leq \sqrt{2(a^2+b^2)}$),

$$\frac{\|yc\|}{\|gc\|} \leq \frac{\|cc_i\| + (1+\varepsilon)r_i}{\sqrt{\|cc_i\|^2 + r_i^2}} \leq \frac{\sqrt{2(\|cc_i\|^2 + r_i^2)} + \varepsilon \cdot r_i}{\sqrt{\|cc_i\|^2 + r_i^2}} \leq \sqrt{2} + \varepsilon \ .$$

By Lemma 2, there is a guard $q \in Q$ with $\|qg\| \leq \varepsilon^2 r_m/40$, so by the triangle inequality, $\|gc\| \leq \|qc\| + \|gq\| \leq (1+\varepsilon^2/40)r_m$. Hence, $\|yc\| \leq (\sqrt{2}+\varepsilon)(1+\varepsilon^2/40)r_m \leq (\sqrt{2}+2\varepsilon)r_m$.

▶ **Corollary 9.** *For a stream S of points in $\mathbb{R}^d$, the Guarded Ball Cover of $\mathcal{O}((1/\varepsilon)^2\log(1/\varepsilon))$ stored points maintains $(\sqrt{2}+\varepsilon)$-coreset(S).*

5 Lower Bound for Farthest Pair and Farthest Neighbor Queries

We show that computing a $\sqrt{2}$-approximation (with $\varepsilon = 0$) of Farthest Neighbor queries or Farthest Pair is impossible in the streaming model without returning points outside of S.

This applies to the computational model of "coreset-based algorithms" in which the space bound is counted in the number of input points stored and the algorithm must return an input point upon a query (or two input points for the Farthest Pair); crucially, once a point is deleted from memory, it cannot be retrieved. This model is akin to comparison-based model in sorting or selection, as used in [13] for streaming lower bounds for quantile estimation.

▶ **Theorem 10.** *For any $\varepsilon > 0$ and $d = \Omega(1/\varepsilon)$, any coreset-based randomized streaming algorithm answering approximate Farthest Neighbor queries or maintaining the approximate Farthest Pair in $\mathbb{R}^d$ with multiplicative error $\leq \sqrt{2} + \varepsilon$, has to store $\Omega(1/\varepsilon)$ points.*

Proof. We use the easy direction of Yao's minimax principle and design a distribution over instances (points and Farthest Neighbor queries) so that any deterministic streaming algorithm using space $o(1/\varepsilon)$ will, with high constant probability, answer a $\mathsf{FN}(q)$ query incorrectly, i.e., the point returned on the query q will be more than $(\sqrt{2}+\varepsilon)$-factor closer to q than the farthest point. The same argument applies to Farthest Pair.

Suppose without loss of generality that $2/\varepsilon \in \mathbb{Z}$. We insert $k := 1/(2\varepsilon) + 1$ points of the standard basis, i.e., $\mathbf{e}_i = (0, 0, \ldots, 1, 0, \ldots, 0)$, where the i-th coordinate is 1, for $i = 0, \ldots, k-1$ (the order of insertions does not matter). The random part of the construction is to choose $j \in \{0, \ldots, k-1\}$ uniformly at random and make a Farthest Neighbor query for point $q_j = (2\varepsilon, 2\varepsilon, \ldots, -1, 2\varepsilon, \ldots, 2\varepsilon, 0, \ldots, 0)$, where the coordinate j is -1 and only the first k coordinates are not 0. Clearly, the farthest point from q_j is e_j and their Euclidean distance is $\sqrt{4 + (k-1)\cdot 4\varepsilon^2} = \sqrt{4 + 2\varepsilon}$, using the choice of k.

However, with some constant probability, point e_j is not stored as the algorithm stores $o(1/\varepsilon) = o(d)$ points. Conditioning on the event that e_j is not stored, the algorithm needs to answer the query with a point e_i for $i \neq j$. However, the distance between q_j and e_i with $i \neq j$ is $\sqrt{(1-2\varepsilon)^2 + 1 + (k-2)\cdot 4\varepsilon^2} = \sqrt{2 - 4\varepsilon + (k-1)\cdot 4\varepsilon^2} = \sqrt{2 - 2\varepsilon}$, using the choice of k. Thus, the approximation ratio of the algorithm is at least $\sqrt{(4+2\varepsilon)/(2-2\varepsilon)} > \sqrt{2} + \varepsilon$. ◀

6 Conclusions and Open Problems

We have designed streaming algorithms storing $\mathcal{O}((1/\varepsilon^2)\log(1/\varepsilon))$ points from $\mathbb{R}^d$ that can estimate several extent statistics of the input. All error guarantees are almost optimal, with the only exception of the Minimum Enclosing Ball application, where there exists a small gap between the approximation guarantee of $1.22 + \varepsilon$ and the lower bound converging to $(\sqrt{2}+1)/2 \approx 1.207$ for $d \to \infty$. This is achieved by simplifying (and also fixing) the Blurred Ball Cover from [5] into a "Guarded Ball Cover", where we store $\Theta(1)$ points per ball, compared to $\mathrm{poly}(1/\varepsilon)$ points per ball for the Blurred Ball Cover.

We believe that the space bound can be improved, at least by shaving off the $\log(1/\varepsilon)$ factor. One possible direction for improvement is the use of randomization as both our algorithms and those of [5] are deterministic, while the simple lower bound of $\Omega(1/\varepsilon)$ for "coreset-based" algorithms holds even when randomization is used.

While our algorithms are more space-efficient than that of [5], the amortized update times are somewhat worse, as discussed in Section 3. Thus, we ask if it is possible to optimize the $\mathrm{poly}(d \cdot \varepsilon^{-1})$ amortized update time while retaining the near-quadratic space bound. More importantly, it would be interesting to develop a "fast" streaming algorithm for extent problems, that is, with amortized update time $\mathcal{O}(d \cdot \mathrm{poly}(\log \varepsilon^{-1}))$.

Beyond the streaming setting, an important property of data sketches is mergeability [1], which enables to summarize the input in a parallel or distributed way and then merge the resulting sketches into one summary of the whole dataset. It is open how to design a merge operation for Guarded (or Blurred) Ball covers while retaining the space and approximation guarantees.

References

1 Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Trans. Database Syst.*, 38(4):26, 2013. doi:10.1145/2500128.

2 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004. doi:10.1145/1008731.1008736.

3 Pankaj K. Agarwal, Jiří Matoušek, and Subhash Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Comput. Geom.*, 1:189–201, 1991. doi:10.1016/0925-7721(92)90001-9.

4 Pankaj K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1481–1489. SIAM, 2010. doi:10.1137/1.9781611973075.120.

5 Pankaj K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. *Algorithmica*, 72(1):83–98, 2015. doi:10.1007/S00453-013-9846-4.

6 Mihai Badoiu and Kenneth L. Clarkson. Smaller core-sets for balls. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 801–802. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644240.

7 Mihai Badoiu and Kenneth L. Clarkson. Optimal core-sets for balls. *Comput. Geom.*, 40(1):14–22, 2008. doi:10.1016/J.COMGEO.2007.04.002.

8 Mihai Badoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC)*, pages 250–257. ACM, 2002. doi:10.1145/509907.509947.

9 Timothy M. Chan and Vinayak Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. In *Proceedings of the 12th Workshop on Algorithms and Data Structures (WADS)*, pages 195–206. Springer, 2011. doi:10.1007/978-3-642-22300-6_17.

10 Xi Chen, Vincent Cohen-Addad, Rajesh Jayaram, Amit Levi, and Erik Waingarten. Streaming Euclidean MST to a constant factor. In *Proceedings of the 55th ACM Symposium on Theory of Computing (STOC)*, pages 156–169. ACM, 2023. doi:10.1145/3564246.3585168.

11 Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. New streaming algorithms for high dimensional EMD and MST. In *Proceedings of the 54th ACM Symposium on Theory of Computing (STOC)*, pages 222–233. ACM, 2022. doi:10.1145/3519935.3519979.

12 Xiaoyu Chen, Shaofeng H.-C. Jiang, and Robert Krauthgamer. Streaming Euclidean max-cut: Dimension vs data reduction. In *Proceedings of the 55th ACM Symposium on Theory of Computing (STOC)*, pages 170–182. ACM, 2023. doi:10.1145/3564246.3585170.

13 Graham Cormode and Pavel Veselý. A tight lower bound for comparison-based quantile summaries. In *Proceedings of the 39th ACM Symposium on Principles of Database Systems (PODS)*, pages 81–93. ACM, 2020. doi:10.1145/3375395.3387650.

14 Graham Cormode and Ke Yi. *Small Summaries for Big Data*. Cambridge University Press, 2020.

15 Artur Czumaj, Shaofeng H.-C. Jiang, Robert Krauthgamer, Pavel Veselý, and Mingwei Yang. Streaming facility location in high dimension via geometric hashing. In *Proceedings of the 63rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 450–461. IEEE, 2022. doi:10.1109/FOCS54457.2022.00050.

16 Ömer Egecioglu and Bahman Kalantari. Approximating the diameter of a set of points in the Euclidean space. *Inf. Process. Lett.*, 32(4):205–211, 1989. doi:10.1016/0020-0190(89)90045-8.

17 Hossein Esfandiari, Praneeth Kacham, Vahab Mirrokni, David P. Woodruff, and Peilin Zhong. High-dimensional geometric streaming for nearly low rank data. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL: https://openreview.net/forum?id=yQfA0etfB7.

18 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *Int. J. Comput. Geom. Appl.*, 18(1/2):3–28, 2008. doi:10.1142/S0218195908002520.

19 Ashish Goel, Piotr Indyk, and Kasturi R. Varadarajan. Reductions among high dimensional proximity problems. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 769–778. ACM/SIAM, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365776.

20 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

21 Sariel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011.

22 Piotr Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 539–545. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644200.

23 Shaofeng H.-C. Jiang, Robert Krauthgamer, and Shay Sapir. Moderate dimension reduction for k-center clustering. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, volume 293, pages 64:1–64:16, 2024. doi:10.4230/LIPICS.SOCG.2024.64.

24 Piyush Kumar, Joseph S. B. Mitchell, and E. Alper Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *ACM J. Exp. Algorithmics*, 8, 2003. doi:10.1145/996546.996548.

25 Sepideh Mahabadi, Ilya P. Razenshteyn, David P. Woodruff, and Samson Zhou. Non-adaptive adaptive sampling on turnstile streams. In *Proceedings of the 52nd ACM Symposium on Theory of Computing (STOC)*, pages 1251–1264. ACM, 2020. doi:10.1145/3357713.3384331.

26 Yury Makarychev, Naren Sarayu Manoj, and Max Ovsiankin. Near-optimal streaming ellipsoidal rounding for general convex polytopes. In *Proceedings of the 56th ACM Symposium on Theory of Computing (STOC)*, pages 1526–1537. ACM, 2024. doi:10.1145/3618260.3649692.

27 Vinayak Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. Master's thesis, University of Waterloo, 2011.

28 David P. Woodruff and Taisuke Yasuda. High-dimensional geometric streaming in polynomial space. In *Proceedings of the 63rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 732–743. IEEE, 2022. doi:10.1109/FOCS54457.2022.00075.

29 Hamid Zarrabi-Zadeh and Timothy M. Chan. A simple streaming algorithm for minimum enclosing balls. In *Proceedings of the 18th Annual Canadian Conference on Computational Geometry (CCCG)*, 2006.

A A Note on Lemma 2 in [5]

We report on a possible issue in the proof of Lemma 2 in [5] (a similar issue appears in the conference version [4]). Lemma 2 in [5] states that for any i, $r(B_{i+1}) \geq (1+\varepsilon^2/8) \cdot r(B_i)$, where B_i and B_{i+1} are two consecutive balls of the Blurred Ball Cover. The property that the radii of balls increase geometrically is crucially needed to bound the space requirements.

The proof of Lemma 2 goes as follows: Ball $B_{i+1} = \textsf{APPROX-MEB}(\bigcup_{j \leq i} K_j \cup A), \varepsilon/3)$ is created (K_j is the coreset of B_j and APPROX-MEB is the subroutine of [7] that computes it) because one of the points in a buffer A is not contained in any $(1+\varepsilon)$-expansion of a

ball B_j for $j \leq i$. Define ball B' as the MEB of $\bigcup_{j \leq i} K_j \cup A$. It is subsequently claimed that $r(B_{i+1}) \geq r(B')$, without a proof. However, B' is the MEB of $\bigcup_{j \leq i} K_j \cup A$, while B_{i+1} is an *approximate* MEB for $\bigcup_{j \leq i} K_j \cup A$, namely the $(1 + \varepsilon/3)$-expansion of B_{i+1} contains all these points but a smaller expansion of B_{i+1} may not contain them all.

We observe that this is fixable by computing a tighter MEB approximation $B_{i+1} = \text{APPROX-MEB}(\bigcup_{j \leq i} K_j \cup A), \varepsilon^2/16)$. That, however, may result in a coreset K_{i+1} of size $\Theta(\varepsilon^{-2})$, which (unlike our approach) increases the space bound in [5] from $\mathcal{O}(\varepsilon^{-3} \log(1/\varepsilon))$ to $\mathcal{O}(\varepsilon^{-4} \log(1/\varepsilon))$. We are unaware of a fix that does not affect the space bound or error guarantees.

Improved Dominance Filtering for Unions and Minkowski Sums of Pareto Sets

Konstantinos Karathanasis
Department of Computer Engineering and Informatics, University of Patras, Greece
PIKEI New Technologies, Patras, Greece

Spyros Kontogiannis
Department of Computer Engineering and Informatics, University of Patras, Greece
Computer Technology Institute and Press "Diophantus", Patras, Greece

Christos Zaroliagis
Department of Computer Engineering and Informatics, University of Patras, Greece
Computer Technology Institute and Press "Diophantus", Patras, Greece

Abstract

A key task in multi-objective optimization is to compute the *Pareto frontier* (a.k.a. *Pareto subset*) P of a given d-dimensional objective space F; that is, a maximal subset $P \subseteq F$ such that every element in P is *non-dominated* (i.e., it is better in at least one criterion, against any other point) within F. This process, called *dominance-filtering*, often involves handling objective spaces derived from either the *union* or the *Minkowski sum* of two given partial objective spaces which are Pareto sets themselves, and constitutes a major bottleneck in several multi-objective optimization techniques. In this work, we introduce three new data structures, ND$^+$-trees, QND$^+$-trees and TND$^+$-trees, which are designed for efficiently indexing non-dominated objective vectors and performing dominance-checks. We also devise three new algorithms that efficiently filter out dominated objective vectors from the union or the Minkowski sum of two Pareto sets. An extensive experimental evaluation on both synthetically generated and real-world data sets reveals that our new algorithms outperform state-of-art techniques for dominance-filtering of unions and Minkowski sums of Pareto sets, and scale well w.r.t. the number of $d \geq 3$ criteria and the sets' sizes.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Data structures design and analysis

Keywords and phrases Multi-Objective Optimization, Multi-Dimensional Data Structures, Pareto Sets, Algorithm Engineering

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.59

Related Version *Full Version*: `http://arxiv.org/abs/2508.20689`

Funding This work was partially supported by the EU I3 Instrument under GA No 101115116 (project AMBITIOUS) and by the University of Patras under GA No 83770 (programme "MEDICUS").

1 Introduction

In multiobjective combinatorial optimization (MOCO) problems, given an implicit description (e.g., via linear constraints) of a *solution space* X and the corresponding *objective space* F with d-dimensional ($d \geq 2$) objective-value vectors of all elements in X, the goal is to compute the *Pareto frontier*, or *Pareto subset*: a maximal subset of F whose elements are non-dominated (i.e., they are better in at least one criterion, against any other point) within F. Many algorithms for MOCO problems, especially when having to work with instances of substantial sizes, rely heavily on the *dominance-filtering* subtask, aiming to efficiently combine (the Pareto frontiers of the objective spaces for) partial solution spaces and filtering out all the dominated objective-value vectors. In this work we focus on two special cases of dominance-filtering, in which the merged objective space F is created as


© Konstantinos Karathanasis, Spyros Kontogiannis, and Christos Zaroliagis;
licensed under Creative Commons License CC-BY 4.0



either the *union* $A \cup B$, or the *Minkowski sum*[1] $A \oplus B$ of two Pareto sets A, B. These are the two most frequently used variants by solvers of various MOCO problems, e.g., of *decomposition* techniques for multiobjective integer programming [18], of *Pareto local search* for multiobjective set cover [15], and of *dynamic programming* methods for multiobjective shortest paths (MOSP) [16, 20, 22], multi-objective knapsack [6], multi-objective vehicle routing [19], or multi-objective network design [2, 4]. As manifested in [21], dominance-checking constitutes a major computational burden of most state-of-the-art algorithms for MOSP problems during the identification of new solutions. Hence, the development of efficient data structures and algorithms to handle dominance-filtering in unions and Minkowski sums of Pareto sets is of utmost importance in MOCO problems.

Related Work and Motivation. The literature offers a diverse collection of dominance filtering techniques. For $d = 2$ objectives, some highly efficient algorithms have been developed [9, 12]. For the more challenging case of $d \geq 3$ objectives, some general approaches have been explored [11, 13]. In dynamic settings, where solutions are not known in advance and are revealed gradually, the choice of an indexing data structure plays a crucial role in efficiently updating the Pareto frontier. Several indexing data structures for dominance checking have been proposed in the literature, such as *balanced binary search trees* [17], *ND-trees* [10, 14], and a variant of *k-d trees* [1, 3]. To the best of our knowledge, the most efficient algorithms for dominance-filtering of unions and Minkowski sums of Pareto sets for $d \geq 3$ objectives appear in [13]. These methods utilize *space-partitioning ND-trees* [10, 14], or *divide-and-conquer* strategies. Despite their effectiveness, these methods suffer from an inherent inefficiency that occurs when the input data emerge from real-world scenarios that typically contain *plateaus* (large collections of objective vectors with identical values in one or more dimensions, e.g., tolls in road networks), and/or are correlated (e.g., distance and time in road networks). In such cases, ND-trees turn out to be highly unbalanced, which results in significant time bottlenecks for the elementary operations of removing dominated elements from an ND-tree and of re-balancing the tree.

Our Contribution. This work focuses on dominance-filtering techniques for unions and Minkowski sums of Pareto sets for $d \geq 3$ optimization criteria. As our first contribution, we propose three novel data structures for indexing sets of non-dominated elements, which are custom-tailored to overcome the critical bottlenecks of the algorithms in [13]: (1) **ND$^+$-trees**, which inherit some desirable features of *k-d* trees [1] and ND-trees [10, 14]. (2) **QND$^+$-trees**, which dynamically adapt partitioning techniques when constructing the indexing tree from a given Pareto set, selecting the most suitable splitting method for each case. This ensures a *provably* balanced tree structure, leading to faster dominance-checks while also achieving dimensionality reduction, whenever this is possible. (3) **TND$^+$-trees** which are specially designed for scenarios where large *plateaus* occur that cause severe imbalances, which the TND$^+$-trees mitigate while also achieving dimensionality reduction, whenever this is possible.

Our second contribution concerns three new algorithms for dominance-filtering of unions and/or Minkowski sums of two Pareto sets for $d \geq 3$ optimization criteria: (1) `PlainNDred`, which reduces the problem's dimensionality by lexicographically sorting the elements, and eliminates the need for element removals from the data structure. (2) `PreND`, which constructs

[1] The *Minkowski sum* $A \oplus B$ contains all the component-wise additions of elements in A and B. If $A = \{(3,5,4),(5,2,1)\}$ and $B = \{(2,1,3),(6,3,2)\}$, then $A \oplus B = \{(5,6,7),(9,8,6),(7,3,4),(11,5,3)\}$.

an initial tree from a subset of the Pareto set, thereby reducing the need for frequent re-balancing, and avoids element removals. (3) `SymND`, which exploits symmetry to compute non-dominated objective vectors, also avoiding element removals. `PlainNDred` and `PreND` are applicable to both the union and the Minkowski sum of two Pareto sets. They can also be applied to a single objective space, as pure dominance-checks, to extract its Pareto frontier. `SymND`, on the other hand, is applicable only to the union of two Pareto sets. All three algorithms are compatible with each of the aforementioned data structures.

Our final contribution is an extensive experimental evaluation to assess the performance of our algorithms and data structures. We consider all nine combinations of a filtering algorithm among `PlainNDred`, `PreND`, and `SymND` with an indexing data structure from ND$^+$-trees, QND$^+$-trees, and TND$^+$-trees. We compare them with the state-of-the-art algorithms in [13] for $d \geq 3$ criteria. For our experimental evaluation, we used real-world data sets, synthetic data sets similar to those in [13], and new synthetic data sets specifically designed to resemble features of real-world instances. Our experimental results reveal that our algorithms are very efficient and scale well w.r.t. both the number of criteria d and the set sizes across all data sets. Notably, they achieve speedups up to 5.9× on real-world data sets and up to 13.2× on synthetic data sets against the best-performing algorithms from [13].

2 Preliminaries

Let $[n] = \{1, 2, \ldots, n\}$, $\forall n \in \mathbb{Z}^+$. In the following, small letters denote scalars, boldfaced small letters denote vectors, and capital letters denote sets. For any element or point $\mathbf{p} \in \mathbb{R}^d$, let $\mathbf{p}[i]$ denote the value of its i-th coordinate, for each $i \in [d]$. We consider multi-objective minimization problems with $d \geq 2$ objective functions:

$$\begin{array}{ll} \text{minimize} & \mathbf{f}(\mathbf{x}) = (\ \mathbf{f}(\mathbf{x})[1] = f_1(\mathbf{x}),\ \mathbf{f}(\mathbf{x})[2] = f_2(\mathbf{x}),\ \ldots\ ,\ \mathbf{f}(\mathbf{x})[d] = f_d(\mathbf{x})\ ) \\ \text{s.t.} & \mathbf{x} \in X \end{array}$$

X is the *solution space*, i.e., the set of feasible solutions for the instance at hand. $F = \mathbf{f}(X) = \{\mathbf{p} \in \mathbb{R}^d : \exists \mathbf{x} \in X, \mathbf{p} = \mathbf{f}(\mathbf{x})\}$ is the corresponding *objective space*, with all d-dimensional vectors that appear as *objective-value vectors* for at least one feasible solution from X. We refer to these objective vectors simply as *(data) points* and focus on $F \subseteq \mathbb{R}^d$, since all dominance checks are conducted among the points of F. For $\mathbf{p}, \mathbf{p}' \in \mathbb{R}^d$, we say that $\mathbf{p}$ *dominates* $\mathbf{p}'$, denoted as $\mathbf{p} < \mathbf{p}'$, if $\mathbf{p} \neq \mathbf{p}'$ and $\mathbf{p}[i] \leq \mathbf{p}'[i]$, $\forall i \in [d]$. For $F \subseteq \mathbb{R}^d$, its *Pareto frontier* (*subset* or *skyline*) is the maximal subset $P \subseteq F$ of points which are not dominated by any other point in F. If $P = F$, then F itself is also called a *Pareto set*. For $A, B \subset \mathbb{R}^d$, their *Minkowski sum* is $A \oplus B = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}$. Given two Pareto sets $A, B \subset \mathbb{R}^d$, their *Pareto union* is the Pareto frontier of $A \cup B$, and their *Pareto sum* is the Pareto frontier of $A \oplus B$. Given $F \subset \mathbb{R}^d$, the *dominance-filtering* problem aims at filtering out all points in F which are dominated by other points in F, so as to construct its Pareto frontier.

3 Algorithmic Background

A generic approach for dominance-filtering is to process the points of $F = \{\mathbf{p_1}, \ldots, \mathbf{p_n}\}$ sequentially, and keep updating a subset P, which will eventually be the Pareto frontier of F, as follows. For each new point $\mathbf{p_i} \in F$: Compare $\mathbf{p_i}$ (sequentially) with each point $\mathbf{p_j} \in P$ ($j < i$). If $\mathbf{p_j} < \mathbf{p_i}$ then reject $\mathbf{p_i}$ (it is dominated by some point in P) and proceed with the next point of F. Otherwise, if $\mathbf{p_i} < \mathbf{p_j}$, then remove $\mathbf{p_j}$ from P (it is dominated by $\mathbf{p_i}$); if there is no other point in P to compare with, append $\mathbf{p_i}$ to P; otherwise, proceed with

a comparison of $\mathbf{p_i}$ with the next point in P. The efficiency of the data structure used to maintain the current subset P and perform the previously mentioned dominance-checks is critical for the performance of this incremental approach. A well-suited data structure for this task is the ND-tree [10], which we discuss subsequently.

3.1 ND-trees

An ND-tree is a typical rooted c-ary tree T, in which a distinct node $r = root(T)$, of degree at most c, is the *root node*, all nodes of degree 1 (except possibly for the root) are its *leaf nodes*, and the remaining nodes of degree from 2 up to $c+1$ are its *internal nodes*. The ND-trees are *leaf-oriented*, meaning that all data points are stored exclusively in leaf nodes. Each leaf node can store up to m points. The parameters c and m must satisfy the condition $c \leq m+1$ [14]. Each node v stores a lower-bounding vector $\mathbf{lb_v}$ and an upper-bounding vector $\mathbf{ub_v}$ for all the data points stored in leaves of the subtree T_v of T rooted at v. Specifically, for each point $\mathbf{p}$ stored in T_v, it holds that $\forall i \in [d], \mathbf{lb_v}[i] \leq \mathbf{p}[i] \leq \mathbf{ub_v}[i]$. An example of an ND-tree can be found in Figure 1.

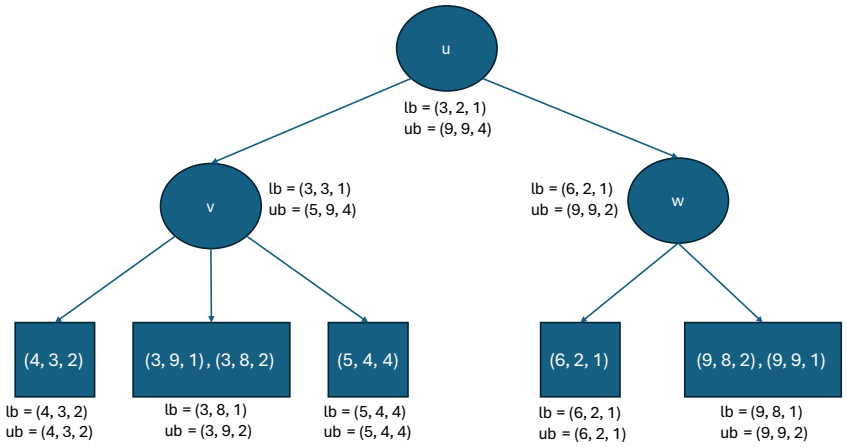


Figure 1 ND-tree containing 3-dimensional points.

The lower and upper-bounding vectors are typically used to determine, as early as possible, if a new data point $\mathbf{p}$ is dominated by any data point already in the tree. For instance, if $\exists i \in [d] : \mathbf{p}[i] < \mathbf{lb_v}[i]$, then $\mathbf{p}$ is not dominated by any data point stored in T_v, and we do not have to explicitly verify this with all of them (of course, it might still be the case that $\mathbf{p}$ dominates some of these data points). If $\mathbf{p} < \mathbf{lb_v}$ then $\mathbf{p}$ dominates all the data points stored in T_v. Finally, if $\mathbf{p} > \mathbf{ub_v}$, then all the data points stored in leaves of T_v dominate $\mathbf{p}$. An ND-tree T supports the following operations.

- `NonDomPrune`$(\mathbf{p}, T)$: This operation effectively utilizes the bounding vectors to perform two tasks, a *dominance-check* for a point $\mathbf{p}$ to decide whether it is dominated by any data point in T, and a *pruning* of T to remove all its data points that are dominated by $\mathbf{p}$. If $\mathbf{p}$ is not dominated by any point in T, `NonDomPrune` returns True; otherwise, it returns False.
- `Insert`$(\mathbf{p}, v)$: This operation inserts a new point $\mathbf{p}$ into a leaf of T_v as follows. If v is a non-leaf node, then a child node w of v is selected with minimum distance from $\mathbf{p}$. The distance of $\mathbf{p}$ from any node u is the Euclidean distance between $\mathbf{p}$ and $\frac{\mathbf{lb_u}+\mathbf{ub_u}}{2}$ (the center of the bounding box containing all data points stored T_u). Consequently, $\mathbf{p}$ is

recursively requested to be inserted in T_w. For a leaf node v, if it stores less than m data points, then $\mathbf{p}$ is simply appended to its list of stored points; otherwise, v is converted into an internal node with c children, and the pending $m+1$ data points are distributed evenly among them. Throughout the insertion process, the bounds of all affected nodes are updated accordingly.

- `SPNDBuild`(P): This operation, introduced in [14], aims to handle situations in which repeated insertions into an ND-tree might eventually lead to an unbalanced tree structure. It takes as input a Pareto set P (e.g., with all the data points stored in an unbalanced ND-tree) and builds from scratch a *perfectly balanced* ND-tree from it, as no pruning is ever required, in which the bounding areas defined by the upper and lower bounds also *are non-overlapping*.

3.2 ND-Tree based Algorithms for Dominance Filtering

The following dominance-filtering algorithms, proposed in [13], are all based on ND-trees and are, to our knowledge, the state-of-the-art techniques for $d \geq 3$ criteria.

- `PlainND`: This algorithm employs `NonDomPrune` and `Insert` to compute either the Pareto union or the Pareto sum of two Pareto sets A and B. It begins with an empty ND-tree T, and processes sequentially the points in F (either $A \cup B$, or $A \oplus B$). For each point $\mathbf{p} \in F$, it calls `NonDomPrune`($\mathbf{p}, T$). If False is returned, $\mathbf{p}$ is discarded. Otherwise, it executes `Insert`($\mathbf{p}, T$) to store $\mathbf{p}$ in T. After having processed all points in F, the points eventually stored in the leaves of T constitute the Pareto frontier of F.
- `PlainSPND`: This algorithm is similar to `PlainND`, but it periodically takes the Pareto set P of data items in the current ND-tree, it then destroys the tree, and consequently calls `SPNDBuild`(P) to create a new, balanced ND-tree. This periodic tree reconstruction can significantly improve the efficiency of intermediate calls to the `NonDomPrune` and `Insert` operations, due to limitations in the imbalance of the evolving ND-tree, while the tree reconstruction cost is amortized among consecutive insertion and pruning operations.
- `PruneSPND`: This algorithm is custom-tailored for computing the Pareto union of two Pareto sets A and B. It exploits the fact that points in A may only be dominated by points in B, and vice versa. Therefore, for the larger of the two sets (say, A) it calls `SPNDBuild`(A) to build a balanced ND-tree T. Subsequently, for each point $\mathbf{p}$ in the smaller set (say, B), it calls `NonDomPrune`($\mathbf{p}, T$) to check if $\mathbf{p}$ is dominated and to remove from T all points dominated by $\mathbf{p}$. If $\mathbf{p}$ is dominated, it is removed from B. After having processed all points in B, T contains all points of A which are not dominated by any point in B, and (eventually) B has only retained those points which are not dominated by any point in A. Their union constitutes the Pareto union.

4 New Data Structures for Dominance-Filtering

We present here our new data structures, ND$^+$-trees, QND$^+$-trees and TND$^+$-trees, designed to boost the efficiency of dominance-filtering. Detailed description of their operations, pseudocode, and the proofs of theorems can be found in the full version of the paper.

4.1 Overview of ND$^+$-trees

An ND$^+$-tree T is a leaf-oriented *binary* tree, with each leaf node storing up to m data points. As in k-d trees [1], every node v is associated with a level-dependent dimension $v.dim = 1 + (v.level \bmod d)$, a subset S_v of data points (to be stored in the leaves of T_v) and

the median value $v.q$ among the coordinates of all the points of S_v in dimension $v.dim$. Each node v also maintains a lower-bounding vector $\mathbf{lb_v}$ (i.e., the coordinate-wise minimum) of all points in S_v, similarly to ND-trees [10]. However, we avoid storing also the upper-bounding vectors, since our experimental evaluation showed that maintaining them is not beneficial for the operations on ND$^+$-trees. Moreover, if v is internal node, then S_v is partitioned in two distinct subsets, one per child of v: the left child gets all points in S_v with $\mathbf{p}[v.dim] < v.q$ and the right child gets all the remaining points of S_v. An example of an ND$^+$-tree is shown in Figure 2. For each ND$^+$-tree T, the following elementary operations are supported:

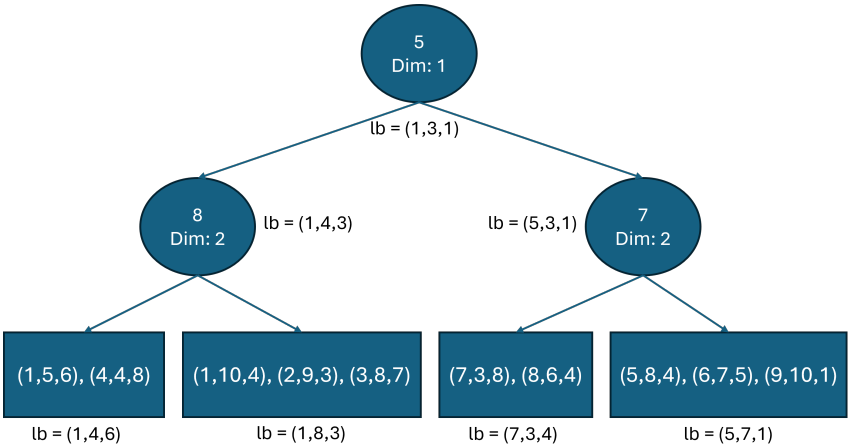


Figure 2 Example of an ND$^+$-tree containing 3-dimensional points with $m = 3$.

- `BuildND`$^+(P, \ell, d)$ constructs an ND$^+$-(sub)tree at level ℓ (an entire tree, when $\ell = 0$) containing all the data points of a Pareto set P.
- `ComputeBoundsND`$^+(r)$ computes the lower-bounding vectors for all nodes of T_r.
- `WidenBoundsND`$^+(v, \mathbf{p})$ updates the lower-bounding vector of node v, if necessary, due to a (previous) addition of a new point $\mathbf{p}$ in T_v.
- `InsertND`$^+(v, \ell, \mathbf{p})$ inserts a new point $\mathbf{p}$ into the subtree T_v rooted at the level-ℓ node v.
- `DominatedND`$^+(v, \ell, \mathbf{p})$ decides whether a new point $\mathbf{p}$ is dominated by any other point in the subtree T_v rooted at the level-ℓ node v.

In the remaining part of this section we provide some theoretical guarantees on the complexities of these elementary operations, when each leaf of the tree stores at most m d-dimensional points, for arbitrary constants $m, d \in O(1)$.

▶ **Theorem 1.** *Given a Pareto set of n points, `BuildND`$^+$ constructs an ND$^+$-tree, with $N = O(n)$ nodes in $O(n \log n)$ time when all splits of a point set produce constant fractions for both parts, and in $O(n^2)$ time otherwise.*

▶ **Theorem 2.** *Given an ND$^+$-tree with N nodes, n points and height h, the following bounds hold for its elementary operations: (i) `ComputeBoundsND`$^+$ takes $O(nmd) = O(n)$ time; (ii) `InsertND`$^+$ takes $O(hd + m) = O(h)$ time; (iii) `DominatedND`$^+$ takes $O(dn) = O(n)$ time.*

4.2 Overview of QND$^+$-trees

For a Pareto set P with its points having *distinct values* per dimension, `BuildND`$^+$ constructs a *balanced* ND$^+$-tree in *quasilinear* time (cf. Theorem 1). However, objective spaces F emerging from real-world scenarios (and their Pareto subsets) rarely adhere to such a strong property. Instead, it is common for large subsets of objective vectors to possess identical

values in certain dimensions, e.g., for tolls in road networks. When a subset of S_v constitutes a large *plateau* around $v.q$ (i.e., those data points have the same value $v.q$ in $v.dim$) for some internal node v, the resulting partition of S_v may be (possibly heavily) uneven. If this pattern occurs frequently at intermediate nodes, then the resulting ND$^+$-tree will be heavily unbalanced, leading to *quadratic* construction time and also *linear* (instead of logarithmic) time for insertion of new points. This may happen even if initially a balanced tree is constructed, due to subsequent insertions of points that constitute a plateau in F.

To tackle these worst-case performances of the ND$^+$-trees, we introduce in this section an alternative data structure, the QND$^+$-trees (*Quartile ND$^+$-trees*). In a nutshell, the QND$^+$-trees are almost identical to the ND$^+$-trees, the only difference being that they perform a more careful bipartition of the data set S_v associated with an internal node v. Specifically, if there is a large plateau (more than one fourth) in dimension $v.dim$ around the splitting value $v.q$, then this plateau is entirely assigned to v's right child, with the remaining points being assigned (irrespectively of their values in dimension $v.dim$) to its left child. Moreover, when checking for dominance-checks in the subtree rooted at v's right child, it is no longer necessary to consider $v.dim$, achieving dimensionality reduction. If no such plateau is discovered in S_v, then the split is done as in an ND$^+$-tree.

We consider the following partitioning strategies: `Median Partitioning (MP)`, also used in ND$^+$-trees, assigns points of S_v with values in $v.dim$ less than $v.q$ to v's left child, and the remaining points of S_v to v's right child; `Quartile Partitioning (QP)` assigns points of S_v with value $v.q$ in dimension $v.dim$ to v's right child, and all other points to v's left child. The following elementary operations are supported for QND$^+$-trees:

- `BuildQND`$^+(P, \ell, d)$ constructs a QND$^+$-tree with the points of Pareto set P, in an analogous manner with `BuildND`$^+(P, \ell, d)$. The only difference is that, before splitting the point set S_v associated to an internal node v, it first computes the quartiles of S_v in dimension $v.dim$ and then applies `QP` when $Q_1 = Q_2$ (this implies that the size of the plateau is at least $|S_v|/4$), and `MP` otherwise. After completing the tree construction, it calls `ComputeBoundsND`$^+$ to compute all the lower bounding vectors.
- `InsertQND`$^+(v, \ell, \mathbf{p})$ inserts a new point $\mathbf{p}$ in T_v, similarly to the corresponding operation on ND$^+$-trees. It first updates $\mathbf{lb_v}$ using `WidenBoundsND`$^+$ and then recursively calls itself for the appropriate child of v (if this is an internal node), depending on the existence (or not) of a plateau in $v.dim$, or else (for a leaf node) it either stores $\mathbf{p}$ in v's data list, otherwise (when there is no free space in v's list) it changes v into an internal node and redistributes evenly all the pending data points among its two children, executing also `ComputeBoundsND`$^+$ to update their lower bounding vectors.
- `DominatedQND`$^+(v, \ell, \mathbf{p}, D)$, an adaptation of `DominatedND`$^+$ on QND$^+$-trees, checks whether a new point $\mathbf{p}$ is dominated by any other point in T_v, taking also into account if there are any dimensions to ignore (due to existence of plateaus) during its recursive calls.

▶ **Theorem 3.** *Given a Pareto set of n points, `BuildQND`$^+$ constructs a QND$^+$-tree with $N = O(n)$ nodes and height $O(\log n + d) = O(\log n)$ in $O(n(\log n + d)) = O(n \log n)$ time.*

▶ **Theorem 4.** *Given a QND$^+$-tree with N nodes and height h, containing n points, then the following time bounds hold for its elementary operations: (i) `InsertQND`$^+$ takes $O(hd) = O(h)$ time; (ii) `DominatedQND`$^+$ takes $O(dn) = O(n)$ time.*

4.3 Overview of TND$^+$-trees

TND$^+$-trees (*Ternary ND$^+$-trees*) are designed to exploit both tree balance and dimensionality reduction due to the existence of plateaus. Contrary to the other trees, TND$^+$-trees are not strictly binary. In particular, whenever a large plateau is discovered within S_v in dimension $v.dim$ of an internal node v, three children are created: The left child is associated with points $\mathbf{p} \in S_v$ with $\mathbf{p}[v.dim] < v.q$, the right child with points $\mathbf{p} \in S_v$ with $\mathbf{p}[v.dim] > v.q$, and the middle child with all the points which constitute the plateau (i.e., $\mathbf{p}[v.dim] = v.q$). Again, for the middle child we also exploit the resulting dimensionality reduction. We will refer to this plateau-based partitioning strategy as a `TriPartitioning (TP)`. If no plateau is detected, then the standard `Median Partitioning (MP)` strategy is applied. The following elementary operations are supported for TND$^+$-trees:

- `BuildTND`$^+(P, \ell, d)$ constructs a TND$^+$-tree for the points of a Pareto set in a fashion analogous to that of `BuildQND`$^+$, applying `(TP)` whenever $Q_1 = Q_2$ or $Q_2 = Q_3$ for the quartiles of S_v in dimension $v.dim$, ensuring that at least 25% of S_v are assigned to the middle child where we also benefit from dimensionality reduction. After completing the tree construction, `ComputeBoundsTND`$^+$ is executed, a slightly modified version of `ComputeBoundsND`$^+$ that also takes into consideration the middle child, to compute the lower-bounding vectors of each node in the tree.
- `InsertTND`$^+(v, \ell, \mathbf{p})$ inserts $\mathbf{p}$ to the TND$^+$-(sub)tree T_v, resembling `InsertQND`$^+$. It first updates v's lower-bounding vector using `WidenBoundsND`$^+$. If v is an internal node, then a recursive call of the method is executed to insert $\mathbf{p}$ to the appropriate child of v, taking also into account whether v possesses a middle child. If v is a leaf node that stores less than m points, then it simply stores $\mathbf{p}$ at v's list of points, otherwise v is converted into an internal node and the $m + 1$ now pending points (including $\mathbf{p}$) are redistributed among its (either two or three, depending on the presence of a plateau) children, using `BuildTND`$^+$. Upon completion, `ComputeBoundsTND`$^+$ is executed to compute the lower bounds of the newly created children.
- `DominatedTND`$^+(v, \ell, \mathbf{p}, D)$ determines whether $\mathbf{p}$ is dominated by any other point in the TND$^+$-(sub)tree T_v.

▶ **Theorem 5.** *Given a Pareto set of n points, `BuildTND`$^+$ constructs an TND$^+$-tree with $N = O(n)$ nodes and height $O(\log n + d) = O(\log n)$ in $O(n(\log n + d)) = O(n \log n)$ time.*

▶ **Theorem 6.** *Given a TND$^+$-tree with n nodes and height h, then the following time bounds hold for its elementary operations: (i) `InsertTND`$^+$ takes $O(hd) = O(h)$ time; (ii) `DominatedTND`$^+$ takes $O(dn) = O(n)$ time.*

4.4 Comparison of the New Data Structures

To illustrate the differences between the three data structures, and to demonstrate how QND$^+$- and TND$^+$-trees produce more balanced structures than ND$^+$-trees in the presence of plateaus, consider building ND$^+$-, QND$^+$-, and TND$^+$-trees from the point set $S = \{(1, 10, 2),\ (2, 9, 6),\ (2, 8, 7),\ (2, 12, 0),\ (2, 7, 8),\ (2, 11, 1),\ (4, 7, 4),\ (5, 7, 3),\ (6, 7, 2),\ (7, 6, 1),\ (8, 6, 0)\}$. Assume that each leaf node can store up to $m = 4$ points.

- **ND$^+$-tree:** Median Partitioning (MP) is first applied in the first dimension, yielding the median value 2. This assigns $(1, 10, 2)$ to the left subtree and all remaining points to the right, giving $L = \{(1, 10, 2)\}$ and $R = \{(7, 6, 1),\ (8, 6, 0),\ (2, 7, 8),\ (4, 7, 4),\ (5, 7, 3),\ (6, 7, 2),\ (2, 8, 7),\ (2, 9, 6),\ (2, 11, 1),\ (2, 12, 0)\}$. MP is then applied to R in the second dimension, with median 7, producing $RL = \{(7, 6, 1),\ (8, 6, 0)\}$ and $RR = \{(2, 12, 0),$

$(2,11,1)$, $(6,7,2)$, $(5,7,3)$, $(4,7,4)$, $(2,9,6)$, $(2,8,7)$, $(2,7,8)\}$. Since RR exceeds the leaf size m, it is split once more using MP in the third dimension, where the median is 4, giving $RRL = \{(2,12,0)$, $(2,11,1)$, $(6,7,2)$, $(5,7,3)\}$ and $RRR = \{(4,7,4)$, $(2,9,6)$, $(2,8,7)$, $(2,7,8)\}$. All resulting subtrees contain at most m points, completing the tree (see Figure 3).

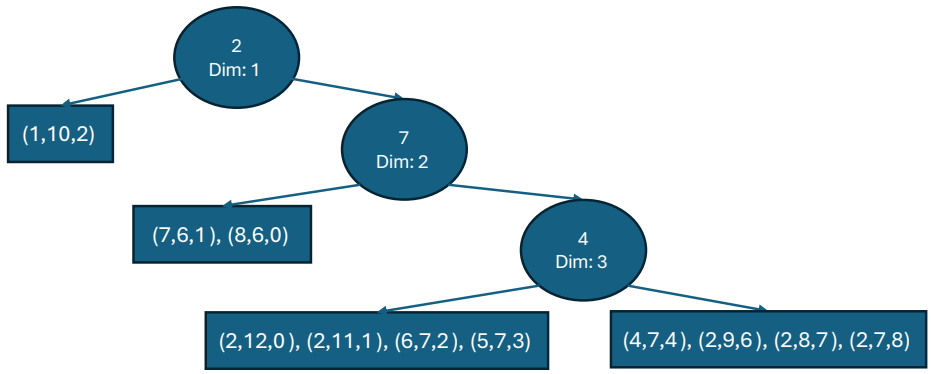


Figure 3 ND$^+$-tree containing 3-dimensional points with $m = 4$.

- **QND$^+$-tree:** Since $Q_1 = Q_2 = 2$ in the first dimension, Quartile Partitioning (QP) is used, yielding $L = \{(7,6,1)$, $(8,6,0)$, $(4,7,4)$, $(5,7,3)$, $(6,7,2)$, $(1,10,2)\}$ and $R = \{(2,7,8)$, $(2,8,7)$, $(2,9,6)$, $(2,11,1)$, $(2,12,0)\}$. In L, $Q_1 = 6 \neq 7 = Q_2$ in the second dimension, so MP is applied, splitting it into $LL = \{(7,6,1)$, $(8,6,0)\}$ and $LR = \{(4,7,4)$, $(5,7,3)$, $(6,7,2)$, $(1,10,2)\}$, both of which satisfy the leaf size constraint. In R, $Q_1 = 8 \neq 9 = Q_2$ in the second dimension, so MP is again applied, giving $RL = \{(2,8,7)$, $(2,7,8)\}$ and $RR = \{(2,9,6)$, $(2,11,1)$, $(2,12,0)\}$, each also within the allowed limit. The tree is thus complete (see Figure 4).

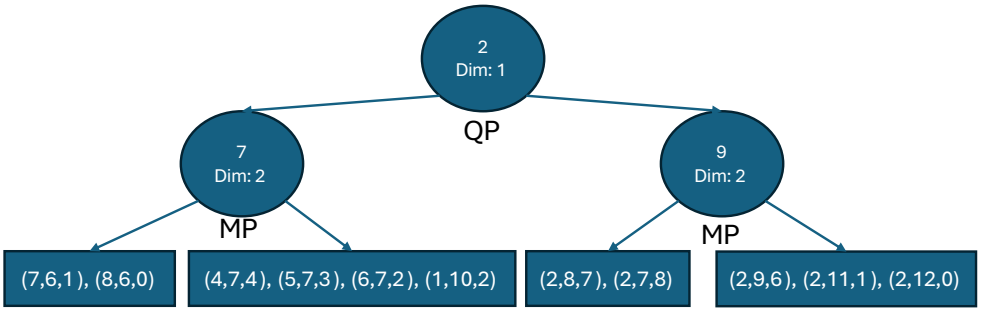


Figure 4 QND$^+$-tree containing 3-dimensional points with $m = 4$.

- **TND$^+$-tree:** $Q_1 = Q_2 = 2$ in the first dimension, so TriPartitioning (TP) is applied, producing $L = \{(1,10,2)\}$, $M = \{(2,7,8)$, $(2,8,7)$, $(2,9,6)$, $(2,11,1)$, $(2,12,0)\}$, and $R = \{(7,6,1)$, $(8,6,0)$, $(4,7,4)$, $(5,7,3)$, $(6,7,2)\}$. L needs no further processing. In M, no plateau is present in the second dimension, so MP is used with median 9, resulting in $ML = \{(2,7,8)$, $(2,8,7)\}$ and $MR = \{(2,9,6)$, $(2,11,1)$, $(2,12,0)\}$. In R, a plateau is found: $Q_2 = 7 = Q_3$, so TP is applied again, yielding $RL = \{(7,6,1)$, $(8,6,0)\}$ and $RM = \{(4,7,4)$, $(5,7,3)$, $(6,7,2)\}$. All resulting subtrees respect the leaf size constraint, completing the tree (see Figure 5).

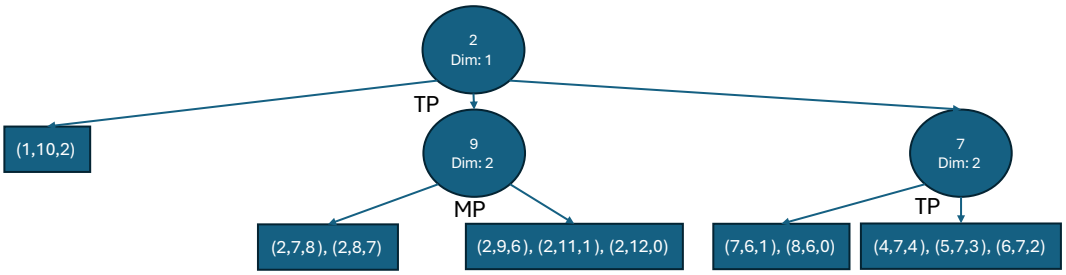


Figure 5 TND$^+$-tree containing 3-dimensional points with $m = 4$.

5 Overview of New Algorithms for Pareto Unions and Sums

We now present our new algorithms for computing Pareto unions or Pareto sums of Pareto sets. These algorithms are designed to work with any of the three data structures of Section 4. For simplicity, all algorithms are presented w.r.t. ND$^+$-trees – details and missing proofs can be found in the full version. In our experimental evaluation (cf. Section 6) we consider all possible combinations of algorithms and data structures.

5.1 PlainNDred

As noted in [13], the main computational burden in `PlainND`, `PlainSPND` and `PruneSPND` is the execution of `NonDomPrune` operations (cf. Section 3). The most demanding task is the removal of all dominated points from the tree, as new points are inserted to it. The main idea behind the `reduced PlainND` algorithm (`PlainNDred` in short) is to avoid this costly *pruning* task of the evolving tree, by ensuring that any point that is inserted to the tree is actually a member of the required Pareto frontier of F. To achieve this, `PlainNDred` first lexicographically sorts F, in quasilinear time. Then, to efficiently manage *dominance-checks*, it processes the data points in that order and uses one of the new indexing structures (ND$^+$-trees in `PlainNDred`, QND$^+$-trees in `PlainQNDred`, and TND$^+$-trees in `PlainTNDred`) to store only the non-dominated ones of F so far. In particular, for each point $\mathbf{p}$ in the lexicographic order, the algorithm must only check if it is dominated by any point in the tree, since $\mathbf{p}$ cannot dominate any of the preceding points in that order, as shown next.

▶ **Lemma 7.** *Let $S = (\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n})$ be a lexicographic order of a set $F \subset \mathbb{R}^d$ of n points. Then the following non-dominance property holds: $\forall 1 \leq i < j \leq n$, $\mathbf{p_j}$ cannot dominate $\mathbf{p_i}$.*

If $\mathbf{p}$ is not dominated by any point already in the tree, the algorithm inserts it. Moreover, *dominance-checks* can safely ignore dimension 1, since $\mathbf{p}[1] \geq \mathbf{q}[1]$ for any point preceding $\mathbf{p}$ in the lexicographic order. Therefore, `PlainNDred` needs only to check the remaining $d-1$ dimensions. As a result, the algorithm builds a tree considering only the last $d-1$ dimensions of the points in F. We denote as $\mathbf{p_{red}}$ the projection of $\mathbf{p}$ on the last $d-1$ dimensions. The following statement demonstrates time complexity of `PlainNDred`.

▶ **Theorem 8.** *For an n-point set F that is either the Minkowski sum or the union of two Pareto sets, the time complexity of `PlainNDred` algorithm is $O(n^2(d-1)) = O(n^2)$.*

Proof. The points of F are first lexicographically sorted, in time $O(n \log n)$. Then, for each point, we perform a dominance-check against the previously processed points that belong to the tree. Each pairwise dominance-check takes time $O(d-1)$ since only the last $d-1$ dimensions matter. Even if all points in F are non-dominated and no pruning occurs, each point is compared to all previously processed points. Therefore, the algorithm makes at most $(d-1)\frac{n(n-1)}{2} \in O((d-1)n^2)$ comparisons, for all dominance-checks. ◀

5.2 PreND

The consecutive insertions into the tree by `PlainNDred` are likely to gradually unbalance it, thereby diminishing the efficiency of subsequent dominance-checks and insertions. To address this challenge, one option would be to periodically rebuild the tree, as is done by `PlainSPND` in [13]. However, building the entire tree from scratch is not a trivial task. To avoid that, we could leverage once more the lexicographic order of the point set $F = (\mathbf{p_1}, \ldots, \mathbf{p_n})$ to compute first an initial subset P of the Pareto frontier, which is then used to construct a *balanced* ND$^+$-tree that will be large enough so that the subsequent insertions of the remaining non-dominated points, again examined in lexicographic order, will not be able to unbalance it severely. This is exactly the main idea of the `presorted ND` algorithm (`PreND` in short).

To compute this subset of the Pareto frontier, we deploy the `ParetoSubset` algorithm. It starts with the initialization of a vector $\mathbf{y}$ of length d, corresponding to the number of dimensions, with each dimension assigned the value ∞, and then makes a single pass over the data points, in lexicographic order, using $\mathbf{y}$ to keep track of the smallest values seen in each dimension, up to the current point $\mathbf{p_i}$. For the next point in order, $\mathbf{p_{i+1}}$, if there exists a dimension $j \in [d]$ such that $\mathbf{p_{i+1}}[j] < \mathbf{y}[j]$, then $\mathbf{p_{i+1}}$ is not dominated by any preceding point. Moreover, due to the lexicographic order, $\mathbf{p_{i+1}}$ cannot be dominated by any subsequent point $\mathbf{p_k} : k \geq i+2$ (cf. Lemma 7). Therefore, $\mathbf{p_{i+1}}$ is certainly a non-dominated point in F and is appended to P, and $\mathbf{y}$ is updated to always keep the smallest value seen so far, per dimension. Otherwise, when $\mathbf{y} \leq \mathbf{p_{i+1}}$, $\mathbf{p_{i+1}}$ is appended to another subset Q, for further examination, during the second processing phase. Note that, since all points are processed in lexicographic order, Q remains lexicographically sorted. Observe also that, for $d = 2$, `ParetoSubset` already computes the entire Pareto frontier of F.

The `PreND` algorithm initially calls `ParetoSubset` to get the sets P and Q. It then calls `BuildND`$^+$ to construct an initial ND$^+$-tree with the points of P. Subsequently, for each point $\mathbf{q} \in Q$, it calls `DominatedND`$^+$ to determine if $\mathbf{q}$ is dominated by any point of the tree. If it is dominated, then it is discarded. Otherwise, it is appended to P and inserted to the ND$^+$-tree by calling `InsertND`$^+$. After having processed all points in Q, P is the required Pareto frontier of F. Note that `PreND` can be used for constructing the Pareto union or the Pareto sum of two Pareto sets, but also for identifying the Pareto frontier of a single set.

▶ **Theorem 9.** *For an n-point set F that is either the Minkowski sum or the union of two Pareto sets, the time complexity of `PreND` is $O(n^2)$.*

Proof. `ParetoSubset` first lexicographically sorts the set of points, in $O(n \log n)$ time. Then, it iterates through all n points and for each point determines in $O(d)$ time whether it belongs to P or Q. Thus, `ParetoSubset` runs in $O(n \log n + nd) = O(n \log n)$ time. Next, an ND$^+$-tree is constructed from the points in P. In the worst case, this step takes $O(n^2)$ time ($O(n \log n)$ for QND$^+$ and TND$^+$-trees). After constructing the tree, for each point in Q, we perform a dominance-check against the points already in the tree. In the worst case, where no pruning occurs and every point is non-dominated, each dominance-check involves comparing the point with all previously processed points. Since the pairwise dominance-checks are executed in time $O(d)$, and we perform this check for at most $\frac{n(n-1)}{2}$ pairs of points, the total time complexity for the dominance-checks is $O(dn^2)$. Hence, the overall time complexity of the `PreND` algorithm is $O(dn^2) = O(n^2)$. ◀

5.3 SymND

Especially for the union of two Pareto sets A, B, recall that points of one set may be dominated only by points of the other set. Thus, applying some sort of symmetric dominance-filtering could be extremely efficient. This is exactly what the `symmetric ND` algorithm (`SymND` in short) does. It constructs first an ND$^+$-tree using the points of A and then executes `DominatedND`$^+$ for each point in B, to remove from B all those points which are dominated by a point in A. Then, it constructs another ND$^+$-tree, using only the remaining points in B, and executes `DominatedND`$^+$ for each point in A, to remove those points which are dominated by a point in B. In the end, both surviving subsets of A and B contain only non-dominated points, and their union constitutes the Pareto union of A and B.

▶ **Theorem 10.** *Given two Pareto sets A and B, the time complexity of `SymND` to compute their Pareto union is $O(|A| \cdot |B|)$.*

Proof. Let $|A| = n_1 \leq |B| = n_2$. Assume also that the size of their Pareto union (to be computed) is k. First, an ND$^+$-tree is constructed using the points of the smaller set A, which takes time $O(dn_1^2)$ in the worst case for ND$^+$-trees, and $O(n_1 \log n_1)$ for QND$^+$ and TND$^+$-trees. Then, for each point in set B, the `DominatedND`$^+$ method is applied. Since there are n_2 points in set B, and each `DominatedND`$^+$ operation takes time $O(dn_1)$ in the worst case (when each point of B is compared to all points in the tree), the total complexity of this step is $O(dn_1n_2)$.

Consequently, we construct a second ND$^+$-tree using the remaining (at most $\hat{n}_2 = \min\{k, n_2\}$) points from set B. This tree construction requires in worst case time $O(d\hat{n}_2^2)$ time for ND$^+$-trees and $O(\hat{n}_2 \log \hat{n}_2)$ for QND$^+$ and TND$^+$-trees. Next, the `DominatedND`$^+$ method is applied for each point in set A, removing any dominated points from A. As there are n_1 points in set A, the total complexity of this second phase is $O(dn_1\hat{n}_2)$. Therefore, the overall time complexity of the `SymND` algorithm is $O(dn_1^2 + dn_1n_2 + d\hat{n}_2^2 + dn_1\hat{n}_2) = O(dn_1(n_2 + \min\{k, n_2\}))$ with ND$^+$-trees and $O(dn_1 \log n_1 + dn_1n_2 + d\hat{n}_2 \log \hat{n}_2 + dn_1\hat{n}_2) = O(dn_1n_2)$ with QND$^+$-trees and TND$^+$-trees. ◀

6 Experimental Evaluation

In our experimental evaluation, we implemented all nine combinations of our proposed indexing data structures and dominance-filtering algorithms. We distinguish each combination with an appropriate naming as follows: For each algorithm, its short name is used to indicate an implementation with ND$^+$-trees, and variants with QND$^+$-trees and TND$^+$-trees are indicated by the appearance in the short name of the substrings "QND" and "TND", respectively. For example, `PlainNDred` indicates the implementation of `reduced PlainND` with ND$^+$-trees, `PreQND` indicates the implementation of `presorted ND` with QND$^+$-trees, and `SymTND` indicates the implementation of `symmetric ND` with TND$^+$-trees.

In addition, we implemented nine algorithms of [13], which constitute, to the best of our knowledge, the state-of-the-art dominance-filtering algorithms for MOCO problems with $d \geq 3$ dimensions. Apart from the algorithms `PlainSPND` and `PruneSPND` which were discussed in Section 3, several more algorithms were provided in [13]: `NonDomDC` explores divide-and-conquer strategies that partition the initial set of solutions into smaller subsets to reduce unnecessary comparisons; `FilterX2` and `FilterSym` are bidirectional filters that are also based on divide-and-conquer techniques; `BatchedSPND` that utilizes SPND-trees; `LimMem` provides a memory-efficient alternative for scenarios where memory availability is limited; finally, `Doubling(Filter)` and `Doubling(Tree)` are adaptations of `FilterSym` and

`PruneSPND`, respectively, which are custom-tailored for Minkowski sums. All implemented algorithms are listed in Table 1, where their applicability on the specific dominance-filtering variant (union and/or Minkowski sum) is also mentioned.

Table 1 Algorithms that were implemented and tested in our experimental evaluation.

Reference: [13]		Reference: this work	
Algorithm	**Usage**	**Algorithm**	**Usage**
FilterX2	Union	SymND	Union
FilterSym	Union	SymQND	Union
BatchedSPND	Minkowski sum	SymTND	Union
Doubling(Filter)	Minkowski sum	PreND	Union & Minkowski sum
Doubling(Tree)	Minkowski sum	PreQND	Union & Minkowski sum
LimMem	Minkowski sum	PreTND	Union & Minkowski sum
NonDomDC	Union & Minkowski sum	PlainNDred	Union & Minkowski sum
PlainSPND	Union & Minkowski sum	PlainQNDred	Union & Minkowski sum
PruneSPND	Union	PlainTNDred	Union & Minkowski sum

6.1 Data Sets

To evaluate the performance of our algorithms, we used both real-world and synthetic data sets. Note that real-world data sets with three or more objectives ($d \geq 3$) are very rare. Since our main goal is to test scalability with dimensionality, we have also used two families of synthetic data sets with up to $d = 10$ objectives: the randomly constructed data sets of [13], and some new, carefully generated synthetic data sets that resemble some crucial features of real-world instances for MOCO problems. Below, we provide a brief overview of all these data sets; more details can be found in the full version of the paper. The **RW** sets are based on the New York City road network [5], and are equipped with two cost metrics: travel time and distance. We extend them to higher dimensions according to well established augmentation techniques: for $d = 3$, we adopted [17] and introduced a third objective which is related to hazardous material transportation [7]. For $d = 5$, we adopted [8] and added a fourth objective which is a random integer from 1 to 100, and a fifth objective which is a random integer from 1 up to the number of graph edges. The **URS** sets were synthetically generated according to the procedure described in [13] (sampling points uniformly in d-dimensional space and then projecting them into the unit sphere to ensure that these are Pareto sets). Apart from these "baseline" RW and URS data sets, we also considered a few extensions towards incorporating some typical features of real-world instances. The **RWP** and **URSP** sets try to model realistic instances with repeated or flat objective values (e.g., tolls for road networks), by introducing plateaus in some objectives of the RW and URS sets, respectively. The **RWC** and **URSC** sets simulate interdependencies between objectives (also encountered quite often in real-world instances), by introducing correlations between some objectives. Finally, the **URSPC** set combines both correlation and plateau features in a single Pareto set. The size of data sets given as input to all algorithms varies from 10K to 1M points, while the number of objectives (dimensions) d varies from 3 to 10 (3, 5 for RW, 5 for RWP and RWC, 4, 6, 8, 10 for URS, URSP, URSC and 5, 6, 8, 10 for URSPC).

6.2 Overview of Experimental Results

We provide here an overview of our experimental results. All details can be found in the full version of the paper.

Comparison with the state-of-the-art. Tables 2 and 3 compare our algorithms with the best-performing algorithms from [13] for the Pareto sum and the Pareto union of two Pareto sets, respectively. We compute the speedup factor per (algorithm, data set) pair, as follows: for each (dimensionality, input size) pair, we identify the fastest algorithm of [13]. We then compute the ratio of its runtime to that of our own algorithm. In the tables we report the minimum and maximum speedups observed across all (dimensionality, input size) pairs.

Table 2 Min and Max speedups for each algorithm for the *Pareto Sum* operation.

Algorithm	RW		RWP		RWC		URS		URSP		URSC		URSPC	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
PreND	1.1	3.8	2.9	4.4	1.4	2.6	1.8	5.2	1.8	4.9	1.7	4.7	2.9	7.7
PreQND	1.0	3.7	2.8	4.3	1.4	2.5	1.7	5.6	1.8	5.3	1.7	5.2	2.9	8.2
PreTND	1.0	3.7	2.8	4.2	1.4	2.5	1.7	5.5	1.8	5.2	1.7	5.0	2.9	7.8
PlainNDred	1.1	3.7	2.6	4.5	1.4	2.5	2.9	10.8	2.4	7.0	1.7	6.2	2.9	12.2
PlainQNDred	1.1	3.6	2.6	4.3	1.4	2.6	2.9	11.8	2.4	9.2	1.7	7.5	3.0	13.2
PlainTNDred	1.1	3.6	2.5	4.4	1.4	2.6	3.0	8.6	2.4	8.8	1.7	7.2	2.9	12.7

Table 3 Min and Max speedups for each algorithm for the *Pareto Union* operation.

Algorithm	RW		RWP		RWC		URS		URSP		URSC		URSPC	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
PreND	1.9	5.5	2.0	4.2	1.7	4.1	1.5	5.2	1.7	3.6	0.4	2.6	0.5	1.5
PreQND	2.1	5.9	2.0	4.9	1.7	4.5	1.6	5.8	1.8	4.0	0.5	2.9	0.5	1.7
PreTND	2.0	5.7	1.7	4.6	1.6	4.3	1.6	5.7	1.8	3.9	0.5	2.8	0.5	1.6
SymND	1.4	3.0	1.4	3.0	1.1	2.9	1.7	4.8	1.3	3.7	1.4	3.6	1.0	3.5
SymQND	1.5	3.2	1.4	3.4	1.1	3.1	1.7	5.4	1.4	4.4	1.6	3.9	1.0	3.7
SymTND	1.4	3.1	1.3	3.1	1.1	2.9	1.6	5.2	1.3	3.9	1.4	3.5	1.0	3.4
PlainNDred	2.1	5.3	1.7	2.6	1.7	3.1	1.7	6.3	1.8	3.8	0.5	2.2	0.5	1.6
PlainQNDred	2.2	5.8	2.3	3.0	2.0	3.6	1.9	6.8	1.9	4.3	0.6	2.4	0.5	1.6
PlainTNDred	2.2	5.6	2.0	2.7	2.0	3.3	1.8	6.7	1.8	4.1	0.6	2.3	0.5	1.7

For Pareto sums (Table 2), `PreND`, `PlainNDred` and their variants exhibit nearly identical performance, consistently outperforming all other algorithms on RW/RWC/RWP, with speedups reaching up to 4.5×. For the synthetic data sets, all our algorithms significantly outperform the algorithms of [13], with `PlainNDred` and its variants achieving the largest speedups, up to 11.8× for URS, 9.2× for URSP, 7.5× for URSC, and 13.2× for URSPC.

For Pareto unions (Table 3), the variants of `PreND` achieve the greatest speedup on RW/RWP/RWC. On URS/URSP, all our algorithms show similar speedup ranges. On URSC/URSPC, the variants of `SymND` emerge as the top performers. Note that, although the variants of `PreND` and `PlainNDred` seem to occasionally be slower than `PruneSPND` on URSC and URSPC, they are faster or identical in average in most cases. `SymND` and it variants outperform `PruneSPND` for all data sets, with speedups of up to 3.9×.

Regarding the three data structures, for the Pareto union of two Pareto sets, QND$^+$-trees and TND$^+$-trees outperform ND$^+$-trees across almost all data sets. However, for the Pareto sum, ND$^+$-trees generally perform better than QND$^+$-trees and TND$^+$-trees on the real-world data sets. In contrast, for the synthetic data sets, QND$^+$-trees are typically the most efficient, followed by TND$^+$-trees, with ND$^+$-trees trailing behind.

Exploring the impact of tree height on algorithmic performances. The balance of a tree is crucial for the performance of our algorithms. When the tree is well-balanced, pruning mechanisms at each level can reduce more efficiently the search space and limit the number

of candidate points that may dominate a new one. To evaluate this in practice, we conducted an experiment using a URSP set A consisting of $n = 100,000$ points, with a plateau of size $n/2$ around the median in a random dimension. We constructed an ND$^+$, a QND$^+$ and a TND$^+$ tree using their `Build` methods on A. For each tree, we computed the average height, the balance indicator $BI = \max(\text{height}) - \min(\text{height})$, and the number of dominance-checks required per point from a second set B when queried against the respective trees.

Table 4 Comparison of the novel data structures, w.r.t. average tree height, balance indicator (BI), and number of dominance-checks, for $100K$ points.

Tree	4 Dimensions			6 Dimensions			8 Dimensions			10 Dimensions		
	Height	BI	Checks	Height	BI	Checks	Height	BI	Checks	Height	BI	Checks
ND$^+$	16	10	56	15	7	243	14	4	511	14	2	537
QND$^+$	13	0	68	13	0	230	13	0	431	13	0	423
TND$^+$	13	1	38	13	1	137	13	1	371	13	1	428

The results, summarized in Table 4, show that QND$^+$ and TND$^+$ trees are consistently more balanced than ND$^+$ trees and also exhibit shorter heights. Although ND$^+$ trees are only slightly taller on average, their significantly higher balance indicator values reveal a more skewed structure. Consequently, except in the case of 4 dimensions, the ND$^+$ trees required more dominance-checks than the other two variants. It is important to note that, as highlighted in our complexity analysis (in the full version of the paper), the worst-case scenario may require a point to be compared against all nodes in the tree. However, the experimental results indicate that in practice, the number of dominance-checks is substantially lower. This highlights the practical efficiency of our tree structures and the effectiveness of their lower-bounding mechanisms.

7 Conclusions and Future Work

We introduced three new data structures and three efficient algorithms for computing the Pareto unions and Pareto sums of Pareto sets, for which we provided a theoretical analysis for their worst-case performances and conducted a thorough experimental evaluation against state-of-art techniques (for $d \geq 3$) from [13], on several real-world and synthetically generated data sets. In all instances and dominance-filtering scenarios all of our algorithms consistently outperformed each algorithm in [13] (except for the case of Pareto unions on URSC and URSPC data sets, in which only `SymND` and its variants outperformed the best algorithm in [13]). Future work will focus on enhancing the performance of `PreND` by developing an alternative method to `ParetoSubset`, so as to precompute larger subsets of the Pareto frontier without significantly increasing computational costs.

References

1 Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. doi:10.1145/361002.361007.

2 Anthony Chen, Juyoung Kim, Seungjae Lee, and Youngchan Kim. Stochastic multi-objective models for network design problem. *Expert Syst. Appl.*, 37(2):1608–1619, 2010. doi:10.1016/J.ESWA.2009.06.048.

3 Wei-Mei Chen, Hsien-Kuei Hwang, and Tsung-Hsi Tsai. Maxima-finding algorithms for multidimensional samples: A two-phase approach. *Comput. Geom.*, 45(1-2):33–53, 2012. doi:10.1016/J.COMGEO.2011.08.001.

4 Mina Dehghani, Vahab Vahdat, Maghsoud Amiri, Elaheh Rabiei, and Seyedmohammad Salehi. A multi-objective optimization model for a reliable generalized flow network design. *Comput. Ind. Eng.*, 138, 2019. doi:10.1016/J.CIE.2019.106074.

5 Camil Demetrescu, Andrew V Goldberg, and David S Johnson. *The shortest path problem: Ninth DIMACS implementation challenge*, volume 74. American Mathematical Soc., 2009.

6 Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectr.*, 22(4):425–460, 2000. doi:10.1007/S002910000046.

7 Erhan Erkut, Stevanus A. Tjandra, and Vedat Verter. Chapter 9 hazardous materials transportation. In Cynthia Barnhart and Gilbert Laporte, editors, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 539–621. Elsevier, 2007. doi:10.1016/S0927-0507(06)14009-8.

8 Carlos Hernández, William Yeoh, Jorge A. Baier, Ariel Felner, Oren Salzman, Han Zhang, Shao-Hung Chan, and Sven Koenig. Multi-objective search via lazy and efficient dominance checks. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 7223–7230. International Joint Conferences on Artificial Intelligence Organization, August 2023. Main Track. doi:10.24963/ijcai.2023/850.

9 Demian Hespe, Peter Sanders, Sabine Storandt, and Carina Truschel. Pareto sums of pareto sets. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 60:1–60:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.60.

10 Andrzej Jaszkiewicz and Thibaut Lust. Nd-tree-based update: A fast algorithm for the dynamic nondominance problem. *IEEE Trans. Evol. Comput.*, 22(5):778–791, 2018. doi:10.1109/TEVC.2018.2799684.

11 Antoine Kerberenes, Daniel Vanderpooten, and Jean Michel Vanpeperstraete. Computing efficiently the nondominated subset of a set sum. *Int. Trans. Oper. Res.*, 30(6):3455–3478, 2023. doi:10.1111/ITOR.13191.

12 David G. Kirkpatrick and Raimund Seidel. Output-size sensitive algorithms for finding maximal vectors. In Joseph O'Rourke, editor, *Proceedings of the First Annual Symposium on Computational Geometry, Baltimore, Maryland, USA, June 5-7, 1985*, pages 89–96. ACM, 1985. doi:10.1145/323233.323246.

13 Kathrin Klamroth, Bruno Lang, and Michael Stiglmayr. Efficient dominance filtering for unions and minkowski sums of non-dominated sets. *Comput. Oper. Res.*, 163:106506, 2024. doi:10.1016/J.COR.2023.106506.

14 Bruno Lang. Space-partitioned nd-trees for the dynamic nondominance problem. *IEEE Trans. Evol. Comput.*, 26(5):1004–1014, 2022. doi:10.1109/TEVC.2022.3145631.

15 Thibaut Lust and Daniel Tuyttens. Variable and large neighborhood search to solve the multiobjective set covering problem. *Journal of Heuristics*, 20:165–188, 2014. doi:10.1007/s10732-013-9236-8.

16 Francisco Javier Pulido, Lawrence Mandow, and José-Luis Pérez-de-la-Cruz. Dimensionality reduction in multiobjective shortest path search. *Comput. Oper. Res.*, 64:60–70, 2015. doi:10.1016/J.COR.2015.05.007.

17 Zhongqiang Ren, Richard Zhan, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Enhanced multi-objective A* using balanced binary search trees. In Lukás Chrpa and Alessandro Saetti, editors, *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, pages 162–170. AAAI Press, 2022. doi:10.1609/SOCS.V15I1.21764.

18 Britta Schulze, Kathrin Klamroth, and Michael Stiglmayr. Multi-objective unconstrained combinatorial optimization: a polynomial bound on the number of extreme supported solutions. *Journal of Global Optimization*, 74:495–522, 2019. doi:10.1007/s10898-019-00745-6.

19 Fei Tan, Zheng-Yi Chai, and Ya-Lun Li. Multi-objective evolutionary algorithm for vehicle routing problem with time window under uncertainty. *Evol. Intell.*, 16(2):493–508, 2023. doi:10.1007/S12065-021-00672-0.

20 George Tsaggouris and Christos D. Zaroliagis. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. *Theory Comput. Syst.*, 45(1):162–186, 2009. doi:10.1007/S00224-007-9096-4.

21 Carlos Hernández Ulloa, Han Zhang, Sven Koenig, Ariel Felner, and Oren Salzman. Efficient set dominance checks in multi-objective shortest-path algorithms via vectorized operations. In Ariel Felner and Jiaoyang Li, editors, *Seventeenth International Symposium on Combinatorial Search, SOCS 2024, Kananaskis, Alberta, Canada, June 6-8, 2024*, pages 208–212. AAAI Press, 2024. doi:10.1609/SOCS.V17I1.31560.

22 Arthur Warburton. Approximation of pareto optima in multiple-objective, shortest-path problems. *Oper. Res.*, 35(1):70–79, 1987. doi:10.1287/OPRE.35.1.70.

Fast and Memory-Efficient BWT Construction of Repetitive Texts Using Lyndon Grammars

Jannik Olbrich
Ulm University, Germany

Abstract

The Burrows-Wheeler Transform (BWT) serves as the basis for many important sequence indexes. On very large datasets (e.g. genomic databases), classical BWT construction algorithms are often infeasible because they usually need to have the entire dataset in main memory. Fortunately, such large datasets are often highly repetitive. It can thus be beneficial to compute the BWT from a compressed representation. We propose an algorithm for computing the BWT via the Lyndon straight-line program, a grammar based on the standard factorization of Lyndon words. Our algorithm can also be used to compute the extended BWT (eBWT) of a multiset of sequences. We empirically evaluate our implementation and find that we can compute the BWT and eBWT of very large datasets faster and/or with less memory than competing methods.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Shared memory algorithms; Theory of computation → Data compression; Theory of computation → Sorting and searching; Mathematics of computing → Combinatorics on words

Keywords and phrases Burrows-Wheeler Transform, Grammar compression

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.60

Supplementary Material *Software (Source Code)*: `https://gitlab.com/qwerzuiop/lyndongrammar` [38], archived at `swh:1:dir:0274cae4abb893b49bc035ef3a57986a418afe40`

1 Introduction

The *Burrows-Wheeler Transform* (BWT) [13] is one of the most important data structures in sequence analysis and is applied in a wide range of fields from data compression to bioinformatics. It is obtained by assigning the ith symbol of the BWT to the last character of the ith lexicographically smallest conjugate of the input text. The BWT can be computed in linear time and space. The run-length compressed BWT (RLBWT) uses the fact that the number of equal-character runs in the BWT is small compared to the text length when the text is repetitive [13]. Mantaci et al. [32] extended the notion of the BWT to string collections: the *extended* BWT (eBWT) of a string collection consists of the last characters of the strings in $\mathcal{M}$ arranged in infinite periodic order (see Section 2 for definitions).

The BWT serves as the basis of several *self-indexes*, i.e., data structures supporting access to the text as well as pattern matching queries. One of the most successful such indexes is the FM-index [20], which is used e.g. in the important bioinformatics programs BWA [30] and Bowtie [28]. However, the FM-index uses space proportional to the size of the dataset and is thus not applicable for huge datasets that vastly exceed the size of main memory. Such datasets often contain data from thousands or millions of individuals from the same species and are therefore extremely repetitive. The *r-index* [22] is based on the RLBWT and is the first text index of size proportional to the number r of runs in the BWT. It can be constructed in space proportional to r from the RLBWT, which raises the problem of constructing the RLBWT of huge datasets with small working memory. For this reason, various tools have emerged that exploit the repetitiveness of these datasets.

The Lyndon grammar of a text is a straight-line program (SLP), i.e., a context-free grammar in Chomsky normal form that generates a single text, where each symbol corresponds to a node in the Lyndon tree. It was introduced in [25] as the basis for a self-index. In this


© Jannik Olbrich;
licensed under Creative Commons License CC-BY 4.0

paper, we describe a method for efficiently computing several BWT variants of a text or a text collection via the input's Lyndon grammar and show that our method surpasses other tools with respect to time or memory used, especially when using multiple threads.

1.1 Related work

Boucher et al. [11] developed *prefix-free parsing (PFP)*, a dictionary-compression technique that results in a dictionary and parse from which the (RL)BWT can be constructed in space proportional to their total size and in time linear to the size of the dataset. While the dictionary typically grows slowly as the size of the dataset increases, the parse grows linearly with the input size (with a small constant chosen at runtime). Consequently, for extremely large and repetitive datasets, "the parse is more burdensome than the size of the dictionary" [41]. In an effort to remedy this, Oliva et al. [41] developed *recursive prefix-free parsing*, which applies PFP to the parse. PFP has also been used to construct the eBWT [10]. The currently fastest implementation for computing the suffix array SA (and BWT) for general texts is `libsais` and is based on the linear-time *suffix array induced sorting (SAIS)* algorithm [37, 40]. `ropebwt3` uses `libsais` to compute the BWT of chunks of the input and successively merges these BWTs [29]. Díaz-Domínguez and Navarro used grammar compression to maintain the intermediate data in the SAIS algorithm with low main memory usage, resulting in a linear-time semi-external algorithm that computes a variant of the eBWT [14, 17]. Specifically for a collection of sequences where each is similar to a given reference (e.g. human chromosomes), Masillo introduced `CMS-BWT`, which computes the BWT via *matching statistics* [35].

Lyndon words have previously been used in the construction of SA and play an important role in the original eBWT. In [33, 34], a strategy for computing SA was presented where "local suffixes" (i.e., substrings ending at boundaries between Lyndon factors) can be processed separately in each Lyndon factor. Later, Baier effectively generalized the underlying idea by showing that sorting the suffixes by their longest Lyndon prefixes can be used for linear time SA construction [2]. The *bijective* BWT (BBWT) is the eBWT of the text's Lyndon factors [23, 27, 32]. Bonomo et al. showed that it is possible to compute the eBWT via the BBWT and gave an $\mathcal{O}(N \log N / \log\log N)$ construction algorithm based on properties of Lyndon words [8, 9]. Later, Bannai et al. modified the SAIS algorithm to compute the BBWT in linear time [3]. Since publication, Baier's algorithm was improved in terms of time and memory usage [6, 39] and further generalized such that it can be used to compute eBWT and BBWT in linear time [40]. The principles used this generalized algorithm lie at the base of our algorithm for computing the BWT, BBWT and eBWT variants from a Lyndon grammar.

1.2 Our contributions

Let N be the size of a text or text collection, $n_{\max}$ the maximum size of a string in the text collection, and g the size of the Lyndon SLP of the text or text collection. We give two online in-memory algorithms for constructing the Lyndon SLP with $\mathcal{O}(Ng)$ and $\mathcal{O}\big(N \log g + g \log^2 g\big)$ worst-case time complexity, both using $\mathcal{O}(g)$ words of memory. Notably, the text or text collection is streamed from right to left and does not have to reside in main memory. Additionally, we give an expected linear-time algorithm that uses $\mathcal{O}(g + n_{\max})$ words of memory. Furthermore, we give an $\mathcal{O}(g)$ algorithm for sorting the symbols of a Lyndon SLP lexicographically by their generated strings, and an $\mathcal{O}(N)$ algorithm for constructing the (run-length compressed) BWT, BBWT, or eBWT from a Lyndon SLP. We implemented our algorithms and demonstrate empirically that we can construct the BWT, BBWT or eBWT of repetitive texts or text collections faster and/or with less space than competing methods.

2 Preliminaries

For $i, j \in \mathbb{N}_0$ we denote the set $\{k \in \mathbb{N}_0 : i \leq k \leq j\}$ by the interval notations $[i\,..\,j] = [i\,..\,j+1) = (i-1\,..\,j] = (i-1\,..\,j+1)$. For an array A we analogously denote the *subarray* from i to j by $\mathsf{A}[i\,..\,j] = \mathsf{A}[i\,..\,j+1) = \mathsf{A}(i-1\,..\,j] = \mathsf{A}(i-1\,..\,j+1) = \mathsf{A}[i]\mathsf{A}[i+1]\ldots\mathsf{A}[j]$. We use zero-based indexing, i.e., the first entry of the array A is $\mathsf{A}[0]$.

A *string* S of *length* n over an *alphabet* Σ is a sequence of n characters from Σ. We denote the length n of S by $|S|$ and the ith symbol of S by $S[i-1]$, i.e., strings are zero-indexed. In this paper we assume any string S of length n to be over a totally ordered and linearly sortable alphabet (i.e., the characters in S can be sorted in $\mathcal{O}(n)$). Analogous to arrays we denote the *substring* from i to j by $S[i\,..\,j] = S[i\,..\,j+1) = S(i-1\,..\,j] = S(i-1\,..\,j+1) = S[i]S[i+1]\ldots S[j]$. For $j > i$ we let $S[i\,..\,j]$ be the *empty string* ε. For two strings u and v and an integer $k \geq 0$ we let uv be the concatenation of u and v and denote the k-times concatenation of u by u^k. A string S is *primitive* if it is non-periodic, i.e., $S = w^k$ implies $w = S$ and $k = 1$. The *suffix* i of a string S of length n is the substring $S[i\,..\,n)$ and is denoted by $\mathrm{suf}_i(S)$. Similarly, the substring $S[0\,..\,i]$ is a *prefix* of S. A suffix (prefix) is *proper* if $i > 0$ ($i+1 < n$). For two possibly empty strings u and v, uv is a conjugate of vu.

We assume totally ordered alphabets. This induces a total order on strings. Specifically, we say a string S of length n is *lexicographically smaller* than another string T of length m if and only if there is some $\ell \leq \min\{n, m\}$ such that $S[0\,..\,\ell) = T[0\,..\,\ell)$ and either $n = \ell < m$ or $\ell < \min\{n, m\}$ and $S[\ell] < T[\ell]$, and write $S <_{lex} T$ in this case. A non-empty string S is in its *canonical form* if and only if it is lexicographically minimal among its conjugates. If S is additionally strictly lexicographically smaller than all of its other conjugates, S is a *Lyndon word*. Equivalently, S is a Lyndon word if and only if S is lexicographically smaller than all its proper suffixes [18].

In the following, we use `abbabcbcabb` as our running example.

▶ **Theorem 1** (Chen-Fox-Lyndon theorem [15]). *Any non-empty string S has a unique* Lyndon factorization*, that is, there is a unique sequence of Lyndon words (*Lyndon factors*) $v_1 \geq_{lex} \ldots \geq_{lex} v_k$ with $S = v_1 \ldots v_k$.*

The Lyndon factorization of our running example is $\texttt{abbabcbc}, \texttt{abb}$.

▶ **Definition 2** (Standard Factorization [15]). *The* standard factorization *of a Lyndon word w of length $|w| \geq 2$ is the tuple (u, v) where $w = uv$ and v is the longest proper suffix of w that is Lyndon. The standard factorization (u, v) of a Lyndon word w with $|w| \geq 2$ always exists and both u and v are Lyndon.*

The standard factorization of the first Lyndon factor of our running example is $(\texttt{abb}, \texttt{abcbc})$ as `abcbc` is Lyndon and no longer proper suffix is Lyndon.

▶ **Lemma 3** ([40, Lemma 3.20]). *Any Lyndon word w with $|w| \geq 2$ is of the form $w[0]w_{c_1} \cdots w_{c_k}$, where $w_{c_1} \geq_{lex} w_{c_2} \geq_{lex} \ldots \geq_{lex} w_{c_k}$ are the Lyndon factors of $\mathrm{suf}_1(w)$.*

Figure 1 illustrates the following definitions.

▶ **Definition 4** (next smaller suffix array, nss). *Let the* next smaller suffix *array* nss *of a string S be such that $\mathrm{nss}[i] = \min\{j \in (i\,..\,|S|] \mid \mathrm{suf}_j(S) <_{lex} \mathrm{suf}_i(S)\}$.*

▶ **Definition 5** (Infinite Periodic Order). *We write $S <_\omega T$ if and only if the infinite concatenation $S^\infty = SS\ldots$ is lexicographically smaller than the infinite concatenation $T^\infty = TT\ldots$.*

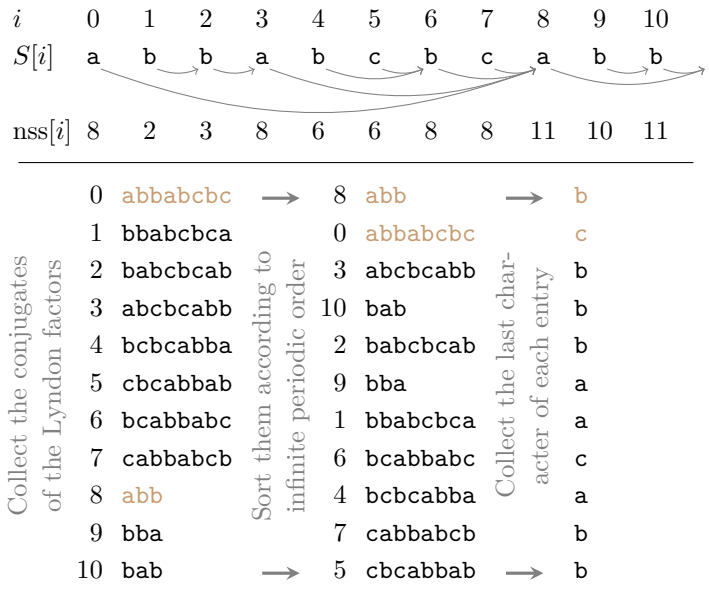


Figure 1 Next smaller suffix array nss (top) and BBWT (bottom) of the running example $S =$ `abbabcbcabb`. Each arrow points from i to nss[i]. Lyndon factors are coloured (■).

For instance, `ab` $<_{lex}$ `aba` $<_{lex}$ `abb` and `abb` $>_{\omega}$ `ab` $>_{\omega}$ `aba` since `abb`$\ldots >_{lex}$ `abab`$\ldots >_{lex}$ `abaab`$\ldots$.

▶ **Definition 6** (Bijective Burrows-Wheeler Transform (BBWT)). *The* bijective Burrows-Wheeler Transform (BBWT) *of a string S is the string obtained by taking the last character of each conjugate of the Lyndon factors of S arranged in infinite periodic order.*

▶ **Definition 7** (Extended Burrows-Wheeler Transform (eBWT)). *The* extended Burrows-Wheeler Transform (eBWT) *of a multiset $\mathcal{M}$ of strings is the string obtained by taking the last character of each conjugate of the strings in $\mathcal{M}$ arranged in infinite periodic order.*

Note that, by definition, the BBWT of a string S is the eBWT of the Lyndon factors of S. Similarly, the eBWT of a multiset $\mathcal{M}$ of strings is the BBWT of the concatenation of the canonical forms of the strings in $\mathcal{M}$ arranged in lexicographically decreasing order [3] (the Lyndon factors of the resulting string are conjugates of the roots of the strings in $\mathcal{M}$).[1]

For a primitive string S, its BWT, the eBWT of $\{S\}$, and the BBWT of the smallest conjugate of S are identical. Consequently, the $-BWT of a string S (i.e., BWT(S\$), where \$ is smaller than all characters in S) commonly computed via the suffix array of S, is identical to BWT(\$$S$) = BBWT(\$$S$) = eBWT($\{$\$$S\}$).

2.1 Lyndon Grammar

▶ **Definition 8** (Lyndon Tree/Forest [4, 25]). *The* Lyndon tree *of a Lyndon word w – denoted by LTree(w) – is the ordered full binary tree defined recursively as follows: If $|w| = 1$, then LTree(w) consists of a single node labelled by w, and if $|w| \geq 2$ and w has the standard factorization (u, v), the root of LTree(w) is labelled by w, the left child of the root is LTree(u), and the right child of the root is LTree(v). For a non-Lyndon word, we let the* Lyndon Forest *be the sequence of Lyndon trees of the Lyndon factors.*

[1] The eBWT was originally defined for sets of primitive strings [32]. This limitation is unnecessary [10, 40].

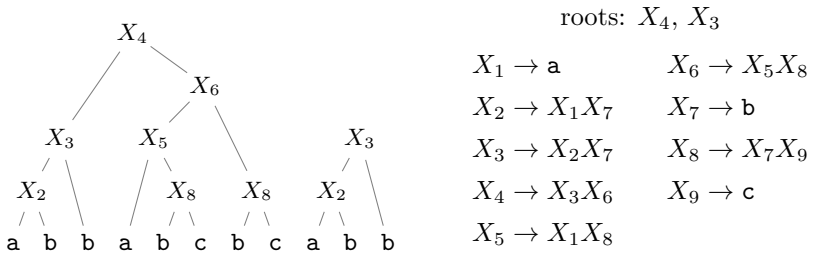


Figure 2 Lyndon Forest (left) and Lyndon SLP (right) for the running example $S = \texttt{abbabcbcabb}$. For clarity, instead of the node labels the corresponding grammar symbols are shown. Note the structural similarity between the Lyndon forest and the arrows indicating nss in Figure 1.

The Lyndon Forest of the running example is shown in Figure 2. Note that the Lyndon Forest of a string S is closely related to the Lyndon array λ of S [25], where $\lambda[i]$ is the length of the longest prefix of $\mathrm{suf}_i(S)$ that is Lyndon (equivalently, $\lambda[i] = \mathrm{nss}[i] - i$ [7], cf. Figure 1). A succinct representation of λ occupies $2|S|$ bits and can be computed in linear time directly from the text [7, 16, 31].

▶ **Definition 9** (Lyndon straight-line program [25]). *A* Lyndon straight-line program *(SLP) is a context-free grammar $\mathcal{G}$ over an alphabet Σ in Chomsky normal form, where $[X_i]$ is a Lyndon word for each symbol X_i and each rule $X_i \to X_aX_b$ is such that the standard factorization of $[X_i]$ is $([X_a], [X_b])$, where $[X_i]$ denotes the unique string generated by X_i. A symbol X_i is called* terminal symbol *if there is a rule of the form $X_i \to c$ ($c \in \Sigma$), and* non-terminal symbol *otherwise. When $r_1, \ldots, r_k$ are the root symbols of $\mathcal{G}$,[2] $\mathcal{G}$ generates $[\mathcal{G}] = [r_1] \ldots [r_k]$, and $[r_1] \geq \cdots \geq [r_k]$ is the Lyndon factorization of $[\mathcal{G}]$. The* size $|\mathcal{G}|$ *of $\mathcal{G}$ is the number of production rules plus the number of start symbols. The* derivation tree *of the SLP $\mathcal{G}$ is a labelled ordered tree where the root node has the children $r_1, \ldots, r_k$. We assume that distinct symbols generate distinct words.*

A Lyndon SLP arises from renaming the nodes of a Lyndon Forest such that two nodes with isomorphic subtrees (i.e., representing the same string) are assigned the same symbol. The Lyndon Forest and Lyndon SLP of our running example can be seen in Figure 2.

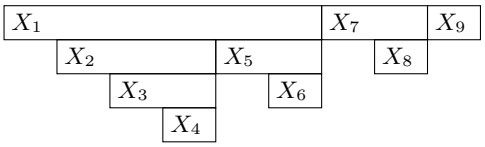
Clearly, the size of a Lyndon SLP $\mathcal{G}$ is bounded by $\mathcal{O}(|[\mathcal{G}]|)$. This bound is tight and there is no non-trivial bound in terms of e.g. the number of runs in the BWT, the size γ of the smallest string attractor [26] or the size of the smallest SLP: consider $S = \texttt{a}^{n-1}\texttt{b}$ for $n > 1$. The smallest SLP generating S has size $\mathcal{O}(\log n)$, its BWT has two runs, and the smallest string attractor has size $\gamma = |\{0, n-1\}| = 2$, but every suffix of S is Lyndon and thus the Lyndon SLP generating S has $\Theta(n)$ symbols.

In the following, we assume the set of symbols V of an SLP $\mathcal{G}$ to be numbered consecutively, i.e., $V = \{X_1, \ldots, X_{|V|}\}$.

3 Properties of sorted Lyndon Grammars

Consider a Lyndon grammar $\mathcal{G}$ where the symbols are sorted lexicographically, i.e., $[X_i] <_{lex} [X_j] \iff i < j$. In this section we examine such Lyndon SLPs and find properties that enable sorting any Lyndon SLP lexicographically in linear time (Section 4) and give rise to an online algorithm for constructing Lyndon SLPs (Section 6.3). Note that the grammar shown in Figure 2 is lexicographically sorted.

[2] In slight deviation of the usual definition of SLPs, we allow multiple root/start symbols. The reason for this is that we want to represent arbitrary strings with the Lyndon SLP, not just Lyndon words.



$X_1 \to \mathtt{a}$ $X_7 \to \mathtt{b}$ $X_9 \to \mathtt{c}$
$X_2 \to X_1X_7$ $X_5 \to X_1X_8$ $X_8 \to X_7X_9$
$X_3 \to X_2X_7$ $X_6 \to X_5X_8$
$X_4 \to X_3X_6$

Figure 3 First-symbol forest of the (lex. sorted) Lyndon SLP of the running example (cf. Figure 2).

▶ **Definition 10** (Prefix- and Suffix-symbols). *Define $\mathcal{P}_L(X_i)$ ($\mathcal{P}_R(X_i)$) as the set of symbols occurring on the leftmost (rightmost) path of X_i's derivation tree. More formally, if X_i is a terminal symbol define $\mathcal{P}_L(X_i) = \mathcal{P}_R(X_i) = \{X_i\}$, and if $X_i \to X_aX_b$ is a non-terminal symbol define $\mathcal{P}_L(X_i) = \{X_i\} \cup \mathcal{P}_L(X_a)$ and $\mathcal{P}_R(X_i) = \{X_i\} \cup \mathcal{P}_R(X_b)$. We correspondingly define $\mathcal{P}_L^{-1}(X_i)$ ($\mathcal{P}_R^{-1}(X_i)$) to be the set of symbols on whose leftmost (rightmost) path of the derivation tree X_i occurs, i.e. $\mathcal{P}_L^{-1}(X_i) = \{X_j \mid X_i \in \mathcal{P}_L(X_j)\}$ and $\mathcal{P}_R^{-1}(X_i) = \{X_j \mid X_i \in \mathcal{P}_R(X_j)\}$.*

For example, we have $\mathcal{P}_L(X_6) = \{X_6, X_5, X_1\}$, $\mathcal{P}_R(X_6) = \{X_6, X_8, X_9\}$ and $\mathcal{P}_L^{-1}(X_1) = \{X_1, X_2, X_3, X_4, X_5, X_6\}$ for the grammar shown in Figure 2.

▶ **Definition 11.** *Define $\mathcal{D}(X_i)$ as the set of symbols occurring in X_i's derivation tree. More formally, if X_i is a terminal symbol define $\mathcal{D}(X_i) = \{X_i\}$, and if $X_i \to X_aX_b$ is a non-terminal symbol define $\mathcal{D}(X_i) = \{X_i\} \cup \mathcal{D}(X_a) \cup \mathcal{D}(X_b)$.*

Let $r(X_i)$ be the largest symbol whose derivation tree has X_i on the leftmost path, i.e., $r(X_i) = \max\big(\mathcal{P}_L^{-1}(X_i)\big)$. Similarly, let $l(X_i)$ be the smallest symbol whose derivation tree has X_i on the leftmost path, i.e. $l(X_i) = \min\big(\mathcal{P}_L^{-1}(X_i)\big)$. For example, $l(X_1) = X_1$ and $r(X_1) = X_6$ in the grammar from Figure 2. In the remainder of this section, we show that all symbols X_j in the interval $[l(X_i)\,..\,r(X_i)]$ generate strings with $[X_i]$ as prefix and even satisfy $X_j \in \mathcal{P}_L(X_i)$. This then implies that these intervals form a tree-structure.

▶ **Lemma 12.** *$l(X_i) = X_i$ for all i.*

Proof. For $X_j \in \mathcal{P}_L^{-1}(X_i)$ we have $[X_i] \leq_{lex} [X_j]$ as $[X_i]$ is prefix of $[X_j]$ by definition. ◀

The following lemma follows from basic properties of Lyndon words and the definition of the Lyndon grammar/tree. Its proof is omitted due to space constraints.

▶ **Lemma 13.** *For all $X_j \in [l(X_i)\,..\,r(X_i)]$ it holds $X_i \in \mathcal{P}_L(X_j)$, i.e., X_i occurs on the leftmost path of X_j's derivation tree. Equivalently, $[l(X_i)\,..\,r(X_i)] = \mathcal{P}_L^{-1}(X_i)$.*

▶ **Corollary 14.** *Any two intervals $[l(X_i), r(X_i)]$ and $[l(X_j), r(X_j)]$ either do not intersect or one is fully contained within the other.*

Proof. Let X_i and X_j be different such that $l(X_i) < l(X_j) \leq r(X_i)$ (Lemma 3 implies $l(X_i) \neq l(X_j)$). This implies $X_j \in \mathcal{P}_L(X_i)$ by Lemma 13 and thus $\mathcal{P}_L(X_j) \subset \mathcal{P}_L(X_i)$ and $r(X_j) \leq r(X_i)$. ◀

▶ **Corollary 15.** *The intervals $[l(X_i), r(X_i)]$ induce a forest.*

The following definition defines this forest. An example can be seen in Figure 3.

▶ **Definition 16.** *We say an interval* $[l(X_j), r(X_j)]$ *is* embedded *in an interval* $[l(X_i), r(X_i)]$ *if it is a subinterval of* $[l(X_i), r(X_i)]$*, i.e.* $l(X_i) < l(X_j) \leq r(X_j) \leq r(X_i)$*. If* $[l(X_j), r(X_j)]$ *is embedded in* $[l(X_i), r(X_i)]$ *and there is no interval embedded in* $[l(X_i), r(X_i)]$ *in which* $[l(X_j), r(X_j)]$ *is embedded,* $[l(X_j), r(X_j)]$ *is a* child interval *of* $[l(X_i), r(X_i)]$*.*

The first-symbol forest *is a rooted forest, where each symbol* X_i *of the SLP corresponds to a node* u_i*, and* u_j *is a child of* u_i *if and only if* $[l(X_j), r(X_j)]$ *is a child interval of* $[l(X_i), r(X_i)]$*. Note that the parent of* $X_i \to X_a X_b$ *is* X_a*. Also, the terminal symbols correspond to the roots of the trees in the forest.*

▶ **Lemma 17.** *Let* X_i *be a symbol of a Lyndon SLP. We have* $\mathcal{P}_L(X_i) = \{X_{\ell_0}, \ldots, X_{\ell_k}\}$ *where* $X_{\ell_j} \to X_{\ell_{j-1}} X_{c_j}$ *for all* $j \in [1\,..\,k]$*,* X_{ℓ_0} *is a terminal symbol,* $X_{\ell_k} = X_i$*, and* $[X_{c_1}] \geq \cdots \geq [X_{c_k}]$ *is the Lyndon factorization of* $\mathrm{suf}_1([X_i])$*.*

Proof. Follows immediately from Lemma 3. ◀

4 Lexicographically sorting a Lyndon Grammar

Given a Lyndon SLP $\mathcal{G}$ with roots $r_1, \ldots, r_k$, we want to rename the symbols such that $[X_i] <_{lex} [X_j]$ holds if and only if $i < j$. This is possible in $\mathcal{O}(|\mathcal{G}|)$ using a similar principle to what is used for computing the Lyndon grouping in [2, 40]: consider a symbol X_i. By Lemma 17 and simple properties of Lyndon words, the symbols in $\mathcal{D}(X_i) \setminus \mathcal{P}_L(X_i)$ are lexicographically greater than X_i. Therefore, when considering the symbols in lexicographically decreasing order we can "induce" the position of a symbol $X_i \to X_a X_b$ upon encountering X_b: the symbols in $\mathcal{P}_L^{-1}(X_i)$ must be the lexicographically largest symbols with X_a as prefix symbol that have not yet been induced.

Algorithm 1 shows the procedure. Throughout, $\mathsf{A}[i]$ is either the ith symbol in the lexicographical order or $\perp$, and for each inserted X_i, $\mathsf{G}[i]$ is the index in A of the smallest child of X_i that has been inserted into A (or $r(X_i) + 1$ if no child has been inserted yet). In the first for-loop, all terminal symbols are inserted into A, and in the second for-loop, the remaining symbols are induced. Consequently, after the second for-loop, A contains the lexicographic order of the symbols, and G is (almost) it's inverse, i.e. $\mathsf{G}[\mathsf{A}[i]] = i + 1$.

There are two operations in Algorithm 1 that are not immediately obvious, namely finding $\left|\mathcal{P}_L^{-1}(X_i)\right|$ and iterating over all X_j with $X_j \to X_a X_{\mathsf{A}[i]}$ for a given $\mathsf{A}[i]$. For the first, note that the values $\left|\mathcal{P}_L^{-1}(X_i)\right|$ can be trivially computed in linear-time using the recurrence $\left|\mathcal{P}_L^{-1}(X_a)\right| = 1 + \sum_{X_i \to X_a X_b} \left|\mathcal{P}_L^{-1}(X_i)\right|$. Secondly, iterating efficiently over all X_j with $X_j \to X_a X_{\mathsf{A}[i]}$ can be achieved using a linear-time preprocessing step where we collect for each symbol X_i the symbols which have X_i as second symbol on the right-hand side.

The following theorem follows. Its proof is omitted due to space constraints.

▶ **Theorem 18.** *Algorithm 1 lexicographically sorts a Lyndon SLP in linear time.*

5 BWT construction

In this section, we describe an algorithm that derives the BBWT from the Lyndon SLP. Specifically, combining the second phase of `GSACA` [2, 40] with run-length encoding on the lexicographically sorted Lyndon SLP results in a very efficient algorithm on real data.

The following lemmas and corollary establish a relationship between the lexicographical order of suffixes and the lexicographical order of certain symbols of the Lyndon grammar.

Algorithm 1 Lexicographically sorting a Lyndon SLP.

$(\mathsf{A}, s) \leftarrow (\perp^n, 0)$;
for $X_i \to c$ *with* $c \in \Sigma$ **do** /* in increasing lexicographical order of c */
  $\mathsf{A}[s] \leftarrow i$; /* There are s symbols X with $[X][0] < c$ */
  $s \leftarrow s + \left|\mathcal{P}_L^{-1}(X_i)\right|$; /* There are $\left|\mathcal{P}_L^{-1}(X_i)\right|$ symbols X with $[X][0] = c$ */
  $\mathsf{G}[i] \leftarrow s$; /* X_i is the lex. smallest symbol in $\mathcal{P}_L^{-1}(X_i)$ */
end
for $i = n - 1 \to 0$ **do**
  for $X_j \to X_a X_{\mathsf{A}[i]}$ **do**
    // The symbols in $\mathcal{P}_L^{-1}(X_j)$ must be in $A\big[G[a] - \left|\mathcal{P}_L^{-1}(X_j)\right| .. G[a]\big)$
    $\mathsf{G}[j] \leftarrow \mathsf{G}[a]$;
    $\mathsf{G}[a] \leftarrow \mathsf{G}[a] - \left|\mathcal{P}_L^{-1}(X_j)\right|$;
    $\mathsf{A}[\mathsf{G}[a]] \leftarrow j$; /* X_j is the lex. smallest symbol in $\mathcal{P}_L^{-1}(X_j)$ */
  end
end

▶ **Lemma 19** ([40, Lemma 3.3]). *Let $\mathcal{L}_i$ be the longest prefix of $\mathrm{suf}_i(S)$ that is Lyndon. Then, $\mathcal{L}_i <_{lex} \mathcal{L}_j$ implies $\mathrm{suf}_i(S) <_{lex} \mathrm{suf}_j(S)$.*

▶ **Lemma 20.** *Consider an occurrence of $X_i \to X_a X_b$ at position j in the text S (this implies $S[j\,..\,j + |[X_i]|] = [X_i]$). Then, $[X_b]$ is the longest prefix of $\mathrm{suf}_{j+|[X_a]|}(S)$ that is Lyndon.*

Proof. Follows immediately from the relationship between Lyndon forest and Lyndon grammar [25] and [21, Lemma 15]. ◀

▶ **Corollary 21.** *Consider occurrences of $X_i \to X_a X_b$ at position j and of $X_{i'} \to X_{a'} X_{b'}$ at position j'. It holds $\mathrm{suf}_{j+|[X_a]|}(S) <_{lex} \mathrm{suf}_{j'+|[X_{a'}]|}(S)$ if $[X_b] <_{lex} [X_{b'}]$.*

To see why Corollary 21 is useful, consider a symbol $X_i \to X_a X_b$. Each occurrence of X_i in the derivation tree corresponds to a suffix of the text with prefix $[X_b]$, and each such suffix introduces an occurrence of the last character of $[X_a]$ to the BWT in the SA-interval of $[X_b]$. In our running example (cf. Figure 2), each occurrence of $X_8 \to X_7 X_9$ introduces a b in the SA-interval of $[X_9] = \mathsf{c}$.

In [40], an array $\mathsf{SA}'_\circ$[3] is computed such that $\mathsf{BBWT}[i] = S[\mathsf{SA}'_\circ[i]]$ holds for all i. This array arises from sorting the conjugates of the Lyndon factors of S and replacing each start position of a Lyndon word with the end position of the Lyndon word in S. It was shown that Algorithm 2 correctly computes the array $\mathsf{SA}'_\circ$ in linear time [40].[4] Basically, Lemma 19 implies that the positions in $\mathsf{SA}'_\circ$ are grouped by the longest Lyndon prefixes of the corresponding suffixes. The longest Lyndon prefix of $\mathrm{suf}_i(S)$ is $S[i\,..\,\mathrm{nss}[i])$. Because $\mathrm{suf}_{\mathrm{nss}[i]}(S) <_{lex} \mathrm{suf}_i(S)$ by definition, we can proceed by induction from lexicographically small to large.[5] Consider for instance the suffixes starting with c at indices 5 and 7 in our

[3] $\mathsf{SA}'_\circ$ is essentially the analogue to SA for conjugates of Lyndon factors and the infinite periodic order. A precise definition is omitted because only the relationship between $SA'_\circ$ and BBWT is relevant for this paper.
[4] In [40], Lyndon factors (roots of the Lyndon forest) come after every non-root representing the same Lyndon word. To reflect this, we use $\mathsf{L}[2(i-1)+1]$ if an occurrence of X_i is a root and $\mathsf{L}[2(i-1)]$ otherwise.
[5] There is a slight detail omitted here for clarity and brevity. Namely, that for the BBWT we are concerned with the *next smaller conjugate* of the respective Lyndon factor instead of the next smaller suffix [40].

Algorithm 2 Computing $\mathsf{SA}'_\circ$ from the lexicographically sorted Lyndon grammar [40, Algorithms 2 and 3].

```
(s, SA'_∘, L) ← (n, [], []^n);
for i = k → 1 do /* insert the end positions of the roots r_1, ..., r_k of
 the Lyndon SLP                                                        */
    L[2(r_i − 1) + 1].append(s);
    s ← s − |[X_{r_i}]|;
end
for m = 0 → 2g − 1 do
    foreach i in L[m] do
        SA'_∘.append(i);
        for j with nss[j] = i do
            Find X_k such that [X_k] = S[j .. nss[j]);
            L[2(k − 1)].append(j);
        end
    end
end
```

running example (cf. Figure 2). By Lemma 19, they are in the same "Lyndon group" [2] in the sorted list of suffixes because $\mathtt{c}$ is their longest Lyndon prefix. We have $\mathrm{nss}[5] = 5+|\mathtt{c}| = 6$ and $\mathrm{nss}[7] = 7+|\mathtt{c}| = 8$. When processing these Lyndon groups in lexicographically increasing order, the relative order of suffixes 6 and 8 is known at the point in time when the Lyndon group $\mathtt{c}$ is considered (because they are lexicographically smaller), and can therefore determine the lexicographical order of the suffixes 5 and 7.

We are now going to transform Algorithm 2 such that it outputs the BBWT instead of $\mathsf{SA}'_\circ$ and works without nss and the text indices. First, consider a text index $i \in [0\,..\,N)$ and let u_i be the highest node in the Lyndon forest at position i. (Equivalently, u_i corresponds to the longest Lyndon word starting at i.) Now consider the parent v_i of u_i. By its definition, it is clear that u_i is the right child of v_i. Let w_i be the left child of v_i. We can observe that, starting from w_i, the nodes on the rightmost path (excluding w_i) correspond exactly to the set $\{j \in [0\,..\,i) \mid \mathrm{nss}[j] = i\}$. For instance, in our running example there is a node labelled with $[X_8] = \mathtt{bc}$ at position 6 (cf. Figures 1 and 2). This node is the right child of a node labelled with $[X_6]$, which has a node labelled with $[X_5]$ as left child. There are two nodes on the rightmost path from this latter node, namely one at position 4 labelled with $[X_8] = \mathtt{bc}$ and one at position 5 labelled with $[X_9] = \mathtt{c}$. We consequently have $\mathrm{nss}[4] = \mathrm{nss}[5] = 6$.

Second, for $\mathsf{SA}'_\circ[k] = i$ we have $\mathsf{BBWT}[k] = S[i-1]$ and it follows that $\mathsf{BBWT}[k]$ is the last character of the string corresponding to v_i's left child w_i. Because of these two observations, we can replace the insertion of i in Algorithm 2 with the insertion of w_i. Consequently, we can reformulate Algorithm 2 to Algorithm 3.

▶ **Theorem 22.** *Algorithm 3 correctly computes* BBWT *from the sorted Lyndon SLP in* $\mathcal{O}(N)$ *time.*

Proof. The linear worst-case time complexity trivially follows from the fact that each iteration of the inner loop appends at least one character to BBWT. A proof of correctness is omitted due to space constraints. ◀

Note that runs in a list in L compound: If there is a run of symbol X_i, there is also a run of at least the same length of X_ℓ in X_r for each non-terminal $X_s \in \mathcal{P}_R(X_i)$ with $X_s \to X_\ell X_r$. Thus, Algorithm 3 often requires much fewer than N iterations.

Algorithm 3 Deriving the BBWT from the lexicographically sorted Lyndon grammar.

```
L ← []^{2g};                    /* initialize L[i] as an empty RLE list ∀i ∈ [0..2g) */
for i = 1 → k do A[2(r_i − 1) + 1].append(r_i); /* insert the roots r_1, ..., r_k */
for i = 0 → 2g − 1 do
    for (s, count) in L[i] do
        BBWT.append(last_char[s], count);
        while X_s is a non-terminal do /* walk the rightmost path from X_s */
            Let X_s → X_a X_b;
            A[2(b − 1)].append(a, count);
            s ← b;
        end
    end
end
```

5.1 Computing BWT and eBWT

When the text S is a Lyndon word, the BBWT of S is equal to the original BWT of S [3]. Since the BWT is independent of the rotation of the input and $\$S$ is a Lyndon word, we have $\mathsf{BBWT}(\$S) = \mathsf{BWT}(S\$)$. The latter is the \$-BWT commonly computed via the suffix array. Therefore, we can simply compute the \$-BWT by prepending a sentinel character to the text. Note that this is trivial and can be done after building the Lyndon grammar of S.

The eBWT of a set $\mathcal{S}$ of strings is the same as the BBWT of the string $\mathcal{S}^c$ that arises from arranging the canonical forms of the strings in $\mathcal{S}$ in lexicographically decreasing order [40]. In particular, the Lyndon factors of $\mathcal{S}^c$ are exactly the canonical forms of the input strings (assuming that the input strings are primitive). Note that, in Algorithm 3, sorting the canonical forms of the input strings is done implicitly in the first for-loop.

Note that the Lyndon grammars of the input strings are independent of each other (as long as equal Lyndon words are assigned equal grammar symbols). Therefore, we can also construct their Lyndon grammars independently of each other while using the same dictionary. As a consequence, it is easy to use our algorithms for parallel construction of the Lyndon grammar of a collection of sequences. Although our algorithm for deriving the BWT from the grammar is not parallelized, parallel construction of the Lyndon grammar leads to a substantial reduction in wall-clock-time because it is by far the most time-consuming part of the pipeline (see Section 7).

5.2 Other BWT variants for string collections

Besides the original eBWT, several other BWT variants for string collections have been proposed (see [14] for an overview). Several of the variants are especially suited to be computed with our algorithm, because there the input sequences $\mathcal{S} = \{S_1, \ldots, S_n\}$ can be parsed independently (and in parallel) like for the eBWT: The *dollar*-eBWT $\mathsf{dolEBWT}(\mathcal{S}) = \mathsf{eBWT}(\{S\$ \mid S \in \mathcal{S}\})$, the *multidollar* BWT $\mathsf{mdolBWT}(\mathcal{S}) = \mathsf{BWT}(S_1\$_1 \ldots S_n\$_n)$, and the *concatenated* BWT $\mathsf{concBWT}(\mathcal{S}) = \mathsf{BWT}(S_1\$ \ldots S_n\$\#)$, where $\# < \$$ and $\$_1 < \cdots < \$_n$ are

smaller than all characters in $\mathcal{S}$. This is done by constructing the Lyndon SLP of each S_i (with a shared dictionary) and then applying post-processing steps such that the desired eBWT variant can be derived from the resulting SLP. This is possible because the \$'s separate the input strings in the sense that no Lyndon word starting inside some S_i can contain a \$.

More specifically, for the doleBWT, no post-processing is required and we can just apply our eBWT construction algorithm, except that finding the canonical forms of the strings is trivial because $\mathsf{eBWT}(\{S\$ \mid S \in \mathcal{S}\}) = \mathsf{eBWT}(\{\$S \mid S \in \mathcal{S}\})$ and $\$S$ is a Lyndon word. For the mdolBWT and concBWT, we replace each S_i in $S_1\$_1 \dots S_n\$_n$ and $S_1\$ \dots \$S_n\$\#$, respectively, with the roots of the grammar generating S_i and compute the grammar of a rotation of the resulting string using Algorithm 3: both $\$_1S_2\$_2 \dots S_n\$_nS_1$ and $\#S_1\$ \dots \$S_n\$$ are Lyndon. Therefore, $\mathsf{BWT}(S_1\$_1 \dots S_n\$_n) = \mathsf{BBWT}(\$_1S_2\$_2 \dots S_n\$_nS_1)$ and $\mathsf{BWT}(S_1\$ \dots \$S_n\$\#) = \mathsf{BBWT}(\#S_1\$ \dots \$S_n\$)$ and we can apply our BBWT construction algorithm to these rotations.

Note that all other BWT variants for string collections given in [14] can be simulated with the multidollar BWT by using different relative orders of the separator symbols.

6 Practical Lyndon grammar construction

As shown in [1], the simple folklore algorithm for constructing the Lyndon array can be used to construct the Lyndon forest. With a fitting lookup data structure, we can trivially construct the Lyndon SLP instead. We essentially use Algorithm 4.

Algorithm 4 Prepending $S[i]$ [1].

```
Find c with [X_c] = S[i];                    /*Set c to the symbol generating S[i]*/
while stack is not empty do
    t ← stack.top();
    if [X_t] ≤_lex [X_c] then break;
    else
        stack.pop();
        Find X_c' with X_c' → X_c X_t;
        c ← c';
    end
end
stack.push(c);
```

There are two steps that have to be explained: First, one must be able to compare $[X_t]$ and $[X_c]$ lexicographically, and secondly, one must be able to find $X_{c'}$ with $X_{c'} \to X_cX_t$ (this is also called the *naming function*). For the former, we store for each symbol on the stack a fixed-length prefix of the generated string. In many cases, this is sufficient for determining the lexicographical order. Possibilities to handle the other cases are explained in Sections 6.1, 6.2 and 6.3. The following paragraphs will deal with the naming function.

For the naming function, we use a hash table to find the symbol names. This provides (expected) constant time lookup/insertion.[6] In practice however, N hash table lookups are unnecessarily slow, especially because we assume the input to be repetitive. In particular, having fewer but longer keys is cache friendlier and thus faster on modern computer architectures.

[6] Note that this is also possible in $\mathcal{O}(\log\log g)$ deterministic time with at most $2g + g\log g + o(g\log g))$ bits of memory, where g is the number of symbols of the grammar [42].

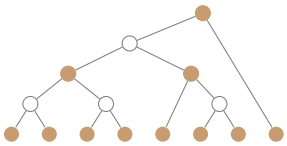


Figure 4 Partial representation of a Lyndon Forest with $n_{thres} = 2$. Heavy nodes are coloured in (■) while light nodes are not.

Specifically, we decrease the number of hash table accesses by partitioning the nodes of the Lyndon forest into *heavy* and *light* nodes and querying a second hash table H for each heavy node. The naming function is then only used when such a query to H does not return an answer. Leaves of the Lyndon forest are considered heavy. An inner node v is considered heavy, if and only if its number of *immediate heavy descendants* (heavy descendants for which only light nodes occur on the path to v) exceeds a fixed constant $n_{thres} > 1$ (see Figure 4).[7]

For a heavy node v of the Lyndon forest we query H with the sequence of immediate heavy descendants $v_1, \ldots, v_k$ of v. This means that we only store heavy nodes on the stack, for a light node we instead store all immediate heavy descendants.

Different occurrences of the same symbol represent the exact same Lyndon tree and therefore correspond to the exact same sequence of immediate heavy descendants. Consequently, if a subtree isomorphic to the subtree rooted at a heavy node has occurred previously, the query to H is sufficient to resolve it. Otherwise, we use the naming function to find the correct symbol and insert the sequence of immediate heavy descendants with it into H.

Note that each heavy node has at most $2n_{thres}$ immediate heavy descendants because it has exactly two children (or zero), each of which is either light (and thus has at most n_{thres} immediate heavy descendants) or heavy itself. This implies that the total memory usage increases by a factor of at most $\mathcal{O}(n_{thres})$.

6.1 Constant time suffix comparisons

Because the Lyndon forest (and thus the Lyndon SLP) is closely related to the Lyndon array, we can use the Lyndon array to decide whether $[X_t] \leq_{lex} [X_c]$ holds in constant time. Specifically, when considering $S[i]$ in Algorithm 3, we have $[X_c] <_{lex} [X_t]$ while $\lambda[i] > |[X_c]|$.

Unfortunately, as far as we know, all linear-time algorithms for constructing the Lyndon array directly from the text need random access to the text [1, 7, 19] and thus the text should reside in main memory.

For a BWT of a real-world string collection $\mathcal{S}$ however, this is typically not a problem because, as noted in Section 5.1, the SLPs of the strings in $\mathcal{S}$ can be computed independently (with a shared dictionary). Consequently, it suffices to have only the string (and its (succinct) Lyndon array) in main memory whose Lyndon SLP is currently computed. This increases the RAM usage by $n_{\max}(\lceil \log_2 |\Sigma| \rceil + 2)$ bits, where $n_{\max}$ is the length of the longest string in $\mathcal{S}$. For real biological data, $n_{\max}$ is generally small compared to the size of the entire input.

6.2 Naïve suffix comparisons using the Lyndon grammar

In this section, we describe a simple method for comparing $[X_t]$ and $[X_c]$ lexicographically which requires constant extra memory and works well in practice. We assume that $a, b < i$ holds for each rule $X_i \to X_a X_b$. This is clearly the case for SLPs constructed with Algorithm 4 if we assign increasing indices to new symbols.

[7] In our experiments, $n_{thres} = 31$.

First note that, because we assign equal Lyndon words to equal symbols, $[X_t] = [X_c]$ if and only if $t = c$. The key to our algorithm is that we find the longest common symbol $X_\ell \in \mathcal{P}_L(X_t) \cap \mathcal{P}_L(X_c)$. Additionally, we determine $X_{t'} \in \mathcal{P}_L(X_t)$ and $X_{c'} \in \mathcal{P}_L(X_c)$ with $X_{t'} \to X_\ell X_r$ and $X_{c'} \to X_\ell X_{r'}$. Note that, by definition, $r \neq r'$ (otherwise, $c' = t'$, contradicting the choice of ℓ) and therefore the lexicographical order of $[X_r]$ and $[X_{r'}]$ is the same as the lexicographical order of $[X_t]$ and $[X_c]$. Formally, $[X_t] <_{lex} [X_c] \iff [X_r] <_{lex} [X_{r'}]$. If such a tuple (ℓ, t', c') does not exist, either $[X_t]$ and $[X_c]$ do not share a non-empty prefix (i.e., $[X_t][0] \neq [X_c][0]$), or one of X_t and X_c is a prefix of the other.

Note that the indices of the elements in $\mathcal{P}_L(X_i)$ have the same relative order as the lengths of the generated strings in the sense that for all $a, b \in \mathcal{P}_L(X_i)$ we have $a < b$ if and only if $|[X_a]| < |[X_b]|$. For this reason, we can proceed with a two-pointer search to find ℓ (and t' and c'). More specifically, assuming that the desired $\ell \in \mathcal{P}_L(X_t) \cap \mathcal{P}_L(X_c)$ exists, we have $\ell \in \mathcal{P}_L(X_a)$ if $|[X_t]| < |[X_c]|$ for $X_c \to X_a X_b$, and vice versa. A concrete implementation is omitted due to space constraints.

Because in each step, at least one of the symbol indices decreases, the time complexity for a comparison is at most linear in the size of the SLP. In fact, because in each step we go from a symbol to one of its children, the time complexity is actually bounded by the *height* of the SLP. This in turn implies a time complexity of $\mathcal{O}(Ng)$ when using Algorithm 4 with the described method for constructing the Lyndon SLP.

Note that this worst-case time complexity is tight, e.g. on the string $a^k b a^k$ ($k \in \mathbb{N}$). However, the Lyndon forests of random strings have expected height proportional to the logarithm of the input size [36] and our approach works very well in practice (see Section 7).

6.3 Construction in $\mathcal{O}\left(N \log g + g \log^2 g\right)$

In this Section, we describe an algorithm that is able to compute the Lyndon grammar online in $\mathcal{O}\left(N \log g + g \log^2 g\right)$ deterministic time from right to left in a streaming fashion using $\mathcal{O}(g)$ words of extra memory.

Basically, this is done by maintaining the grammar's set of symbols $\mathcal{S}$ in an ordered sequence, lexicographically sorted by their respective generated strings. Using e.g. a B-Tree [5], one can then find the rank of any symbol in $\mathcal{O}(\log|\mathcal{S}|)$ time. Thus, determining the lexicographical order of two symbols in $\mathcal{S}$ can also be done in $\mathcal{O}(\log|\mathcal{S}|)$ time. What remains to be shown is how the symbols can be maintained in this sorted arrangement.

As shown in Section 3, a lexicographically sorted Lyndon grammar has a forest-structure (cf. Figure 3). This first-symbol forest can be represented using a *balanced parenthesis sequence (BPS)* of length $2|\mathcal{S}|$ [24], which can be obtained using a depth-first traversal of the first-symbol forest (starting at the roots) by writing an opening parenthesis `'('` when visiting a node for the first time and a closing parenthesis `')'` when all subtrees of a node have been visited [24]. Each parenthesis pair corresponds to a symbol in the grammar, where the ith opening parenthesis corresponds to the ith smallest symbol (by lexicographical order). For a grammar with symbol set $\mathcal{S}$, let this BPS be $\mathcal{B}_\mathcal{S}$. We represent $\mathcal{B}_\mathcal{S}$ as an ordered sequence $\mathcal{T}_\mathcal{S}$, which contains two markers $(_i$ and $)_i$ for each symbol X_i in $\mathcal{S}$, such that the ranks of $(_i$ and $)_i$ are the indices of X_i's opening and closing parenthesis in $\mathcal{B}_\mathcal{S}$, respectively. For instance, the sequence $\mathcal{T}_\mathcal{S}$ for the lexicographically sorted Lyndon grammar of our running example (see Figure 3) would be $(_1(_2(_3(_4)_4)_3)_2(_5(_6)_6)_5)_1(_7(_8)_8)_7(_9)_9$.

Now consider adding a new symbol $X_i \to X_a X_b$, where X_a and X_b are in $\mathcal{T}$ (i.e. $X_i \notin \mathcal{S}$, $X_a, X_b \in \mathcal{S}$). By Lemma 15, the parent of X_i's parenthesis pair in $\mathcal{B}_{\mathcal{S}\cup\{X_i\}}$ must be X_a's parenthesis pair. Therefore, it suffices to determine X_i's lexicographically smallest "sibling" $X_j \to X_a X_c$ ($X_j \in \mathcal{S}$) with $[X_c] >_{lex} [X_b]$; X_i's parenthesis pair must appear immediately

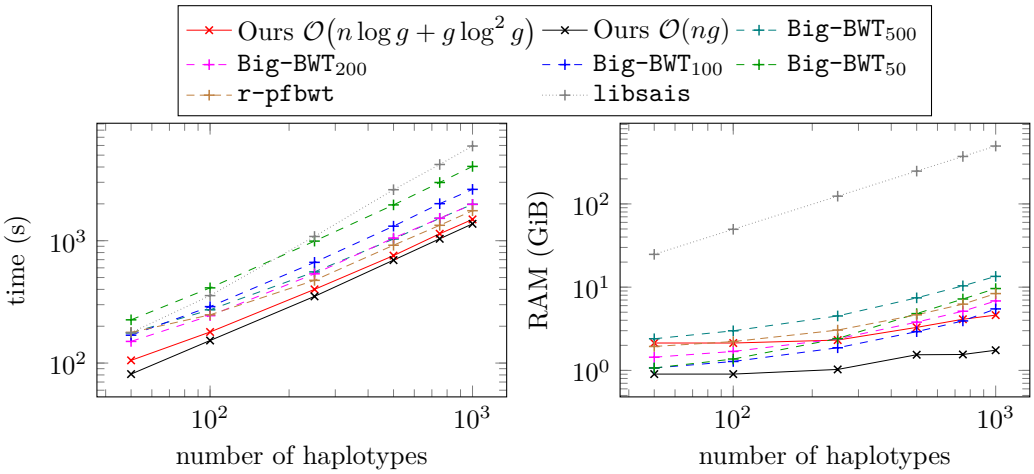


Figure 5 Wall clock time and maximum resident memory of BWT construction algorithms on chromosome 19 sequences. The sequences were concatenated and all programs used only one thread.

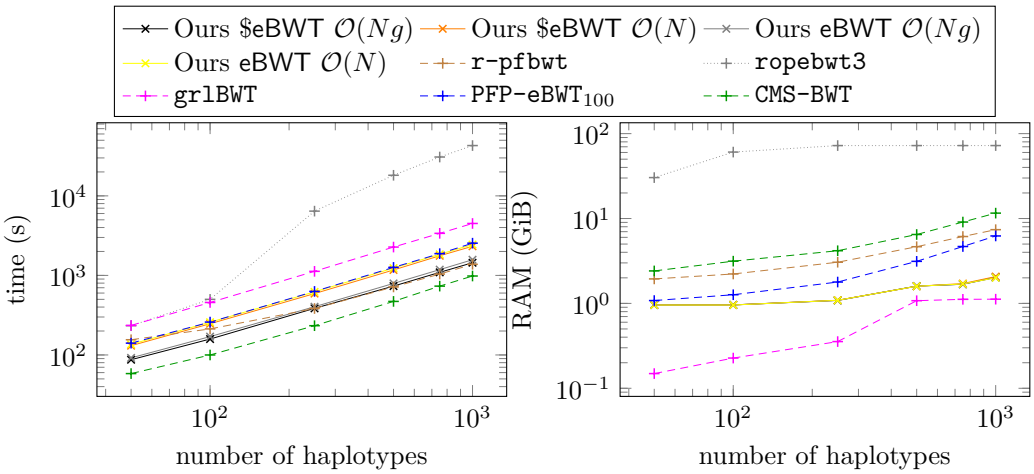


Figure 6 Wall clock time and maximum resident memory of tools for constructing BWT variants for string collection on chromosome 19 sequences. All programs used only one thread.

in front of $(_j$ in $\mathcal{T}_{\mathcal{S}\cup\{X_i\}}$. If there is no such sibling, X_i is (currently) the largest child of X_a and thus X_i's parenthesis pair must be immediately in front of $)_a$ instead. Note that comparing $[X_b]$ and $[X_c]$ is possible in $\mathcal{O}(\log|\mathcal{S}|)$ because both X_b and X_c are in $\mathcal{T}_{\mathcal{S}}$.

In order to be able to find the correct sibling of X_i as described, we additionally maintain for each X_a an ordered sequence $\mathcal{T}_a$ containing the symbols $\mathcal{S}_a = \{X_k \to X_a X_b \mid X_k, X_b \in \mathcal{S}\}$ in lexicographical order. Inserting $X_i \to X_a X_b$ into $\mathcal{T}_a$ can then be accomplished with $\mathcal{O}(\log|\mathcal{S}_a|) \subseteq \mathcal{O}(\log|\mathcal{S}|)$ comparisons, each of which is possible in $\mathcal{O}(\log|\mathcal{S}|)$ via $\mathcal{T}_{\mathcal{S}}$.

In total, we obtain a time complexity of $\mathcal{O}\big(|\mathcal{S}|\log^2|\mathcal{S}|\big)$ for maintaining the grammar's symbols lexicographically sorted, and $\mathcal{O}(N\log|\mathcal{S}|)$ for constructing the Lyndon forest.

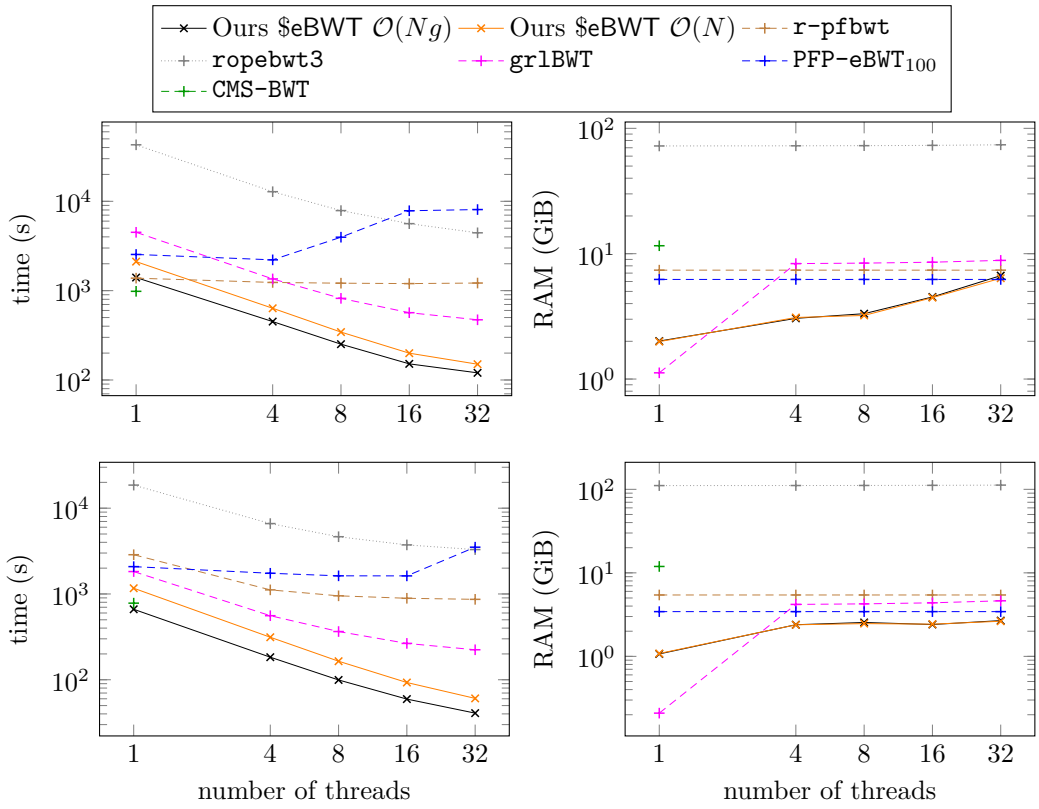


Figure 7 Wall clock time and maximum resident memory of tools for constructing BWT variants on 1000 Chromosome 19 sequences (top) and 10^6 SARS-CoV-2 sequences (bottom) using multiple threads. Note that `CMS-BWT` does not support multithreading. For the SARS-CoV-2 data, `PFP-eBWT` required the `--reads` flag.

7 Experiments

The source code of our implementation is publicly available.[8]

We compare our BWT algorithms for single texts with the programs `Big-BWT`[9] [11] and `r-pfbwt`[10] [41], as well as `libsais`.[11] The latter uses a modified version of the Suffix-Array Induced Sorting (SAIS) algorithm [37] to compute the BWT and, since it is based on the currently fastest SA construction implementation for general real-world data [40], can be viewed as a lower bound for algorithms using the suffix array to compute the BWT.

For the BWT of text collections, we compare with `PFP-eBWT`[12] [10], `r-pfbwt` [41], `ropebwt3`[13] [29], `grlBWT`[14] [17] and `CMS-BWT`[15] [35] (for `CMS-BWT` we used the first sequence in the collection as reference). All tool except the last one support multi-threading. Note

[8] `https://gitlab.com/qwerzuiop/lyndongrammar`
[9] `https://gitlab.com/manzai/Big-BWT`, last accessed: 22.04.2025, git hash `944cb27`
[10] `https://github.com/marco-oliva/r-pfbwt`, last accessed: 22.04.2025, git hash `1fea5c3`
[11] `https://github.com/IlyaGrebnov/libsais`, last accessed: 22.04.2025, git hash `a138159`
[12] `https://github.com/davidecenzato/PFP-eBWT`, last accessed: 22.04.2025, git hash `4ca75ce`
[13] `https://github.com/lh3/ropebwt3`, last accessed: 22.04.2025, git hash `36a6411`
[14] `https://github.com/ddiazdom/grlBWT`, last accessed: 22.04.2025, git hash `f09e7fa`
[15] `https://github.com/fmasillo/cms-bwt`, last accessed: 22.04.2025, git hash `1099d07`. Note that the speed and memory usage of `CMS-BWT` has improved massively since its publication in [35].

that not all of these tools compute the same BWT variant [14]. Also note that all algorithms based on PFP as well as `grlBWT` are semi-external, i.e., write/read some temporary data to/from disk.

As test data, we use up to 1000 human Chromosome 19 haplotypes from [12] ($\approx 6 \cdot 10^{10}$bp) and 10^6 SARS-CoV-2 sequences ($\approx 3 \cdot 10^{10}$bp).[16] All experiments were conducted on a Linux-6.8.0 machine with an Intel Xeon Gold 6338 CPU and 512 GB of RAM. All programs were compiled with GCC 13.3.0. Before each test, the test file was scanned once to ensure it is cached by the kernel. The results can be seen in Figures 5, 6 and 7. The subscripts for the PFP-based algorithms indicate the used modulus.

For our programs, computing the doleBWT using the suffix comparison method from Section 6.2 is generally the fastest. Computing the original eBWT is slightly slower because we need to find the smallest rotation of each input string. In the single-threaded case, for both the 1000 Chromosome 19 haplotypes and 10^6 SARS-CoV-2 sequences, over 98% of the time was spent constructing the grammar (for our fastest algorithm). Using the linear-time algorithm for constructing the Lyndon array from [7] to ensure expected linear time complexity slows our programs down by up to 60%. As expected, the suffix comparison method from Section 6.3 is much slower than our other methods. The increase in memory consumption of our program in the multithreaded case is due to the use of thread-safe hash tables and multiple sequences and stacks residing in main memory.

For the Chromosome 19 collections and a single thread, `CMS-BWT` is the fastest program (at the cost of high memory usage), followed by `r-pfbwt` (for larger cases) and our algorithms. For more threads, our program is always the fastest. Regarding the memory consumption, `grlBWT` uses the least amount of main memory for single-threaded processing, followed by our programs. For the SARS-CoV-2 sequences, our program is the fastest, especially with multiple threads, and for more than one thread also the most memory efficient.

8 Conclusion and Further Work

We described an algorithm to compute the BBWT – and by extension the common \$-BWT and various versions of the eBWT – from the lexicographically sorted Lyndon grammar of a text or text collection. Furthermore, we gave an algorithm that lexicographically sorts a Lyndon grammar and discussed approaches to efficiently compute the Lyndon grammar of a text or text collection. We implemented the described algorithms and found that they outperform other current state-of-the art programs in terms of time or memory consumed (often both).

References

1 Golnaz Badkobeh, Maxime Crochemore, Jonas Ellert, and Cyril Nicaud. Back-to-front online lyndon forest construction. In Hideo Bannai and Jan Holub, editors, *33rd Annual Symposium on Combinatorial Pattern Matching*, volume 223 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPICS.CPM.2022.13`.

2 Uwe Baier. Linear-time Suffix Sorting - A New Approach for Suffix Array Construction. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching*, volume 54 of *Leibniz International Proceedings in Informatics*, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CPM.2016.23`.

[16] Downloaded from `https://www.covid19dataportal.org` on 28.08.2024.

3 Hideo Bannai, Juha Kärkkäinen, Dominik Köppl, and Marcin Piątkowski. Constructing the Bijective and the Extended Burrows-Wheeler Transform in Linear Time. In Paweł Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching*, volume 191 of *Leibniz International Proceedings in Informatics*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CPM.2021.7.

4 Hélène Barcelo. On the action of the symmetric group on the free lie algebra and the partition lattice. *Journal of Combinatorial Theory, Series A*, 55(1):93–129, 1990. doi:10.1016/0097-3165(90)90050-7.

5 Rudolf Bayer and Edward McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, pages 107–141, 1970. doi:10.1145/1734663.1734671.

6 Nico Bertram, Jonas Ellert, and Johannes Fischer. Lyndon Words Accelerate Suffix Sorting. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms*, volume 204 of *Leibniz International Proceedings in Informatics*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.15.

7 Philip Bille, Jonas Ellert, Johannes Fischer, Inge Li Gørtz, Florian Kurpicz, J. Ian Munro, and Eva Rotenberg. Space Efficient Construction of Lyndon Arrays in Linear Time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming*, volume 168 of *Leibniz International Proceedings in Informatics*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.14.

8 Silvia Bonomo, Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Suffixes, Conjugates and Lyndon Words. In Marie-Pierre Béal and Olivier Carton, editors, *Developments in Language Theory*, pages 131–142, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-38771-5_13.

9 Silvia Bonomo, Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Sorting conjugates and suffixes of words in a multiset. *International Journal of Foundations of Computer Science*, 25(08):1161–1175, 2014. doi:10.1142/S0129054114400309.

10 Christina Boucher, Davide Cenzato, Zsuzsanna Lipták, Massimiliano Rossi, and Marinella Sciortino. Computing the original eBWT faster, simpler, and with less memory. In *International Symposium on String Processing and Information Retrieval*, pages 129–142. Springer, 2021. doi:10.1007/978-3-030-86692-1_11.

11 Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big BWTs. *Algorithms for Molecular Biology*, 14:1–15, 2019. doi:10.1186/s13015-019-0148-5.

12 Christina Boucher, Travis Gagie, I Tomohiro, Dominik Köppl, Ben Langmead, Giovanni Manzini, Gonzalo Navarro, Alejandro Pacheco, and Massimiliano Rossi. PHONI: Streamed matching statistics with multi-genome references. In *2021 Data Compression Conference*, pages 193–202. IEEE, 2021. doi:10.1109/dcc50243.2021.00027.

13 Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. *Digital SRC Research Report*, 124, 1994.

14 Davide Cenzato and Zsuzsanna Lipták. A survey of BWT variants for string collections. *Bioinformatics*, 40(7):btae333, 2024. doi:10.1093/bioinformatics/btae333.

15 Kuo Tsai Chen, Ralph H. Fox, and Roger C. Lyndon. Free differential calculus, iv. the quotient groups of the lower central series. *Annals of Mathematics*, pages 81–95, 1958. doi:10.2307/1970044.

16 Maxime Crochemore and Luís M.S. Russo. Cartesian and Lyndon trees. *Theoretical Computer Science*, 806, 2020. doi:10.1016/j.tcs.2018.08.011.

17 Diego Díaz-Domínguez and Gonzalo Navarro. Efficient construction of the BWT for repetitive text using string compression. *Information and Computation*, 294:105088, 2023. doi:10.1016/j.ic.2023.105088.

18 Jean-Pierre Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, 1983. doi:10.1016/0196-6774(83)90017-2.

19 Jonas Ellert. Lyndon Arrays Simplified. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms*, volume 244 of *Leibniz International Proceedings in Informatics*, pages 48:1–48:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.48.

20 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM (JACM)*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

21 Frantisek Franek, ASM Sohidull Islam, M Sohel Rahman, and William F Smyth. Algorithms to Compute the Lyndon Array. In Jan Holub and Jan Žďárek, editors, *Prague Stringology Conference*, pages 172–184, 2016. URL: http://www.stringology.org/event/2016/p15.html.

22 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in BWT-runs bounded space. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1459–1477. SIAM, 2018. doi:10.1137/1.9781611975031.96.

23 Joseph Yossi Gil and David Allen Scott. A bijective string sorting transform, 2012. doi:10.48550/arXiv.1201.3077.

24 Guy Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science*, pages 549–554, Los Alamitos, CA, USA, 1989. IEEE Computer Society. doi:10.1109/sfcs.1989.63533.

25 Tsuruta Kazuya, Köppl Dominik, Nakashima Yuto, Inenaga Shunsuke, Bannai Hideo, Takeda Masayuki, Tsuruta Kazuya, Köppl Dominik, Nakashima Yuto, Inenaga Shunsuke, Bannai Hideo, and Takeda Masayuki. Grammar-compressed Self-index with Lyndon Words. *Information Processing Society of Japan*, 13:84–92, 2020. URL: https://ipsj.ixsq.nii.ac.jp/records/206700.

26 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, Stoc 2018, pages 827–840, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188814.

27 Manfred Kufleitner. On bijective variants of the burrows-wheeler transform. In Jan Holub and Jan Žďárek, editors, *Prague Stringology Conference*, pages 65–79, Czech Technical University in Prague, Czech Republic, 2009. URL: http://www.stringology.org/event/2009/p07.html.

28 Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, 2012. doi:10.1038/nmeth.1923.

29 Heng Li. BWT construction and search at the terabase scale. *Bioinformatics*, 40(12):btae717, 2024. doi:10.1093/bioinformatics/btae717.

30 Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *bioinformatics*, 25(14):1754–1760, 2009. doi:10.1093/bioinformatics/btp324.

31 Felipe A. Louza, Sabrina Mantaci, Giovanni Manzini, Marinella Sciortino, and Guilherme P. Telles. Inducing the Lyndon Array. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String Processing and Information Retrieval*, pages 138–151, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-32686-9_10.

32 Sabrina Mantaci, Antonio Restivo, G. Rosone, and Marinella Sciortino. An Extension of the Burrows Wheeler Transform and Applications to Sequence Comparison and Data Compression. In Alberto Apostolico, Maxime Crochemore, and Kunsoo Park, editors, *16th Annual Symposium on Combinatorial Pattern Matching*, pages 178–189. Springer, 2005. doi:10.1007/11496656_16.

33 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Sorting Suffixes of a Text via its Lyndon Factorization. In Jan Holub and Jan Zdárek, editors, *Prague Stringology Conference*, pages 119–127, 2013. URL: http://www.stringology.org/event/2013/p11.html.

34 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Suffix array and Lyndon factorization of a text. *Journal of Discrete Algorithms*, 28:2–8, 2014. doi:10.1016/j.jda.2014.06.001.

35 Francesco Masillo. Matching Statistics Speed up BWT Construction. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms*, volume 274 of *Leibniz International Proceedings in Informatics*, pages 83:1–83:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.83.

36 Lucas Mercier and Philippe Chassaing. The height of the Lyndon tree. *Discrete Mathematics & Theoretical Computer Science*, DMTCS Proceedings vol. AS, 25th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2013), January 2013. doi:10.46298/dmtcs.2357.

37 Ge Nong, Sen Zhang, and Wai Hong Chan. Linear Suffix Array Construction by Almost Pure Induced-Sorting. In *2009 Data Compression Conference*, pages 193–202, 2009. doi:10.1109/dcc.2009.42.

38 Jannik Olbrich. lg. Software, swhId: swh:1:dir:0274cae4abb893b49bc035ef3a57986a418afe40 (visited on 2025-09-04). URL: https://gitlab.com/qwerzuiop/lyndongrammar, doi:10.4230/artifacts.24673.

39 Jannik Olbrich, Enno Ohlebusch, and Thomas Büchler. On the Optimisation of the GSACA Suffix Array Construction Algorithm. In Diego Arroyuelo and Barbara Poblete, editors, *String Processing and Information Retrieval*, pages 99–113, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-20643-6_8.

40 Jannik Olbrich, Enno Ohlebusch, and Thomas Büchler. Generic Non-recursive Suffix Array Construction. *ACM Transactions on Algorithms*, 20(2):18, 2024. doi:10.1145/3641854.

41 Marco Oliva, Travis Gagie, and Christina Boucher. Recursive prefix-free parsing for building big BWTs. In *2023 data compression conference*, pages 62–70. IEEE, 2023. doi:10.1109/DCC55655.2023.00014.

42 Yoshimasa Takabatake, Tomohiro I, and Hiroshi Sakamoto. A Space-Optimal Grammar Compression. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 67:1–67:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2017.67.

ε-Net Algorithm Implementation on Hyperbolic Surfaces

Vincent Despré
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Camille Lanuel
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Marc Pouget
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Monique Teillaud
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Abstract

We propose an implementation, using the CGAL library, of an algorithm to compute ε-nets on hyperbolic surfaces proposed by Despré, Lanuel and Teillaud [17]. We describe the data structure, detail the implemented algorithm and report experimental results on hyperbolic surfaces of genus 2. The implementation differs from the cited algorithm on several aspects. In particular, we use a different data structure, based on combinatorial maps, to represent a triangulation of a surface. We explain how to generate fundamental polygons to represent our input hyperbolic surfaces and the arithmetic issues related to the number type of the coordinates of their vertices.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Geometric topology

Keywords and phrases Hyperbolic surface, Delaunay triangulation, Data structure, Combinatorial map, Implementation, CGAL

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.61

Related Version *Full Version*: `https://hal.science/hal-04820353v1` [16]

Supplementary Material
Software: `https://github.com/camille-lanuel/ESA_2025_implementation_epsilon_net` [15]
archived at `swh:1:dir:2fab64276f7d193b0b712c135fa9eebba62f0509`

Funding This work was partially supported by grant ANR-23-CE48-0017 of the French National Research Agency ANR (project Abysm).

Acknowledgements Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

1 Introduction

Hyperbolic surfaces are well studied in mathematics since hyperbolic geometry is the natural geometry on surfaces of genus larger than one [26]. However, there are long standing open problems that could be experimentally investigated like finding the hyperbolic surface of genus 3 with the longest *systole* (i.e., the length of the shortest non-contractible closed geodesic) or the hyperbolic surface of genus 2 with the smallest diameter.

A few computational tools for hyperbolic surfaces have only been available recently. Iordanov and Teillaud [28] introduced a package in CGAL [29] to construct Delaunay triangulations with Bowyer's incremental algorithm [5], only for the Bolza surface, which is the most symmetric surface of genus two. The authors in [12] proposed an implementation of


© Vincent Despré, Camille Lanuel, Marc Pouget, and Monique Teillaud;
licensed under Creative Commons License CC-BY 4.0



edge flips to transform any triangulation of a closed hyperbolic surface into the Delaunay triangulation. Their software [13] also generates surfaces of genus 2. The algorithm does not support insertions of new points.

An ε-net (see Section 2.2) is a natural tool to approximate distances on a surface, therefore it helps to address the aforementioned open problems. Our contribution is the implementation[1] of an ε-net algorithm proposed in [17] inspired from Shewchuk's Delaunay refinement [35]. To the best of our knowledge, it is the first implementation for this problem.

There are two candidate data structures to represent a hyperbolic surface in this context: one focuses on a fundamental domain in the universal cover [17] and the other considers the surface as a combinatorial map [12]. We use an enriched version of the data structure implemented in the CGAL package *2D triangulations on hyperbolic surfaces* [13] based on a combinatorial map.

Our implementation is independent from the genus of the surface. However, the only tractable surfaces that we can currently generate have genus two. Indeed, except for the specific case of the Bolza surface mentioned above, which involves algebraic numbers, so far the only surfaces that are reachable by exact computations are a dense subset of the set of surfaces of genus two given by a domain with rational coordinates. As mentioned in [24], even for generalized Bolza surfaces, the representation of fundamental domains with algebraic numbers in genus higher than two is an obstacle to overcome when implementing.

The algorithm iteratively inserts in the triangulation the circumcenter of a *large triangle*, which is a triangle whose circumradius is greater than ε. Even though the coordinates of a circumcenter lie in an algebraic extension of degree two with respect to the coordinates of the vertices of the triangle, our algorithm only constructs points with exact rational coordinates that approximate the circumcenters. Consequently, our algorithm only uses exact rational arithmetic for the construction of the Delaunay triangulation. The final step is to check that the vertices of this triangulation actually form an ε-net in spite of the approximation of inserted circumcenters; this is the only place where we use algebraic numbers of degree 2. As explained in Section 7.1, it is very hard to sample the set of hyperbolic surfaces with a satisfactory distribution in practice. We use the generator introduced by Dubois et al. [12], which samples a dense subset of the set of surfaces of genus 2 and describes them by rational points (Section 5). Quite noticeably, a random hyperbolic surface is likely to have a reasonably long systole, which is actually the case in our experiments, and our final ε-net check is always successful. We observe in [16, Sec 7.3] that for a surface specifically designed to have a very small systole, the final ε-net check may fail, which means that a higher precision on the rational approximation of the circumcenter would be needed.

The key component of the algorithm, alongside the Delaunay flip part, is the point location. To insert a point in a triangulation of the surface, we work in the triangulation lifted in the hyperbolic plane $\mathbb{H}^2$, and determine the lift of a triangle containing the point (Section 2.1). Similar to the Euclidean plane, there are multiple strategies for traversing the triangulation and locate a point [23]. We tested both the straight and the visibility walks, which demonstrate comparable efficiency for our purposes. Theorem 1 ensures that the visibility walk terminates successfully in the context of finite or periodic Delaunay triangulations of $\mathbb{H}^2$.

The paper is organized as follows: We discuss point location strategies in triangulations in $\mathbb{H}^2$ in Section 3. We detail our data structure in Section 4. The generation of input surfaces and the related arithmetic issues are introduced in Section 5. Our implemented algorithm is presented in Section 6. Our experiments are reported in Section 7.

[1] `https://github.com/camille-lanuel/ESA_2025_implementation_epsilon_net`

2 Background and Notation

2.1 Hyperbolic Surfaces

We refer the reader to textbooks [9, 2, 7] for more details on topology and hyperbolic surfaces. In this paper, we use the term *hyperbolic surface* for a closed (connected, compact, and without boundary) oriented hyperbolic surface. Such a surface S can be seen as the quotient of the hyperbolic plane $\mathbb{H}^2$ under the action of a discrete subgroup Γ of the group of orientation-preserving isometries of $\mathbb{H}^2$. The surface S is locally isometric to $\mathbb{H}^2$ and thus has constant curvature -1. Note that Hilbert's theorem prevents to isometrically embed $\mathbb{H}^2$ or any hyperbolic surface in $\mathbb{R}^3$, thus all our illustrations will be drawn in the Poincaré disk model. The computation of distances in $\mathbb{H}^2$ will be detailed in Section 6.2. We denote as g the genus of S, that is its number of handles, and σ its systole which is the length of the shortest non-contractible closed geodesic.

The action of Γ on $\mathbb{H}^2$ induces a projection $\rho : \mathbb{H}^2 \mapsto \mathbb{H}^2/\Gamma = S$. For $x \in S$, an element $\widetilde{x} \in \Gamma x := \rho^{-1}(x) \subset \mathbb{H}^2$ is called a *lift* of x. Throughout this paper, objects written with a tilde $\widetilde{\cdot}$ are in $\mathbb{H}^2$ while objects on S are written without.

A *fundamental domain* for S is a connected subset of $\mathbb{H}^2$ containing exactly one lift of every point of S, except on its boundary. The *Dirichlet domain* $\mathcal{D}_{\widetilde{b}}$ of a point $\widetilde{b} \in \mathbb{H}^2$ is defined as the closed Voronoi cell of $\widetilde{b}$ in the Voronoi diagram of the point set $\Gamma\widetilde{b}$. It is a fundamental polygon for S: the interiors of two copies of the domain are disjoint, $\Gamma\mathcal{D}_{\widetilde{b}} = \mathbb{H}^2$, and its boundary is made of geodesic segments called *sides*. To each side s of $\mathcal{D}_{\widetilde{b}}$, there is a unique element $\gamma_s \in \Gamma$, called *side pairing*, such that $\gamma_s(s)$ is also a side. The side pairings generate Γ, so that $\mathcal{D}_{\widetilde{b}}$ and its side pairings determine the metric of the surface.

We work with the *Poincaré disk model* in which $\mathbb{H}^2$ is represented by the open unit disk of $\mathbb{C}$ (Figure 1). The geodesics are either diameters of the disk, or circular arcs orthogonal to the unit circle. Hyperbolic circles are Euclidean circles but their centers do not coincide in general.

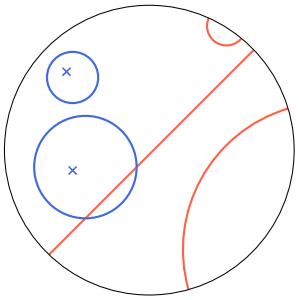


Figure 1 Geodesics (red) and hyperbolic circles (blue) in the Poincaré disk.

For a triangulation T of S, we note $\widetilde{T}$ the infinite periodic triangulation of $\mathbb{H}^2$ whose vertices, edges and faces are all lifts of those of T. Conversely, any vertex, edge or face of $\widetilde{T}$ projects on S as a vertex, edge or face of T. The triangulation T is a *Delaunay triangulation* if $\widetilde{T}$ is a Delaunay triangulation of $\mathbb{H}^2$, that is, no circumcircle of a triangle of $\widetilde{T}$ contains a vertex of $\widetilde{T}$ in its interior.

2.2 ε-Nets

Let us recall definitions [8]. Let (X, d) be a metric space and $\varepsilon > 0$. A subset $P \subset X$ is an *ε-covering* if $d(x, P) \leqslant \varepsilon$ for all $x \in X$, i.e., the closed balls of radius ε centered at points of P cover X. It is an *ε-packing* if $d(p, q) \geqslant \varepsilon$ for all $p \neq q \in P$, that is, the open balls of radius $\varepsilon/2$ centered at points of P are pairwise disjoint. If P is both an ε-covering and an ε-packing, then it is an *ε-net*, also known as an $(\varepsilon, \varepsilon)$-Delone set.

When $\varepsilon < \sigma$, we say that S is *ε-thick*. The number of points in an ε-packing of a hyperbolic surface S is upper-bounded by $16(g-1)\left(1/\varepsilon^2 + 1/\sigma^2\right)$, or $16(g-1)/\varepsilon^2$ for an ε-thick surface [17].

2.3 Original Algorithm

Let us summarize the algorithm [17], from which our implementation derives. The algorithm is inspired by Shewchuk's Delaunay refinement [35]. It starts with a Delaunay triangulation of S with a single vertex b. In a nutshell, as long as there is a large triangle in the triangulation, its circumcenter is inserted and the triangulation is updated (see Figure 2): the triangle in which the circumcenter lies is first split into three new triangles, then the Delaunay property is retrieved using edge flips [30, 18]. The set of vertices of the final Delaunay triangulation is an ε-net of S, and all the intermediate sets of vertices are ε-packings.

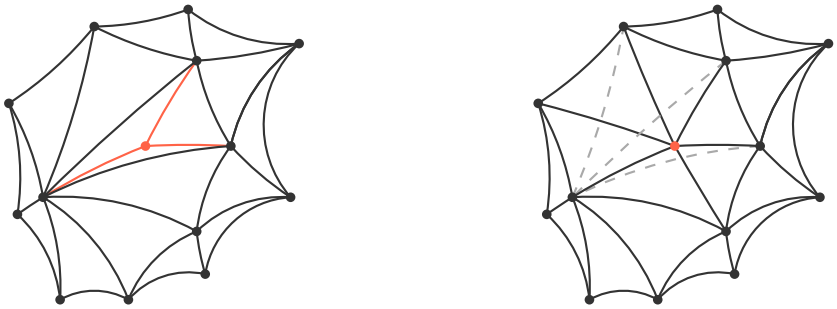


Figure 2 Insertion of a point in a Delaunay triangulation using a flip algorithm.

The algorithm also stores the Dirichlet domain $\mathcal{D}_{\tilde{b}}$ of a lift $\widetilde{b}$ of b together with its side pairings, which can be computed from any fundamental domain for S [14]. The data structure stores the lift in $\mathcal{D}_{\tilde{b}}$ of each vertex of the triangulation (or one of its lifts if it lies on the boundary of $\mathcal{D}_{\tilde{b}}$), and for each triangle, the information needed to retrieve its (at most three) lifts having at least one vertex in $\mathcal{D}_{\tilde{b}}$.

The most crucial step of the algorithm is the location of the circumcenter c_Δ of a triangle Δ in a triangulation of S. To this aim, a lift of c_Δ is located in the lifted triangulation in $\mathbb{H}^2$, as follows. First, the circumcenter $\widetilde{c}_\Delta$ of a lift $\widetilde{\Delta}$ of Δ with at least one vertex in $\mathcal{D}_{\tilde{b}}$ is computed. The point $\widetilde{c}_\Delta$ is a lift of c_Δ that does not necessarily lie in $\mathcal{D}_{\tilde{b}}$. To locate $\widetilde{c}_\Delta$, the algorithm first walks in the tiling $\Gamma\mathcal{D}_{\tilde{b}}$ to find the element $\gamma_c \in \Gamma$ such that $\widetilde{c}_\Delta \in \gamma_c \mathcal{D}_{\tilde{b}}$. The lift $\widetilde{c_b} = \gamma_c^{-1}\widetilde{c}_\Delta$ of c_Δ lying in $\mathcal{D}_{\tilde{b}}$ can then be computed. Second, the triangle in $\mathcal{D}_{\tilde{b}}$ in which $\widetilde{c_b}$ lies is identified by checking, for each triangle, its (at most three) lifts having at least one vertex in $\mathcal{D}_{\tilde{b}}$. It can be shown that the walk in the tiling $\Gamma\mathcal{D}_{\tilde{b}}$ traverses a bounded number of Dirichlet domains. The complexity of the algorithm is bounded by $O\left(N^2\right)$, where N is the number of points in the ε-net.

3 Preliminary Result: Walking in a Triangulation in $\mathbb{H}^2$

As mentioned above, a crucial step of the algorithm relies on finding a triangle containing a given point in a triangulation of $\mathbb{H}^2$ (there is only one such triangle, except for collinear points). There are two classical algorithms in the Euclidean plane: the straight walk and the visibility walk [23], which can be adapted to the hyperbolic plane. Both methods find the triangle containing a query point $\widetilde{q}$ in a triangulation starting from a vertex $\widetilde{p}$ of a triangle $\widetilde{\Delta}$. The *straight walk* visits all triangles along the geodesic segment $\widetilde{pq}$. The algorithm starts by rotating around $\widetilde{p}$ to find a triangle incident to $\widetilde{p}$ that has an edge intersecting the geodesic segment $\widetilde{pq}$. The *visibility walk* consists, for each visited triangle not containing $\widetilde{q}$, of moving to a neighbor through an edge $\widetilde{e}$ if $\widetilde{q}$ and the third vertex of the visited triangle are on different sides of the geodesic line supporting $\widetilde{e}$.

In the Euclidean case, the visibility walk terminates in a Delaunay triangulation [25] but it can loop forever in a non-Delaunay triangulation [11]. The following theorem takes care of the hyperbolic case.

▶ **Theorem 1.** *The visibility walk terminates in a finite or periodic hyperbolic Delaunay triangulation.*

The proof follows the same logic as in the Euclidean case [22] but it requires a new definition of the power of a point with respect to a circle, adapted to the hyperbolic case, which requires to rewrite the details. Due to lack of space, the proof is given in the full version of this article [16, Sec 3].

4 Data Structure

We enrich the data structure of the CGAL package *2D triangulations on hyperbolic surfaces* [13] based on a combinatorial map. A *combinatorial map* is an edge-centered data structure based on *darts*, which are equivalent to half-edges in our setting [31, 10]. A dart can be seen as an oriented edge. In dimension 2, each dart has a pointer β_1 to access the dart of the next edge in the same face, and a pointer β_2 to access the dart of the same edge in the adjacent face, as illustrated in Figure 3. Following β_1 pointers, we obtain all the darts of a given face. Following $\beta_1 \circ \beta_2$ combinations of pointers, we obtain all the darts of a given vertex.

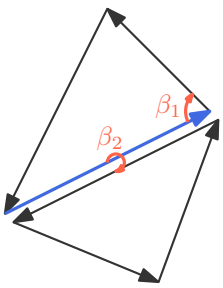


Figure 3 A dart (bold blue) in a 2D combinatorial map, and its pointers.

In the CGAL package *2D triangulations on hyperbolic surfaces* [13], the geometric information of a triangulation T is given as a cross-ratio for each edge, which is associated with darts. The *cross-ratio* of four pairwise distinct points z_1, z_2, z_3, z_4 in the Poincaré disk is defined as $[z_1, z_2, z_3, z_4] = \dfrac{(z_4 - z_2)(z_3 - z_1)}{(z_4 - z_1)(z_3 - z_2)} \in \mathbb{C}$. Let $\widetilde{e} = (\widetilde{p}, \widetilde{r})$ be a lift of an edge e and

$\widetilde{q}, \widetilde{s}$ be the remaining vertices of the lifted triangles incident to $\widetilde{e}$, such that $\widetilde{p}, \widetilde{q}, \widetilde{r}, \widetilde{s}$ are oriented counter-clockwise. The cross-ratio of e in T is $R_T(e) = [\widetilde{p}, \widetilde{q}, \widetilde{r}, \widetilde{s}]$. It is invariant under orientation-preserving isometries, so, the cross-ratio of an edge e can be defined from any of its lifts. The imaginary part of $R_T(e)$ is positive if and only if $\widetilde{s}$ lies in the circumdisk of the triangle $(\widetilde{p}, \widetilde{q}, \widetilde{r})$, i.e., if and only if the edge e is Delaunay flippable.

In addition to cross-ratios, the data structure of [13] contains one anchor, which represents a lift of one triangle in the Poincaré disk. The *anchor* is composed of a dart representing the triangle in the combinatorial map, and of the coordinates of the three vertices $\widetilde{p}, \widetilde{q}, \widetilde{r}$ of a lift of the triangle. The dart corresponds to the edge (p, q). Knowing a lift of a triangle, its neighbors can be retrieved using the cross-ratios of its edges: if $(\widetilde{p}, \widetilde{q}, \widetilde{r})$ and $(\widetilde{p}, \widetilde{r}, \widetilde{s})$ are two triangles in $\mathbb{H}^2$ sharing the edge $(\widetilde{p}, \widetilde{r})$, the coordinates of $\widetilde{s}$ can be deduced from $\widetilde{p}, \widetilde{q}, \widetilde{r}$ and $R_T((p, r))$. The anchor can therefore be used to compute a part of $\widetilde{T}$ one step at a time, for example to draw a lift of each triangle of T.

Our algorithm (see Section 6) performs computations with lifts of triangles at each step, in particular to compute a lift of the circumcenter of a triangle and locate it. To have constant time access to a lift of any triangle, we actually store an anchor for *each* face of the triangulation. The three darts of each triangular face are associated with its anchor.

Note that the set of anchors of all faces of T is not necessarily connected.

Updating the Data Structure

In the ε-net algorithm, the data structure is modified by two operations: splitting a triangle into three new triangles, and flipping an edge.

When splitting a triangle Δ, we know the lift $\widetilde{\Delta} = (\widetilde{p}, \widetilde{q}, \widetilde{r})$ given by its anchor and the point $\widetilde{s}$ to be inserted in $\widetilde{\Delta}$. The darts of Δ are kept and three pairs of darts are created (Figure 4). The pointers β_1 and β_2 of all these darts are set or updated to create the three new triangles in the combinatorial map. We create three new anchors corresponding to the new triangles $(\widetilde{p}, \widetilde{q}, \widetilde{s})$, $(\widetilde{q}, \widetilde{r}, \widetilde{s})$ and $(\widetilde{r}, \widetilde{p}, \widetilde{s})$ and associate them to the darts of their respective triangles. The cross-ratios of the three new edges are computed from the coordinates of $\widetilde{p}, \widetilde{q}, \widetilde{r}$, and $\widetilde{s}$. The cross-ratios of the edges $(p, q), (q, r)$, and (r, p) must be updated. For the edge (p, q), for instance, we use its non-updated cross-ratio and the vertices $\widetilde{p}, \widetilde{q}$ and $\widetilde{r}$ to compute the coordinates of a lift of the third vertex of its neighboring incident triangle. This step is mandatory because the anchor associated with the adjacent triangle in the combinatorial map may store a non-adjacent lift in $\mathbb{H}^2$. The new value of the cross-ratio can then be computed.

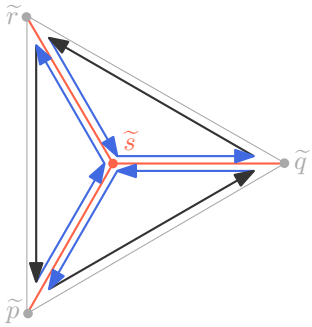


Figure 4 Splitting a triangular face in three in the combinatorial map, new darts are in blue.

To flip an edge, we use the CGAL package [13], which modifies the β_1 pointers and updates the cross-ratios [12]. We still need to update the anchors. To do so, we get the coordinates of the points stored in the anchor of a dart δ of the edge being flipped. We call them $\widetilde{p}, \widetilde{q}, \widetilde{r}$,

in such a way that the edge $(\widetilde{r},\widetilde{p})$ is represented by δ in the combinatorial map. Using the cross-ratio of that edge, we compute the coordinates of $\widetilde{s}$, the third vertex of the other lifted triangle sharing the edge $(\widetilde{r},\widetilde{p})$ (Figure 5). We then create two new anchors corresponding to the lifts of the triangles obtained after the flip, $(\widetilde{q},\widetilde{s},\widetilde{p})$ and $(\widetilde{q},\widetilde{s},\widetilde{r})$ and we associate them to the darts of their respective triangles.

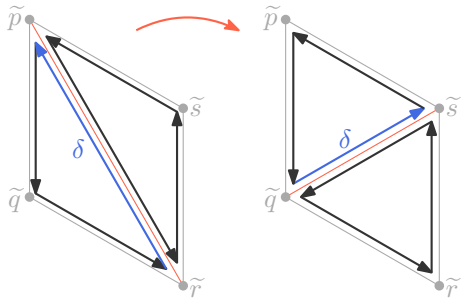


Figure 5 A flip. No darts are added or removed, only adjacencies are modified.

Note that, though these operations remove triangles and create new ones, darts are added but never removed in the data structure.

5 Generation of Input Surfaces

The input of our algorithm is a Delaunay triangulation of a surface with a single vertex. It is constructed from a fundamental domain with all its vertices in $\mathbb{H}^2$ projecting to the same point on the surface. We now explain how to generate such fundamental domains and the arithmetic issues related to the number type of the coordinates of its vertices.

The Teichmüller space $\mathcal{T}_g$, the space of all hyperbolic surfaces of genus g, can be parameterized for instance from a pants decomposition (Fenchel-Nielsen coordinates) [7, § 1.7], its group of isometries (Fricke coordinates) [27, § 2.5], or by a fundamental polygon in $\mathbb{H}^2$ [7, § 6]. Since we aim at computing a triangulation in $\mathbb{H}^2$, starting from a fundamental polygon is the right choice to avoid the use of hyperbolic trigonometric functions. Our algorithm relies on computations of cross-ratios and points in $\mathbb{H}^2$. There is no point in computing with `float` or `double` number types, as this is notoriously unstable.[2] Previous work showed that Delaunay triangulations can be computed on the Bolza surface, represented by a fundamental polygon with very specific algebraic numbers [28, 29]. Even when computing the Delaunay triangulation of points with rational coordinates, the group of the surface leads to algebraic numbers. In practice, algebraic numbers are handled with the CORE library [20] but this approach was shown to fail for generalized Bolza surfaces, already for genus 3 [24].

We now detail a way to generate and compute with generic surfaces of genus 2. Any genus 2 surface has a fundamental domain that is a symmetric octagon centered at the origin [1]. More precisely, three points are first chosen in the upper half of the Poincaré disk. Then a fourth point is computed so that the octagon formed by these four points and the four symmetric points with respect to the origin is a fundamental domain. The Teichmüller space $\mathcal{T}_2$ is thus parameterized by three complex numbers. Restricting to complex numbers with rational real and imaginary parts gives a dense subset of the Teichmüller space $\mathcal{T}_2$ [12]. This means that for any genus 2 surface, we can work on an arbitrary close surface given

[2] `https://www.cgal.org/exact.html`

by a fundamental domain whose vertices have rational coordinates. This allows us to only use exact rational computations for constructing an ε-net (see Section 6.2) and algebraic extensions of degree 2 to check its correctness (see Section 6.4).

A Delaunay triangulation of this octagon is computed as in the CGAL package [13, 12]: the eight vertices of the domain actually project on a same point on the surface, which we denote as b. Triangulating this convex octagon in an arbitrary way gives a triangulation of the surface with one vertex. The next step is to compute the cross-ratios of all edges. Then the Delaunay triangulation is computed using flips. The data structure of the CGAL package [13] is a combinatorial map with a cross-ratio on each edge, and one anchor. To generate the input for our ε-net algorithm, we only have to add an anchor for each face of the triangulation. Since the initial triangulation has only one vertex and six faces, the remaining anchors are computed (and associated with the darts of their respective faces) using cross-ratios, by successively computing lifts of adjacent triangles, as detailed in Section 4.

For higher genus surfaces, a construction of fundamental polygons is described in [36, § 6.11], but unfortunately it is not clearly leading to tractable numbers. We are investigating how polygons with rational coordinates could be generated since in such a case our algorithm will have the same robustness properties as those we illustrate in genus 2.

6 The Implemented Algorithm

The data structure is implemented in `Anchored_hyperbolic_surface_triangulation_2`, a class inherited from the CGAL class `Triangulation_on_hyperbolic_surface_2` [13]. The algorithm to compute an ε-net is implemented in the `epsilon_net(epsilon)` method. It maintains a Delaunay triangulation and iteratively inserts circumcenters of large triangles. We first detail our processing of large triangles aiming at minimizing the number of computations of circumcenters (Section 6.1). We then detail the largeness test, the representation of coordinates of points in $\mathbb{H}^2$ by exact rational numbers and the possible issues of the approximation of circumcenters (Section 6.2). The point location is detailed in Section 6.3. Finally, we explain how to certify that the set of vertices of the output triangulation is an ε-net using exact computation in degree two algebraic extensions (Section 6.4).

6.1 Additional Data Structure

At each step of the algorithm, a large triangle is considered. In the original algorithm, all triangles of the current triangulation are checked until a large one is found. Consequently, all triangles that are not affected by an insertion will be checked again for the next insertion. This choice has no effect on the theoretical complexity of the algorithm, but it is time-consuming in practice. Instead, we maintain a list of triangles to be processed by the algorithm. We ran experiments to guide our choice towards an effective way of maintaining this list. The details of these experiments can be found in [16, App A]. The results show that any attempt to maintain the list using criteria involving circumradii, like storing only large triangles or ordering them according to their circumradius, is highly inefficient.

More precisely, we maintain a list $\mathcal{L}$ of darts representing the triangles to be processed. The only operations on the list are: pop the front dart, and push new darts to the back. Each dart has a mark represented by a Boolean and we maintain the property that each triangle represented in $\mathcal{L}$ has exactly one marked dart; A triangle can have several darts in $\mathcal{L}$, but only one is marked. The computation of a circumradius is *only* performed when a *marked* dart is popped from the front. The list is initialized with one marked dart for each triangle of the input triangulation. Remark that for a reasonable choice of ε, all these triangles are large.

When a dart is popped out from $\mathcal{L}$, if it is marked, we unmark it and the circumradius of the triangle it represents is computed. If the triangle is large, its circumcenter splits the triangle containing it as detailed in Section 4, and one dart is marked and pushed to the back of $\mathcal{L}$ for each of the three new triangles. Darts are added to $\mathcal{L}$ without checking the size of the circumradius of the corresponding triangle and without checking if it is a Delaunay triangle. The Delaunay property is restored using the flip algorithm described in [12] and implemented in [13]. After a flip, in each of the two new triangles, we maintain the property that only one dart is marked. Indeed, such a new triangle may have 0, 1 or 2 marked darts (Figure 6). If it has none, then we mark one and push it to the back of $\mathcal{L}$; if it has two, then we leave them in $\mathcal{L}$ and unmark one; otherwise there is nothing to do.

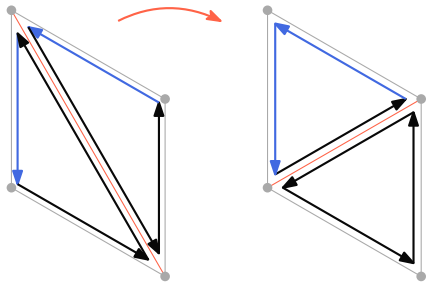


Figure 6 Case when triangles created by a flip have one or two marked darts.

When a non-marked dart is popped out from $\mathcal{L}$, then nothing is done: it means that the triangle it belongs to is represented by another marked dart in $\mathcal{L}$.

The algorithm proceeds with the new front dart until $\mathcal{L}$ is empty. Note that in practice, since darts representing new triangles are always pushed at the end of $\mathcal{L}$, large triangles tend to be close to the front and are processed before triangles with smaller circumradii.

Remark that no element of $\mathcal{L}$ is ever removed from the list, except at the front. As explained in Section 4, when a triangle disappears from the triangulation, its darts stay in the data structure, but their β_1 pointers are modified. So, it can be the case that a dart represented a large triangle when it was inserted in $\mathcal{L}$, but it represents a triangle whose circumradius is not greater than ε when it comes to the front of $\mathcal{L}$.

6.2 Circumcenters and Circumradii

When a marked dart is popped out from $\mathcal{L}$, the circumcenter of a lift of the triangle represented by the dart is first constructed in order to compute its circumradius. The lifted triangle is the one stored in the anchor associated to the dart. The coordinates of a circumcenter of three vertices with rational coordinates are algebraic numbers of degree two [4]. When it is inserted in the triangulation, a circumcenter becomes a vertex. Handling exact circumcenters would thus lead to cascading the algebraic degrees of coordinates, which would result in computations that are known to be impossible to handle in practice.

The coordinates of the circumcenter $\widetilde{c}_\Delta$ of a triangle $\widetilde{\Delta}$ are represented by the type `CGAL::Sqrt_extension`, which handles algebraic numbers of degree two. We do not insert the point $\widetilde{c}_\Delta$ but a rational approximation denoted $\widetilde{c}$. We round each coordinate of $\widetilde{c}_\Delta$ to a double precision number using the `to_double()` method of CGAL and convert it to an exact rational type. More specifically, coordinates of all vertices of our triangulation are represented by the `CGAL::Exact_rational` number type, which is a wrapper for the arbitrary-precision rational type `mpq_t` provided by GMP [21]. All the computations on our data structure are

performed using this `CGAL::Exact_rational` number type for points and cross-ratios. Note that the bitsize of the rationals encoding an inserted point is bounded whereas the bitsize of the rationals encoding a cross-ratio grows when it is updated during a flip.

Let us detail the computation of the largeness test for triangles that we use to avoid as much as possible the use of hyperbolic trigonometric functions. In the Poincaré disk, the distance between two points $\widetilde{u}$ and $\widetilde{v}$ is $d_{\mathbb{H}^2}(\widetilde{u},\widetilde{v}) = \operatorname{arcosh}(1+\lambda(\widetilde{u},\widetilde{v}))$, where $\lambda(\widetilde{u},\widetilde{v}) = \dfrac{2||\widetilde{u}-\widetilde{v}||^2}{(1-||\widetilde{u}||^2)(1-||\widetilde{v}||^2)}$ and $||\cdot||$ is the Euclidean distance. We compute the minimum λ between the approximate circumcenter $\widetilde{c}$ and the three vertices of the anchor. If it is greater than a certified upper bound of $\cosh(\varepsilon)-1$, then we consider the triangle large and insert $\widetilde{c}$ in the triangulation. The certified upper bound on $\cosh(\varepsilon)$ is computed with the Boost interval library [19]. Due to approximations for the computation of $\widetilde{c}$, this process may miss a large triangle and thus does not enforce the ε-covering property. Similarly, even if the point $\widetilde{c}$ that is inserted into the triangulation is at distance at most ε from the vertices of its triangle, it may be at distance smaller than ε from another vertex of the triangulation. We will check these properties in the final step of the algorithm (Section 6.4).

6.3 Point Location

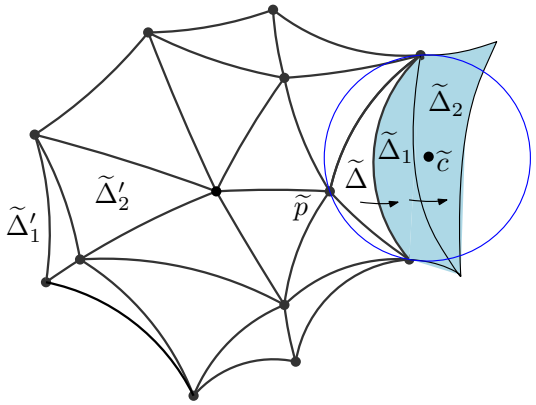


Figure 7 A fundamental domain of the surface is given by the anchors of the triangles (thick black lines). The blue triangles are the lifts constructed by the visibility walk to reach the approximate circumcenter $\widetilde{c}$ of $\widetilde{\Delta}$. Triangles $\widetilde{\Delta}'_1$ and $\widetilde{\Delta}'_2$ are the lifts of Δ_1 and Δ_2 in the domain.

Our implementation maintains a Delaunay triangulation of S using the data structure presented in Section 4. Each point to be inserted is located in this triangulation by locating a lift in the lifted triangulation. We do not store a Dirichlet domain as in the original algorithm (Section 2.3), but instead directly walk in the current triangulation itself.

We tested two walk algorithms: the straight walk and the visibility walk (Figure 7) mentioned in Section 3. In our case, the query point $\widetilde{c}$ is the approximate circumcenter of the lifted triangle $\widetilde{\Delta}$ given by the anchor of the triangle Δ being processed by the algorithm. The walk starts from a vertex $\widetilde{p}$ of the triangle $\widetilde{\Delta}$. Lifts of triangles must be constructed along the walk to find the lifted triangle containing $\widetilde{c}$.

Both walks are based on orientation tests in $\mathbb{H}^2$ with vertices of lifted triangles. The combinatorial map encoding the triangulation provides a direct access to the neighbor Δ' of the triangle Δ adjacent through one of its edges. However, the lift $\widetilde{\Delta}'$ of this triangle given by its anchor may not be adjacent to the lift $\widetilde{\Delta}$. The lift of Δ' that is adjacent to $\widetilde{\Delta}$ is computed from the vertices of $\widetilde{\Delta}$ and the cross-ratio of the common edge as explained in Section 4.

Running the `epsilon_net` method with both walks shows that they perform equivalently (see [16, App B]): the running time of the method remains the same, and they compute a similar number of lifts. The visibility walk does not have to handle the degenerate cases of the straight walk, which may go through a vertex or along an edge; we therefore choose the visibility walk. The bound on the length of the straight walk for the original algorithm, using Dirichlet domains, does not apply to the visibility walk. However we observe in Section 7.2 that the walk has constant length in practice.

6.4 ε-Covering and ε-Packing Checks

As explained in Section 6.2, our algorithm is robust in the sense that the output triangulation is the Delaunay triangulation of a set of points with rational coordinates. On the other hand, the vertices of this triangulation may not be an ε-net, due to approximations of circumcenters. To check the ε-covering property, it is sufficient to check that there is no large triangle. For every triangle Δ, we compute the exact circumcenter $\widetilde{c}_\Delta$ of its anchor, which has coordinates in a degree two algebraic extension, using the number type `CGAL::Sqrt_extension`. We then compute $\lambda(\widetilde{c}_\Delta, \widetilde{v})$ with the same type (see Section 6.2) for a vertex $\widetilde{v}$ of the triangle, and check that it is less than a certified lower bound on $\cosh(\varepsilon) - 1$. To check the ε-packing property, it is sufficient to check that all Delaunay edges are longer than ε. Since all vertices have rational coordinates, the value $\lambda(\widetilde{v}_i, \widetilde{v}_j)$ for two vertices of an edge is rational and it is checked to be larger than the upper bound on $\cosh(\varepsilon) - 1$ computed with the Boost interval library [19]. When these tests succeed, the vertices of the output triangulation are then certified to be an ε-net of the surface. Otherwise, the failure of these tests means that the double precision used for the approximation of circumcenters was not enough.

Our experiments presented in Section 7 show that for surfaces with a long systole, which is the most common case (see Section 7.1), our algorithm is successful. Using a double precision to set the rational coordinates of approximate circumcenters and to compute the bounds on $\cosh(\varepsilon) - 1$ actually constructs valid ε-nets. On the other hand, we also observe in the full version of this article [16, Sec 7.3] that higher precision would be needed to handle a surface with a very small systole.

Our future work will focus on certifying the largeness test for triangles and check the ε-packing property after each insertion. If the ε-packing check fails, this means that the inserted point was not a good enough approximation of the circumcenter and iteratively refining it (which is possible via the CGAL Algebraic Kernel [3]) would eventually recover the ε-packing property. Note that iterative refinement of the rational bounds on $\cosh(\varepsilon) - 1$ must also be computed (which is possible via the Boost library).

7 Experiments

All the experiments of Section 7.2 are performed on 180 random surfaces of genus 2 generated as explained in Section 5 by uniformly choosing, for the Euclidean distance, three points in the upper half of the Poincaré disk. As explained in Section 7.1, these surfaces are expected to have a long systole. Experiments show that 173 of these surfaces have a systole greater than 0.5, while the 7 others have a systole greater than 0.21. See the full version [16, Sec 7.2] for details. Section 7.3 displays and analyzes visualizations of the output.

The source code and complete data of the experiments are available on GitHub [3].

[3] `https://github.com/camille-lanuel/ESA_2025_implementation_epsilon_net`

7.1 Distribution of Input Surfaces

To experiment with an algorithm in practice, it is important to sample the space of possible input with some distribution. We discuss this issue below and explain why our random generator of surfaces is likely to produce surfaces with long systole.

Well-Distributed Surfaces?

The situation was nicely described by Mirzakhani [32] (though there are more recent results [33]). We just give a flavor here. The moduli space $\mathcal{M}_g$ is the set of all hyperbolic surfaces of genus g. It can be equipped by two natural metrics: the Teichmüller distance and the Weil-Petersson one. Roughly speaking, they measure the deformation between two hyperbolic metrics: the first one considers the supremum and the second the average. Ideally, we would like to obtain a uniform sampling of $\mathcal{M}_g$, for any of the two metrics. However, today's mathematical literature does not answer this question. The first problem is that there is no known parameterization of $\mathcal{M}_g$, so, we can only work with a parameterization of its universal cover, the Teichmüller space $\mathcal{T}_g$, i.e., the set of so-called marked hyperbolic surfaces. Indeed, in $\mathcal{T}_g$, as opposed to in $\mathcal{M}_g$, applying a non-identity homeomorphism to a hyperbolic surface gives a different element in $\mathcal{T}_g$: the moduli space $\mathcal{M}_g$ is the quotient of $\mathcal{T}_g$ by the mapping class group Mod_g, which is the group of homeomorphisms of the topological surface of genus g. Unfortunately, the known parameterizations of $\mathcal{T}_g$ are not invariant under the action of Mod_g, which is the main reason why sampling $\mathcal{M}_g$ is so intricate. It is not clear how to describe a fundamental domain of $\mathcal{M}_g$ in $\mathcal{T}_g$ in any parameterization of $\mathcal{T}_g$. As of today, the best that can be done is to sample a parameterization of $\mathcal{T}_g$, being aware that this does not lead to a good distribution on $\mathcal{M}_g$.

For completeness, we mention that there are other ways of constructing random surfaces in theory, by randomly gluing hyperbolic ideal triangles together [6, 34]. However, these methods rely on a compactification of the obtained cusped surfaces, which, roughly speaking, boils down to a conformal uniformization of the metric. This process is obviously very far from yielding a practical construction. Additionally, those approaches lead to surfaces with huge genus (typically, several thousands), which cannot be manipulated in practice.

Systole of random surfaces

Remark that $\mathcal{M}_g$ is not a compact set, as a surface can have an arbitrarily small systole: Indeed, in any compact subset of $\mathcal{M}_g$, the systole function $sys(\cdot)$ is bounded. Mirzakhani showed that small systoles are somewhat rare in $\mathcal{M}_g$: the probability to obtain a surface with a systole smaller than some $\sigma_0 > 0$ using a uniform distribution (for the Weil-Petersson point of view) is of the order of σ_0^2. This means that the typical case is a surface with a reasonably long systole. It is the case for the random surfaces that we generated, as explained in Section 5. For completeness, we also specifically design a surface with a small systole and experimentally observe the impact in the full version of this article [16, Sec 7.3].

7.2 Main Results

For each of the 180 random surfaces, an input for the ε-net algorithm is computed as a single vertex Delaunay triangulation of the surface, as explained in Section 5. The `epsilon_net` method is run with 50 values of ε on every input, decreasing from 0.5 to 0.01 with a step of 0.01. Table 1 presents an overview of the results.

We first observe that the number of vertices in the computed ε-nets is close to the theoretical upper bound for an ε-thick surface, which is $16(g-1)/\varepsilon^2 = 16/\varepsilon^2$ ($g = 2$ for all surfaces of the experiments). In proportion, the number of vertices is 54% of the upper bound on average, with a standard deviation of 2%. The minimum is 47% and the maximum is 63% over all tested surfaces and values of ε.

Table 1 Average number of vertices of the ε-nets.

ε	**0.50**	**0.40**	**0.30**	**0.20**	**0.10**	**0.05**	**0.01**
Avg. # of vertices	34	54	96	216	865	3,454	86,314
$\mathbf{16/\varepsilon^2}$	64	100	178	400	1,600	6,400	160,000

Even though, as in the Euclidean case [23], we have no complexity analysis of the visibility walk for the point location, we observe that the walk traverses a (small) constant number of triangles. The walk locates the approximate circumcenter of a triangle in $\mathbb{H}^2$, starting from this triangle. The average number of computed lifted triangles during each walk tends to decrease when ε becomes smaller, while the average proportion of approximate circumcenters located in their own triangle tends to increase, as shown in Figure 8. On average, 68% of the approximate circumcenters lie in their triangle. Over all surfaces and all tested values of ε, the farthest located points were 4 triangles away from the starting triangle of the walk.

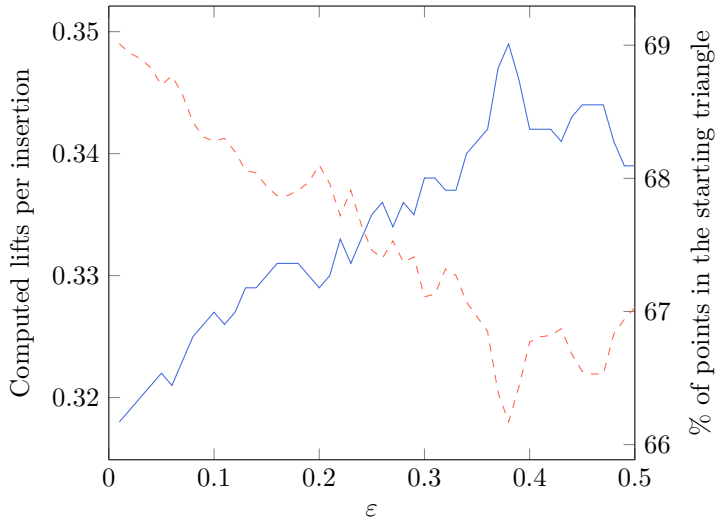


Figure 8 Average number of computed lifted triangles in the walk at each locate query (left, solid), and average percentage of points lying in the initial lifted triangle of the walk (right, dashed).

We counted the number of flips done to restore the Delaunay property of the triangulation after each split. On average, it decreases when ε becomes smaller, going from 3.41 for $\varepsilon = 0.5$ to 2.41 for $\varepsilon = 0.01$, as shown in Figure 9. This shows that the number of flips at each insertion is a small constant in practice. The total number of flips is thus, in practice, linear in the number of points, which is in contrast with the theoretical quadratic bound (see Section 2.3).

In the full version of this article [16, App C], we show an order of magnitude for the running times to give an idea of the practical complexity of our implementation. It appears to be slightly faster than quadratic in $1/\varepsilon$.

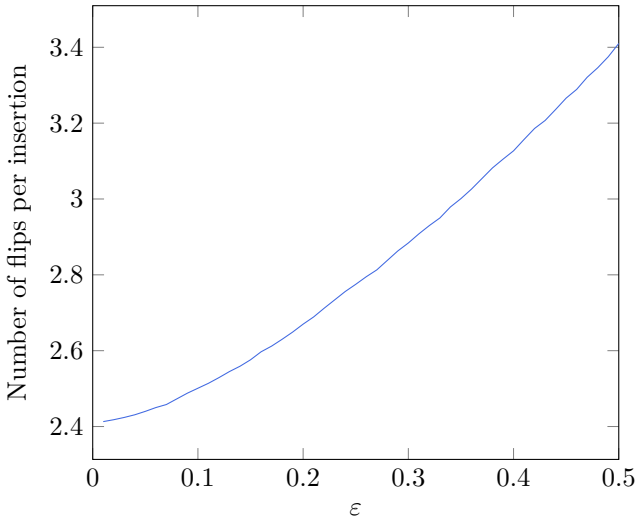


Figure 9 Average number of flips done to recover the Delaunay property after each insertion.

7.3 Visualization of the Output

To visualize an ε-net, we draw a fundamental domain of the surface in the Poincaré disk using its Delaunay triangulation. To do so, it suffices to draw a lift of each triangle in a connected way. We cannot just use the anchors since they would generally not lead to a connected domain. So, we access the anchor of a chosen triangle and build lifts of the other triangles starting from this initial anchor as explained in Section 4. As objects appear smaller near the boundary of the Poincaré disk, we center the drawing at the origin to be able to observe the details. To achieve this, we translate the vertices of the initial anchor such that one of them is 0 before computing the rest of the drawing. If we want similar drawings when running the ε-net algorithm with different values of ε on a given surface, we always use an anchor having a fixed lift $\widetilde{b}$ of b, the unique vertex of the input triangulation (Section 5), as the initial anchor; we then apply the translation that translates $\widetilde{b}$ to 0.

In Figures 10 and 11, the drawings are computed by the lift method of the CGAL package *2D triangulations on hyperbolic surfaces* [13], which uses a weight on edges to order the lifts of triangles of the triangulation T of S. The weight of an edge $(\widetilde{x}, \widetilde{y})$ is defined as $|\widetilde{x}|^2 + |\widetilde{y}|^2$ ($|\cdot|$ being the complex modulus). The drawing is then iteratively computed: given T' the set of triangles that have been lifted, the next triangle to be lifted is the one in $T \setminus T'$ that is incident to the edge of least weight in T'.

Figure 10 shows the obtained Delaunay triangulation of the surface with a small systole studied in the full version of this article [16, Sec 7.3]. Since the systole is very small, there is a very long collar around the shortest non-contractible closed curve.

The drawings of triangulations shown in Figure 11 naturally look like Dirichlet domains since the weights on the edges ensure that the lifts of triangles entirely contained in $\mathcal{D}_{\widetilde{b}}$, the Dirichlet domain of $\widetilde{b}$, are drawn. This becomes clear when $\widetilde{b}$ is translated to the origin (Figure 12). Since the input of the ε-net algorithm is a Delaunay triangulation with the single vertex b, we obtain $\mathcal{D}_{\widetilde{b}}$ by computing the circumcenter of all the lifted triangles incident to $\widetilde{b}$ in the input triangulation, before running the ε-net algorithm.

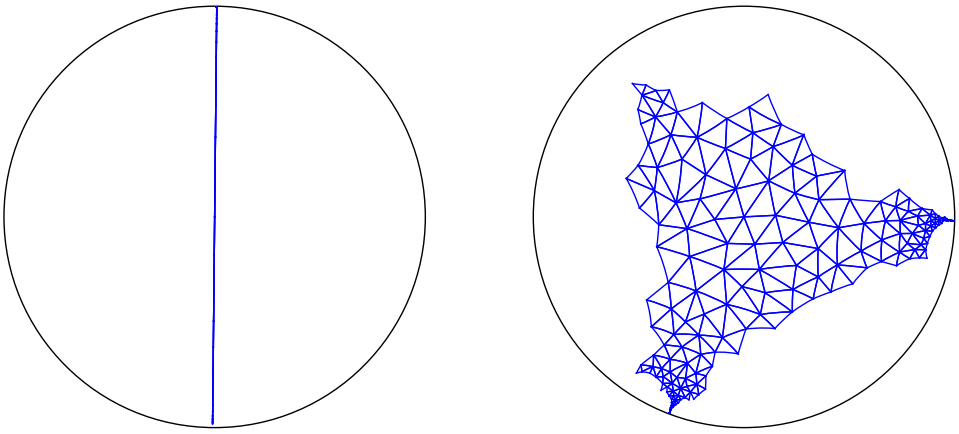


Figure 10 Delaunay triangulation of the computed 0.25-net of the surface with a small systole of [16, Sec 7.3]. Left: Drawing centered at a vertex in the long collar. Right: Drawing centered outside the collar, which appears in the form of "horns" pointing towards the boundary of the Poincaré disk.

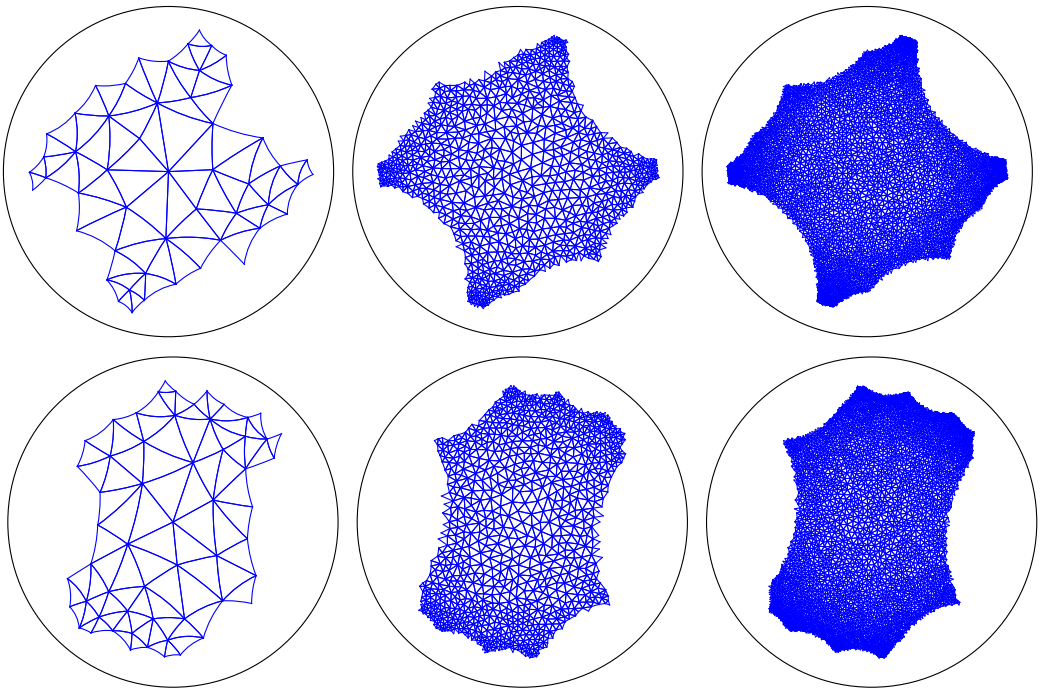


Figure 11 Delaunay triangulations of the computed ε-nets for ε=0.5 (left), ε=0.1 (middle), ε=0.05 (right) on the surfaces of seed 123 (top) and 321 (bottom).

An alternative drawing is obtained by ordering the lifts of the triangles around the initial anchor following a Breadth First Search (BFS) algorithm on the adjacency graph. Such a drawing represents a *combinatorial Dirichlet domain* (see Figure 13). We observe that, when ε decreases, the combinatorial Dirichlet domain fits the Dirichlet domain $\mathcal{D}_{\tilde{b}}$ better. Formalizing this convergence is an interesting open question.

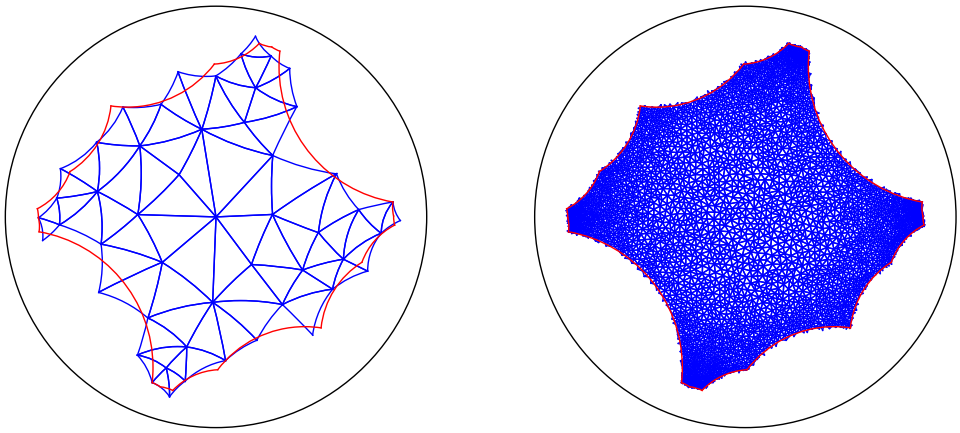


Figure 12 Delaunay triangulations of a 0.5-net (left) and a 0.05-net (right) of the surface of seed 123, and the corresponding Dirichlet domain. Lift computed with the CGAL package [13].

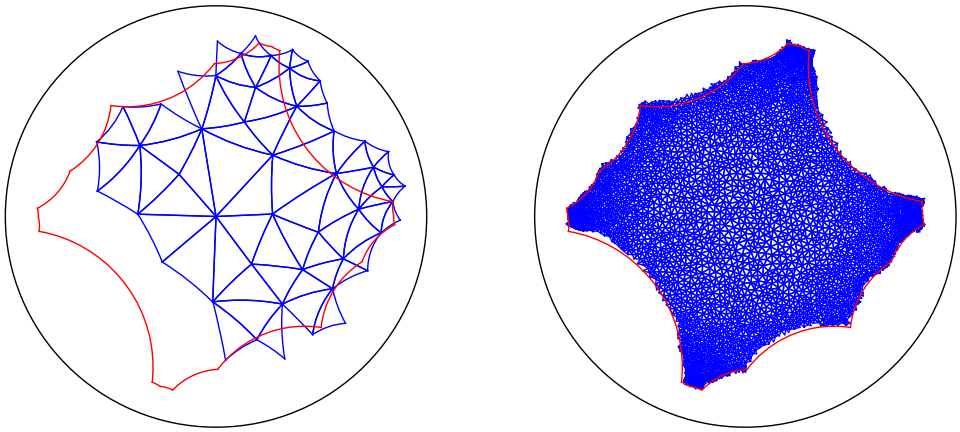


Figure 13 Delaunay triangulations of a 0.5-net (left) and a 0.05-net (right) of the surface of seed 123, and the corresponding Dirichlet domain. Lift computed with a BFS algorithm.

References

1. Aline Aigon-Dupuy, Peter Buser, Michel Cibils, Alfred F. Künzle, and Frank Steiner. Hyperbolic octagons and Teichmüller space in genus 2. *Journal of Mathematical Physics*, 46(3):033513, February 2005. doi:10.1063/1.1850177.
2. Alan F. Beardon. *The Geometry of Discrete Groups*. Graduate Texts in Mathematics. Springer New York, 1st edition, 1983. doi:10.1007/978-1-4612-1146-4.
3. Eric Berberich, Michael Hemmer, Michael Kerber, Sylvain Lazard, Luis Peñaranda, and Monique Teillaud. Algebraic kernel. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. URL: https://doc.cgal.org/6.0.1/Manual/packages.html#PkgAlgebraicKernelD.
4. Mikhail Bogdanov, Olivier Devillers, and Monique Teillaud. Hyperbolic Delaunay complexes and Voronoi diagrams made practical. *Journal of Computational Geometry*, 5(1):56–85, March 2014. doi:10.20382/jocg.v5i1a4.
5. A. Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24(2):162–166, February 1981. doi:10.1093/comjnl/24.2.162.
6. Robert Brooks and Eran Makover. Random construction of Riemann surfaces. *Journal of Differential Geometry*, 68(1):121–157, 2004.

7 Peter Buser. *Geometry and Spectra of Compact Riemann Surfaces*. Modern Birkhäuser Classics. Birkhäuser Boston, 1st edition, 2010. doi:10.1007/978-0-8176-4992-0.

8 Kenneth L. Clarkson. Building triangulations using ϵ-nets. In *38th annual ACM Symposium on Theory of Computing (STOC)*, pages 326–335. Association for Computing Machinery, May 2006. Extended abstract (long paper available at https://kenclarkson.org/enet_tris/p.pdf). doi:10.1145/1132516.1132564.

9 Éric Colin de Verdière. Algorithms for embedded graphs, 2021. Lecture notes. URL: https://monge.univ-mlv.fr/~colinde/cours/all-algo-embedded-graphs.pdf.

10 Guillaume Damiand. Combinatorial maps. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. URL: https://doc.cgal.org/6.0.1/Manual/packages.html#PkgCombinatorialMaps.

11 Leila de Floriani, Bianca Falcidieno, George Nagy, and Caterina Pienovi. On sorting triangles in a Delaunay tessellation. *Algorithmica*, 6:522–532, June 1991. doi:10.1007/BF01759057.

12 Vincent Despré, Loïc Dubois, Benedikt Kolbe, and Monique Teillaud. Experimental analysis of Delaunay flip algorithms on genus two hyperbolic surfaces, December 2022. URL: https://hal.inria.fr/hal-03462834.

13 Vincent Despré, Loïc Dubois, Marc Pouget, and Monique Teillaud. 2D triangulations on hyperbolic surfaces. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.1 edition, 2025. URL: https://doc.cgal.org/6.1/Manual/packages.html#PkgHyperbolicSurfaceTriangulation2.

14 Vincent Despré, Benedikt Kolbe, Hugo Parlier, and Monique Teillaud. Computing a Dirichlet domain for a hyperbolic surface. In *39th International Symposium on Computational Geometry (SoCG)*, volume 258, pages 27:1–27:15, 2023. doi:10.4230/LIPIcs.SoCG.2023.27.

15 Vincent Despré, Camille Lanuel, Marc Pouget, and Monique Teillaud. Implementation of the ϵ-net algorithm. Software, ANR Abysm, swhId: swh:1:dir:2fab64276f7d193b0b712c135fa9eebba62f0509 (visited on 2025-09-04). URL: https://github.com/camille-lanuel/ESA_2025_implementation_epsilon_net, doi:10.4230/artifacts.24672.

16 Vincent Despré, Camille Lanuel, Marc Pouget, and Monique Teillaud. ε-net algorithm implementation on hyperbolic surfaces. Preprint, December 2024. URL: https://hal.science/hal-04820353.

17 Vincent Despré, Camille Lanuel, and Monique Teillaud. Computing an ε-net of a closed hyperbolic surface. In *40th European Workshop on Computational Geometry (EuroCG'24)*, pages 22:1–22:8, 2024. URL: https://eurocg2024.math.uoi.gr/data/uploads/paper_22.pdf.

18 Vincent Despré, Jean-Marc Schlenker, and Monique Teillaud. Flipping geometric triangulations on hyperbolic surfaces. In *36th International Symposium on Computational Geometry (SoCG)*, volume 164, pages 35:1–35:16, June 2020. Final version to appear in JoCG https://jocg.org/. doi:10.4230/LIPIcs.SoCG.2020.35.

19 BOOST development team. *BOOST C++ Libraries*. URL: http://www.boost.org.

20 CORE development team. *The CORE library project*. URL: https://cs.nyu.edu/~exact/core_pages/.

21 GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. URL: https://gmplib.org.

22 Olivier Devillers and Ross Hemsley. The worst visibility walk in a random Delaunay triangulation is $O(\sqrt{n})$. *Journal of Computational Geometry*, 7(1):332–359, July 2016. doi:10.20382/jocg.v7i1a16.

23 Olivier Devillers, Sylvain Pion, and Monique Teillaud. Walking in a triangulation. *International Journal of Foundations of Computer Science*, 13:181–199, 2002. doi:10.1142/S0129054102001047.

24 Matthijs Ebbens, Iordan Iordanov, Monique Teillaud, and Gert Vegter. Delaunay triangulations of generalized Bolza surfaces. *Journal of Computational Geometry*, 13(1):125–177, 2022. doi:10.20382/jocg.v13i1a5.

25 Herbert Edelsbrunner. An acyclicity theorem for cell complexes in d dimensions. *Combinatorica*, 10(3):251–260, 1990. doi:10.1007/BF02122779.

26 Hershel M. Farkas and Irwin Kra. *Riemann Surfaces*, volume 71 of *Graduate Texts in Mathematics*. Springer, New York, NY, 1992. doi:10.1007/978-1-4612-2034-3.

27 Yoichi Imayoshi and Masahiko Taniguchi. *An Introduction to Teichmüller Spaces*. Springer Japan, 1992. doi:10.1007/978-4-431-68174-8.

28 Iordan Iordanov and Monique Teillaud. Implementing Delaunay triangulations of the Bolza surface. In *33rd International Symposium on Computational Geometry (SoCG)*, pages 44:1–44:15, July 2017. doi:10.4230/LIPIcs.SoCG.2017.44.

29 Iordan Iordanov and Monique Teillaud. 2D periodic hyperbolic triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.14 edition, 2019. URL: https://doc.cgal.org/6.0.1/Manual/packages.html#PkgPeriodic4HyperbolicTriangulation2.

30 Charles L. Lawson. Software for C^1 surface interpolation. In *Symposium on Mathematical Software*, 1977. NASA Technical Report JPL-PUBL-77-30. URL: https://ntrs.nasa.gov/citations/19770025881.

31 Pascal Lienhardt. Subdivisions of n-dimensional spaces and n-dimensional generalized maps. In *Proceedings 5th Annual Symposium on Computational Geometry*, pages 228–236, 1989. doi:10.1145/73833.73859.

32 Maryam Mirzakhani. Growth of Weil–Petersson volumes and random hyperbolic surface of large genus. *Journal of Differential Geometry*, 94(2):267–300, 2013.

33 Laura Monk. Benjamini–Schramm convergence and spectra of random hyperbolic surfaces of high genus. *Analysis & PDE*, 15(3):727–752, 2022.

34 Bram Petri. Random regular graphs and the systole of a random surface. *Journal of Topology*, 10(1):211–267, 2017.

35 Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1):21–74, May 2002. doi:10.1016/S0925-7721(01)00047-5.

36 Heiner Zieschang, Elmar Vogt, and Hans-Dieter Coldewey. *Surfaces and Planar Discontinuous Groups*, volume 835 of *Lecture Notes in Mathematics*. Springer Berlin Heidelberg, 1980. doi:10.1007/bfb0089692.

Polynomial-Time Constant-Approximation for Fair Sum-Of-Radii Clustering

Sina Bagheri Nezhad
Portland State University, OR, USA

Sayan Bandyapadhyay
Portland State University, OR, USA

Tianzhi Chen
Portland State University, OR, USA

Abstract

In a seminal work, Chierichetti et al. [20] introduced the (t, k)-fair clustering problem: Given a set of red points and a set of blue points in a metric space, a clustering is called fair if the number of red points in each cluster is at most t times and at least $1/t$ times the number of blue points in that cluster. The goal is to compute a fair clustering with at most k clusters that optimizes certain objective function. Considering this problem, they designed a polynomial-time $O(1)$- and $O(t)$-approximation for the k-center and the k-median objective, respectively. Recently, Carta et al. [15] studied this problem with the sum-of-radii objective and obtained a $(6+\epsilon)$-approximation with running time $O((k \log_{1+\epsilon}(k/\epsilon))^k n^{O(1)})$, i.e., fixed-parameter tractable in k. Here n is the input size. In this work, we design the first polynomial-time $O(1)$-approximation for (t, k)-fair clustering with the sum-of-radii objective, improving the result of Carta et al. Our result places sum-of-radii in the same group of objectives as k-center, that admit polynomial-time $O(1)$-approximations. This result also implies a polynomial-time $O(1)$-approximation for the Euclidean version of the problem, for which an $f(k) \cdot n^{O(1)}$-time $(1+\epsilon)$-approximation was known due to Drexler et al. [24]. Here f is an exponential function of k. We are also able to extend our result to any arbitrary $\ell \geq 2$ number of colors when $t = 1$. This matches known results for the k-center and k-median objectives in this case. The significant disparity of sum-of-radii compared to k-center and k-median presents several complex challenges, all of which we successfully overcome in our work. Our main contribution is a novel cluster-merging-based analysis technique for sum-of-radii that helps us achieve the constant-approximation bounds.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases fair clustering, sum-of-radii clustering, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.62

Related Version *Full Version*: `https://arxiv.org/abs/2504.14683` [44]

Funding This work was supported by the National Science Foundation under Grant No. AF 2311397.

1 Introduction

Given a set of points P in a metric space (Ω, d) and an integer $k > 0$, the task of clustering is to find a partition $X_1, \ldots, X_k$ of P into k groups or clusters such that each group has similar points. The similarity of the clusters is typically modeled using an objective function which is to be minimized. In this work, we focus on the *sum-of-radii* objective, which is defined as the sum of the radii of k balls that contain the points of the respective k clusters. The sum-of-radii objective, while also center-based, has a different flavor from objectives such as k-center, k-median, and k-means, as it directly sums the radii of the clusters rather than measuring distances from each point to its assigned center. In these objectives, k representative points (or cluster centers) are chosen, and the corresponding clusters are


© Sina Bagheri Nezhad, Sayan Bandyapadhyay, and Tianzhi Chen;
licensed under Creative Commons License CC-BY 4.0

formed by assigning the points of P to their nearest centers. Such a partition is popularly known as the *Voronoi* partition. It is not hard to see that an optimal sum-of-radii clustering is not necessarily a Voronoi partition. The study of sum-of-radii was motivated by the idea that it could reduce the so-called *dissection effect* that is observed in k-center type objectives (see attached full version for details).

Sum-of-radii clustering is known to be NP-hard even in planar metrics and metrics of constant doubling dimension [30]. Consequently, it has received substantial attention from the approximation algorithms community. Charikar and Panigrahy [16] designed a Primal-Dual and Lagrangian-relaxation-based 3.504-approximation algorithm that runs in polynomial time (poly-time). Recently, using similar techniques, Friggstad and Jamshidian [26] improved the approximation factor to 3.389. The best-known approximation factor for sum-of-radii in polynomial time is $3 + \epsilon$ for any $\epsilon > 0$, due to Buchem et al. [14]. In stark contrast to other well-studied center-based objectives such as k-center and k-median, the sum-of-radii objective admits a QPTAS [30], which is based on a randomized metric partitioning scheme. Additionally, the problem can be solved exactly in polynomial time in the Euclidean metric of constant dimension [31]. The problem also admits polynomial time exact algorithms in other restricted settings, such as when singleton clusters are not allowed [9] and the metric is unweighted [33].

In recent years, sum-of-radii clustering has also been studied with additional constraints. One such popular constraint is the capacity constraint, which puts restriction on the number of points that each cluster can contain. In a series of articles [35, 8, 36, 25], $O(1)$-approximation algorithms have been designed for capacitated sum-of-radii with running time fixed-parameter tractable (FPT) in k (i.e., $f(k) \cdot n^{O(1)}$ for a function f of k), culminating in an approximation factor of 3. Inamdar and Varadarajan [35] studied sum-of-radii with a matroid constraint where the set of centers of the balls must be an independent set of a matroid. They obtain an FPT 9-approximation for this problem. The approximation factor has recently been improved to 3 by Chen et al. [18]. Obtaining a poly-time $O(1)$-approximation for any of these constrained versions is an interesting open question. However, poly-time $O(1)$-approximations are known for sum-of-radii with lower bounds and with outliers [3, 14].

Sum-of-radii has also been studied with fairness constraints, which is the main focus of our work. Clustering with fairness constraints or fair clustering stems from the idea that protected groups (defined based on a sensitive feature, e.g., gender) must be well-represented in each cluster. In recent years, fair clustering has received significant attention from researchers across several areas of computer science. In a seminal work, Chierichetti et al. [20] introduced the (t, k)-*fair clustering* problem. In this problem, we are given a set P_1 of red points, a set P_2 of blue points, that together contain n points, and an integer balance parameter $t \geq 1$. A clustering is called (t, k)-*fair* if, for any cluster X, the number of red points in X is at least $1/t$ times and at most t times the number of blue points in X. We say that each cluster in a (t, k)-fair clustering is t-*balanced*.

Chierichetti et al. studied (t, k)-fair clustering with k-center and k-median objectives, and obtained poly-time 4- and $O(t)$-approximation, respectively. Since then obtaining a poly-time $O(1)$-approximation for (t, k)-fair median or means remained an intriguing open question. The main challenge in this case is that the optimal clusterings are no longer Voronoi partitions, as they also need to be (t, k)-fair. Subsequently, (t, k)-fair median/means has been studied in a plethora of works. The only setting where it is known to obtain a poly-time $O(1)$-approximation is when $t = 1$ [12], that is for $(1, k)$-fair median/means. These problems have also been considered in the Euclidean case [46, 5].

The (t, k)-fair median/means problem has also been studied with an arbitrary ℓ number of groups. The algorithm of Böhm et al. [12] for $t = 1$ also yields a poly-time $O(1)$-approximation in this case. Note that for $t = 1$, a cluster contains the same number of points from all groups. Bandyapadhyay et al. [6] obtained a poly-time approximation for (t, k)-fair median with a factor that depends on t, ℓ, and k. Bercea et al. [11] and Bera et al. [10] independently defined a generalization of (t, k)-fair clustering. There we are given balance parameters $\alpha_i, \beta_i \in [0, 1]$ for each group $1 \leq i \leq \ell$. A clustering is called *fair representational* if the fraction of points from group i in every cluster is at least α_i and at most β_i for all $1 \leq i \leq \ell$. They show that it is possible to obtain poly-time bi-criteria type $O(1)$-approximations where we are allowed to violate the fairness constraints by an additive small constant factor. Subsequently, Dai et al. [23] designed a DP-based poly-time $O(\log k)$-approximation for this problem. For ℓ groups, their running time is $n^{O(l)}$.

Carta et al. [15] studied fair versions of sum-of-radii. In particular, they study a more general class of *mergeable* constraints. A clustering constraint is called mergeable if the union of two clusters satisfying the constraint also satisfies the constraint. They show that the fairness constraints defined in (t, k)-fair clustering and fair representational clustering are mergeable. In their work, they obtained a $(6 + \epsilon)$-approximation for sum-of-radii with mergeable constraints. In particular, for the above two fairness constraints, their run time is $O((k \log_{1+\epsilon}(k/\epsilon))^k n^{O(1)})$, so FPT in k. The algorithm iteratively guesses the next cluster based on a *k-center completion problem* leading to the FPT run time. Their approximation factor improves to $3+\epsilon$ when $t = 1$. Drexler et al. [24] obtained an FPT $(1+\epsilon)$-approximation for Euclidean sum-of-radii with mergeable constraints. Chen et al. [18] studied a fair version, which is a special case of matroid sum-of-radii, and hence obtained an FPT 3-approximation. A summary of the results on fair clustering under various objectives is provided in Table 1.

As mentioned before, for fair representational models, only bi-criteria type $O(1)$-approximations are known for k-center/median/means, even with two groups. As we focus on our theoretical quest of designing poly-time $O(1)$-approximations fully satisfying the fairness constraints, we study (t, k)-fair sum-of-radii. In light of the above discussion, we state the following two questions.

Question 1: Does (t, k)-fair sum-of-radii (with two groups) admit a poly-time constant-approximation algorithm?

Question 2: Does $(1, k)$-fair sum-of-radii with an arbitrary $\ell \geq 2$ number of groups admit a poly-time constant-approximation algorithm?

After the work of Chierichetti et al. [20], several other notions of fairness have been considered in the context of clustering problems. The following is a sample of these works grouped by the fairness notions: individual fairness [38, 43, 47, 13, 1], proportional fairness [19, 42], fair center representation [17, 40, 39, 21, 34], colorful [7, 37, 4], and min-max fairness [2, 28, 41, 22, 29, 32].

1.1 Our Contributions and Techniques

In our work, we prove two theorems resolving Questions 1 and 2 in the affirmative. First, we prove the following theorem.

Table 1 Summary of approximation results for fair clustering under various objectives. "Poly" denotes polynomial time; "FPT" denotes fixed-parameter tractable in k; ℓ is the number of groups.

Objective	Fairness Type	Approximation	Time	Reference
k-Center	(t,k) (2 groups)	4	Poly	[20]
k-Median	(t,k) (2 groups)	$O(t)$	Poly	[20]
k-Median	$(1,k)$ (ℓ groups)	$O(1)$	Poly	[12]
k-Median	(t,k) (ℓ groups)	$f(t,\ell,k)$	Poly	[6]
k-Median / Center	Representational (bi-criteria)	$O(1)$	Poly	[10, 11]
Sum-of-Radii	Unconstrained	$3+\varepsilon$	Poly	[14]
Sum-of-Radii	Capacitated	3	FPT	[25]
Sum-of-Radii	Matroid constraint	3	FPT	[18]
Sum-of-Radii	(t,k) (2 groups)	$6+\varepsilon$	FPT	[15]
Sum-of-Radii	$(1,k)$ (ℓ groups)	$3+\varepsilon$	FPT	[15]
Sum-of-Radii	(t,k) (2 groups)	$\mathbf{144}+\varepsilon$	**Poly**	**This work**
Sum-of-Radii	$(1,k)$ (ℓ groups)	$\mathbf{180}+\varepsilon$	**Poly**	**This work**

▶ **Theorem 1.** *There is a polynomial-time* $(144+\epsilon)$*-approximation algorithm for* (t,k)*-fair sum-of-radii (with two groups).*

Our result complements the FPT approximation result of Carta et al. [15] by achieving the first $O(1)$-approximation for the problem in polynomial time. The result also implies a poly-time $O(1)$-approximation for Euclidean (t,k)-fair sum-of-radii, for which only an FPT $(1+\epsilon)$-approximation was known [24]. We note that our result should also be compared with that of (t,k)-fair k-median for which only $O(t)$-approximation is known in polynomial time. In particular, our result places sum-of-radii in the same group of objectives as k-center that admits polynomial-time $O(1)$-approximations. Moreover, our result shows that (t,k)-fair sum-of-radii is in contrast to most of the constrained versions of sum-of-radii, including capacitated clustering, for which only FPT $O(1)$-approximations are known.

Next, we give an overview of our approach. Our approximation algorithm is motivated by the algorithms for (t,k)-fair center and (t,k)-fair median [20]. These algorithms have two major steps. In the first step, a *fairlet decomposition* of the points in $X = P_1 \cup P_2$ is computed, i.e., a partition $\mathcal{Y} = \{Y_1, \ldots, Y_m\}$ such that for each fairlet Y_i, it either has 1 red point and at most t blue points or 1 blue point and at most t red points. Let $\beta : P_1 \cup P_2 \to [m]$ be the function that maps each point x to the index of the fairlet that contains x. From each Y_i, an arbitrary point y_i is designated as its *representative*. In the second step, a clustering of these m representatives is computed with the respective cost function. Also, for each Y_i, all of its points are assigned to the cluster that contains y_i. The new clustering is obviously (t,k)-fair, as each cluster is a merger of fairlets. For the analysis of the cost of the computed clustering, they define a fairlet decomposition cost, which is used to bound the assignment cost of the points in the second step. For k-center, this cost is $\max_{x\in X} d(x, y_{\beta(x)})$, and for k-median, it is $\sum_{x\in X} d(x, y_{\beta(x)})$. Indeed, both of these costs when optimal are comparable to the optimal (t,k)-fair clustering cost. For k-center, it is within a constant factor, and for k-median it is within an $O(t)$ factor. Then, it is sufficient to compute a fairlet decomposition in the first step whose cost is within a small constant-factor of the optimal fairlet decomposition cost.

Coming back to (t,k)-fair sum-of-radii, it is not clear how to define a suitable fairlet decomposition cost that can be compared to the optimal (t,k)-fair sum-of-radii cost. In particular, such a cost needs to be defined independent of the number of clusters k. However, for sum-of-radii, the objective is the sum of radii of k clusters. For example, a natural

candidate, the cost for k-median, i.e., $\sum_{x \in X} d(x, y_{\beta(x)})$, is likely to be much larger than the optimal sum-of-radii cost. In the absence of such a suitable fairlet decomposition cost, it is difficult to argue the increase in the assignment cost, when actual points of Y_i are assigned instead of just the representative y_i.

Our approach. Our algorithm is surprisingly simple to state. We first compute a complete bi-partite graph G with P_1 and P_2 being the two parts. The weight of each edge is set to be the distance between the two corresponding endpoints. Subsequently, a degree-constrained, spanning subgraph of this graph is computed where each vertex has a degree in range $[1, t]$, and the sum of the weights of the edges is minimized. Such an optimal subgraph can be computed in polynomial time using the algorithm of Gabow [27]. Moreover, one can show that such a subgraph is a collection of stars each having at most t edges. Thus, our algorithm up to this point is in a similar spirit to that of k-median. As we argued before, the total weight of such a subgraph can be very large compared to the optimal sum-of-radii cost. Our main contribution is to prove that there is a sum-of-radii clustering of the stars (or representatives of them) computed in this way whose cost is at most a constant times the optimal (t, k)-fair sum-of-radii cost. Then, one can compute an approximate sum-of-radii clustering of these stars and return the corresponding clustering of the points in $P_1 \cup P_2$. The obtained clustering is (t, k)-fair, as the clusters are disjoint union of the vertices of stars, each having at most t edges. The proof of the existence of a clustering of the computed stars whose cost is nicely bounded is based on a novel analysis technique that merges a set of optimal clusters to obtain *superclusters*. We give an overview in the following.

Let H be the degree-constrained subgraph computed with the minimum weight possible. Also, let $\mathcal{C}^* = \{C_1^*, C_2^*, \ldots, C_k^*\}$ be a fixed optimal (t, k)-fair sum-of-radii clustering. We repetitively merge pairs of these clusters if there are edges in H across them. Let $\hat{\mathcal{C}} = \{\hat{C}_1, \hat{C}_2, \ldots, \hat{C}_\kappa\}$ be the resulting clustering. By our construction, each star of H is fully contained in one of these merged clusters or *superclusters*. Thus, it is sufficient to show that the radius of each supercluster $\hat{C}_i$ is at most $O(1)$ times the sum of the radii of the *associated* optimal clusters whose merger is $\hat{C}_i$. To bound such radius, we introduce a notion of *minimum-switch* paths between pairs of clusters. These paths play a central role in our analysis. We prove that it is possible to bound the (weighted) length of any such path by $O(1)$ times the sum of the radii of the associated optimal clusters. Then the diameter (or radius) of the supercluster can also be bounded likewise, as any two cluster vertices are connected by a minimum-switch path. The important distinction is that the length of any arbitrary path might not be bounded in such a nice way.

Next, we prove the following theorem concerning Question 2.

▶ **Theorem 2.** *There is a polynomial-time* $(180 + \epsilon)$*-approximation algorithm for* $(1, k)$*-fair sum-of-radii with* $\ell \geq 2$ *groups of points.*

Again our result directly improves the FPT approximation result of Carta et al. [15] and extends to more than 2 groups. The result matches the known constant-approximation bound for k-center/median/means in this case. The proof of the above theorem is similar to the proof of Theorem 1, and so employs the same supercluster-based analysis framework. However, here we need to handle ℓ colors. The main challenge boils down to bounding the diameter of a certain multi-partite graph G_1^* with $\cup_{i=1}^{\ell} P_i$ being the set of vertices. Intuitively, by the analysis for two groups, the diameter of the graphs induced by only $P_1 \cup P_i$ is nicely bounded. However, we still need to bound the diameter of G_1^*. Consequently, we introduce an additional notion of *minimum-color-switch* paths. We prove that the lengths of these paths can also be bounded nicely, exploiting their special properties.

Organization. We introduce notation in Section 2 and present our algorithms for (t, k)-fair and $(1, k)$-fair sum-of-radii in Sections 3 and 4, respectively. Proofs of statements marked by $(*)$ are available in the full version[1].

2 Preliminaries

In *sum-of-radii clustering*, we are given a set P of n points in a metric space with distance d and an integer $k > 0$. We would like to find: (i) a subset C of P containing k points and a non-negative integer r_q (called the radius) for each $q \in C$, and (ii) a function ϕ assigning each point $p \in P$ to a center $q \in C$ such that $d(p, q) \leq r_q$. The subset $X_q = \phi^{-1}(q)$ for each $q \in C$ is called the *cluster* corresponding to q having *radius* r_q. The goal is to find a clustering $\{X_q \mid q \in C\}$ that minimizes the sum of the radii $\sum_{q \in C} r_q$.

In *(t, k)-fair sum-of-radii clustering*, we are given two disjoint groups P_1 (red) and P_2 (blue) having n points in total in a metric space $(\Omega = P_1 \cup P_2, d)$ and an integer balance parameter $t \geq 1$. A clustering is called *(t, k)-fair* if, for each cluster X, the number of points from P_1 in X is at least $1/t$ times the number of points from P_2 in X and at most t times the number of points from P_2 in X. The goal is to compute a (t, k)-fair clustering minimizing the sum of the radii of the clusters. Each cluster in a (t, k)-fair clustering is called *t-balanced*.

In *Balanced sum-of-radii clustering*, we are given $\ell \geq 2$ disjoint groups $P_1, P_2, \ldots, P_\ell$ having n points in total in a metric space $(\Omega = \cup_{i=1}^{\ell} P_i, d)$ such that $|P_1| = |P_2| = \ldots = |P_\ell|$. A clustering is called *balanced* if, for each cluster X, it holds that $|X \cap P_1| = |X \cap P_2| = \ldots = |X \cap P_\ell|$. The goal is to compute a balanced clustering that minimizes the sum of the radii of the clusters. We say that each cluster in a balanced clustering is *1-balanced*.

Consider any metric space (Ω_1, d_1) and a subset $S_1 \subseteq \Omega_1$. For any cluster Q and a point p, $d_1(p, Q) = \max_{q \in Q} d_1(p, q)$. The center of Q in S_1 is the point, $\arg\min_{p \in S_1} d_1(p, Q)$. The radius of Q w.r.t. S_1 and d_1, denoted by $r_{(S_1, d_1)}(Q)$, is the distance between Q and its center in S_1, i.e., $r_{(S_1, d_1)}(Q) = \min_{p \in S_1} d_1(p, Q)$. We refer to the sum of the radii, w.r.t. S_1 and d_1, of the clusters in any clustering $\mathcal{C}$ as the cost of $\mathcal{C}$ w.r.t. S_1 and d_1 and denote it by $\text{cost}_{(S_1, d_1)}(\mathcal{C})$.

We note that the term "(t, k)-fairness" refers specifically to the two-color case, where each cluster must maintain a red-to-blue ratio within $[1/t, t]$. This notion does not naturally extend to more than two colors. In contrast, in the multi-color setting with $\ell \geq 2$ groups, we adopt the term "balanced clustering" (or "$(1, k)$-fair clustering with ℓ groups") to describe the setting where each cluster must contain an equal number of points from each group. While we use similar notation for consistency, these two notions are structurally different and should be interpreted accordingly.

3 The Algorithm for (t, k)-Fair Sum-of-Radii Clustering

In this section, we prove Theorem 1. To set up the stage, we define the following problem.

Min-cost Degree Constrained Subgraph (Min-cost DCS). A *Degree Constrained Subgraph* (DCS) $H = (V, E')$ of a graph $G = (V, E)$ is a subgraph such that the degree of each vertex v in H is in the range $[l(v), u(v)]$ for given integers $l(v)$ and $u(v)$. Suppose we are also given a weight function $w : E \to \mathbb{R}^+ \cup \{0\}$. A min-cost DCS $H = (V, E')$ of G is a DCS that minimizes the sum of the weights of the edges in E' over all DCS.

[1] https://arxiv.org/pdf/2504.14683

▶ **Proposition 1** ([27]). Min-cost DCS *can be solved in* $O(|V|^4)$ *time.*

The proposition follows from the work of Gabow (Theorem 5.2) [27]. There the stated time complexity is $O((\sum_{i\in V} u_i)\min\{|E|\log|V|, |V|^2\})$, which is $O(|V|^4)$, as each upper-bound u_i can be assumed to be at most the degree of the i-th vertex. One technicality is that they study the maximization version (with real weights), but the minimization version can be solved by the standard method of negating edge-weights in min-cost DCS. Also see [45] that has similar discussions and an $O(|V|^6)$ time algorithm for min-cost DCS, which they call *minimum-cost many-to-many matching with demands and capacities.*

▶ **Observation 1** (∗). *A min-cost DCS with* $l(v) = 1$ *for all* $v \in V$ *does not contain a path of length three, and thus it is a disjoint union of star graphs.*

Our algorithm is as follows.

The Algorithm

1. Construct a graph $G = (V, E)$ where $V = P_1 \cup P_2$ and $E = \{\{p, q\} \mid p \in P_1, q \in P_2\}$. Define the weight function w such that for each edge $e = \{p, q\}$, $w(e) = d(p, q)$. Compute a min-cost DCS $H = (V, E')$ of G with $l(v) = 1$ and $u(v) = t$ for all $v \in V$.
2. Construct an edge-weighted graph G' in the following way: For each $p \in \Omega$, add a vertex to G'; For each star S in H, add a vertex corresponding to S to G', which we also call by S; For each $p, q \in \Omega$, add the edge $\{p, q\}$ to G' with weight $d(p, q)$; For all $p \in \Omega$ and S in H, add the edge $\{p, S\}$ to G' with weight $\max_{q\in S} d(p, q)$. Let d' be the shortest path metric in G'. Construct the metric space (Ω', d') where Ω' is the subset of vertices in G' corresponding to the stars in H.
3. Compute a sum of radii clustering $X = \{X_1, \ldots, X_k\}$ of the points in Ω' using the Algorithm of Buchem et al. [14] (with Ω' also being the candidate set of centers).
4. Compute a clustering X' of the points in $P_1 \cup P_2$ using X in the following way. For each cluster $X_i \in X$, add the cluster $\cup_{p\in S|S\in X_i}\{p\}$ to X'. Return X'.

Next, we analyze the algorithm. First, we have the following observations.

▶ **Observation 2** (∗). X' *is a* (t, k)*-fair clustering of* $P_1 \cup P_2$.

Next, we analyze the approximation factor. Let $\mathcal{C}^* = \{C_1^*, C_2^*, \ldots, C_k^*\}$ be a fixed optimal (t, k)-fair clustering. We will prove the following lemma. Our result follows as a corollary.

▶ **Lemma 1.** *Consider the clustering* X *of* Ω' *constructed in Step 3 of the algorithm. Then* $cost_{(\Omega',d')}(X) \le (\mathbf{48} + \epsilon) \cdot \sum_{i=1}^{k} r_{(\Omega,d)}(C_i^*)$.

▶ **Corollary 1** (∗). *Consider the clustering* X' *of* $P_1 \cup P_2$ *constructed in Step 4 of the algorithm. Then* $cost_{(\Omega,d)}(X') \le (\mathbf{144} + \epsilon) \cdot \sum_{i=1}^{k} r_{(\Omega,d)}(C_i^*)$. *Thus, our algorithm is a* $(\mathbf{144} + \epsilon)$*-approximation algorithm.*

3.1 Proof of Lemma 1

In the following, we are going to prove Lemma 1. Consider the min-cost DCS $H = (V, E')$ computed in Step 1. Also, consider the optimal clusters in $\mathcal{C}^*$. We construct a new clustering $\hat{\mathcal{C}} = \{\hat{C}_1, \hat{C}_2, \ldots, \hat{C}_\kappa\}$ by merging clusters in $\mathcal{C}^*$ in the following way, where $1 \le \kappa \le k$. Initially, we set $\hat{\mathcal{C}}$ to $\mathcal{C}^*$. For each edge $\{p, q\}$ of E' such that $p \in \hat{C}_i, q \in \hat{C}_j$ and $i \ne j$, replace $\hat{C}_i, \hat{C}_j$ in $\hat{\mathcal{C}}$ by their union and denote it by $\hat{C}_i$ as well.

When the above merging procedure ends, by renaming the indexes, let $\hat{\mathcal{C}}=\{\hat{C}_1,\hat{C}_2,\ldots,\hat{C}_\kappa\}$ be the new clustering. Then, we have the following observation.

▶ **Observation 3.** *Consider any star S in H. Then, for some $1\leq i\leq\kappa$, all the points of S are contained in $\hat{C}_i$.*

Consider the clustering $\mathcal{C}'=\{C'_1,\ldots,C'_\kappa\}$ of Ω' defined in the following way. For each star S in H, identify the cluster $\hat{C}_i$ in $\hat{\mathcal{C}}$ that contains all the points in S. By Observation 3, such an index i exists. Assign the point p in Ω' corresponding to S to C'_i.

▶ **Lemma 2** (∗). $cost_{(\Omega',d')}(\mathcal{C}')\leq 2\cdot\ cost_{(\Omega,d)}(\hat{\mathcal{C}})$.

We will prove the following lemma.

▶ **Lemma 3.** $cost_{(\Omega,d)}(\hat{\mathcal{C}})\leq 8\cdot\sum_{i=1}^{k} r_{(\Omega,d)}(C^*_i)$.

Lemma 1 follows by Lemma 2 and 3 noting that the Algorithm of Buchem et al. [14] yields a $(3+\epsilon)$-factor approximation to the optimal clustering (along with an appropriate scaling of ϵ). In the rest of this section, we prove Lemma 3.

3.2 Proof of Lemma 3

For simplicity of notation, we drop (Ω,d) from $r_{(\Omega,d)}(.)$, as henceforth centers are always assumed to be in Ω and the metric to be d. Let us consider any fixed $\hat{C}_i$, and suppose it is constructed by merging the clusters $C^*_{i_1},C^*_{i_2},\ldots,C^*_{i_\tau}$. It is sufficient to prove that $r(\hat{C}_i)\leq 8\cdot\sum_{j=1}^{\tau} r(C^*_{i_j})$. For simplicity of notation, we rename $\hat{C}_i$ by $\hat{C}$, and $C^*_{i_1},C^*_{i_2},\ldots,C^*_{i_\tau}$ by $C^*_1,C^*_2,\ldots,C^*_\tau$.

Let $H_1=(V_1,E_1)$ be the induced subgraph of H such that the vertices of V_1 are in $\hat{C}$. We refer to a point of P_1 (resp. P_2) as a red (resp. blue) point. Note that the edges of H are across red and blue points. In the following, we construct an edge-weighted, directed multi-graph $G^*=(V^*,E^*)$ in the following manner. G^* has a vertex v_j corresponding to each cluster C^*_j, where $1\leq j\leq\tau$. There is an edge $e=(v_i,v_j)$ from v_i to v_j for each $p\in P_1\cap C^*_i$ and $q\in P_2\cap C^*_j$ such that $\{p,q\}$ is in E_1. We refer to such an edge as a 0-edge, i.e., its parity is 0. The weight ω_e of the edge e is $d(p,q)$. Similarly, there is a 1-edge (or parity 1 edge) $e=(v_i,v_j)$ from v_i to v_j for each $p\in P_2\cap C^*_i$ and $q\in P_1\cap C^*_j$ such that $\{p,q\}$ is in E_1. The weight ω_e of the edge e is $d(p,q)$. For each edge $e_i\in E^*$, we denote the corresponding edge in E_1 by $\{r_i,b_i\}$, where r_i is the red point and b_i is the blue point. For simplicity of exposition, we are going to make heavy use of this correspondence.

▶ **Observation 4** (∗). *Suppose there is a 0-edge (resp. 1-edge) (v_i,v_j) in E^*. Then there is also a 1-edge (resp. 0-edge) (v_j,v_i) in E^*.*

A directed path (or simply a path) $\pi=\{u_1,\ldots,u_l\}$ from u_1 to u_l in G^* is a sequence of distinct vertices such that (u_i,u_{i+1}) is in G^* for all $1\leq i\leq l-1$. We say that π contains the edges (u_i,u_{i+1}). If π contains all 0-edges (resp. 1-edges), it is called a 0-path (resp. 1-path). Two consecutive edges $e_1=(u_i,u_{i+1}),e_2=(u_{i+1},u_{i+2})$ on π are said to form a *switch* if they have different parity. We say that the switch happens at u_{i+1} and it is the corresponding switching vertex. The switch is called a b-switch if the parity of e_1 is b for $b\in\{0,1\}$. A directed cycle is formed from π by adding the edge (u_l,u_1) (if any) with it. The *reverse* path of π is the path $\{u_l,\ldots,u_1\}$ that contains the edges (u_{i+1},u_i) for all $1\leq i\leq l-1$. Such edges exist according to Observation 4. A 0-path (resp. 1-path) in a subgraph of G^* starting at v_i and ending at v_j is called *maximal* if v_j does not have any outgoing 0-edges (resp. 1-edges) in the subgraph.

▶ **Observation 5** (∗). *For any two vertices $v_i, v_j \in V^*$, there is a directed path from v_i to v_j in G^*.*

Consider any two vertices v_α and v_β of G^*. Let $\pi^* = \{v_\alpha = u_1, \ldots, u_l = v_\beta\}$ be a directed path from v_α to v_β having the minimum number of switches, i.e., a *minimum-switch path* from v_α to v_β.

We prove the following lemma.

▶ **Lemma 4.** $\sum_{e\in\pi^*} \omega_e \leq 6\cdot\sum_{i=1}^{\tau} r(C_i^*)$. *Moreover, if π^* does not have a switch,* $\sum_{e\in\pi^*} \omega_e \leq 4 \cdot \sum_{i=1}^{\tau} r(C_i^*)$.

Before proving this lemma, we show how to prove Lemma 3. Consider any point p in $\hat{C}_j$. Let $p \in C_g^*$. Now, consider any point q in $\hat{C}_j$ that is the farthest point from p. Let $q \in C_h^*$. By Lemma 4 it follows that, there is a path, say π', from v_g to v_h whose sum of the edge weights is at most $6 \cdot \sum_{i=1}^{\tau} r(C_i^*)$. Then, $r(\hat{C}_j) \leq d(p,q) \leq \sum_{e\in\pi'} \omega_e + \sum_{\text{vertex } v_i\in\pi'} 2 \cdot r(C_i^*) \leq 8 \cdot \sum_{i=1}^{\tau} r(C_i^*)$.

Summing over all clusters $\hat{C}_j$ in $\hat{\mathcal{C}}$, we obtain Lemma 3.

3.3 Proof of Lemma 4

The overall idea is to show the existence of a subset of edges $E_2' \subset E$, such that the set of edges $(E' \setminus \pi^*) \cup E_2'$ form a valid degree-constrained subgraph of G on the set of vertices $P_1 \cup P_2$. Additionally, we need that the total weight of the edges of E_2' is small. Then we can show that the weight of π^* is also small, as $H = (V, E')$ is a min-cost DCS of G. However, it might not be possible to remove only the edges of π^* from E' to show the existence of such a set E_2'. We show that there is a subset $E_1' \subseteq E'$ that contains the edges of π^* and can be removed to obtain such a valid degree-constrained subgraph. In the following, we prove that obtaining two such sets E_1' and E_2' is sufficient to prove Lemma 4. For a set of edges $S \subseteq E$, let $w(S) = \sum_{e\in S} w(e)$.

▶ **Lemma 5.** *Suppose there are $E_1' \subseteq E', E_2' \subset E$, such that the set of edges $(E' \setminus E_1') \cup E_2'$ forms a valid degree-constrained subgraph of G and $w(E_2') \leq 6 \cdot \sum_{i=1}^{\tau} r(C_i^*)$. Then, $w(E_1') \leq 6 \cdot \sum_{i=1}^{\tau} r(C_i^*)$.*

Proof. Note that H is a min-cost DCS of G. Consider the graph H' induced by the set of edges $(E' \setminus E_1') \cup E_2'$. By our assumption, H' is a valid DCS of G. It follows that,

$$w(E') \leq w((E' \setminus E_1') \cup E_2'), \text{or, } w(E_1') \leq w(E_2') \leq 6 \cdot \sum_{i=1}^{\tau} r(C_i^*).$$

The last inequality follows from our assumption. ◀

Assuming that the conditions of the above lemma are true, we finish the proof of Lemma 4.

$$\sum_{e\in\pi^*} \omega_e = \sum_{(v_i,v_j)\in\pi^*} \omega_e$$

$$= \sum_{\{p,q\} \text{ corresponding to } (v_i,v_j)\in\pi^* | p\in C_i^*, q\in C_j^*} w(\{p,q\}) \leq w(E_1') \leq 6 \cdot \sum_{i=1}^{\tau} r(C_i^*).$$

If π^* does not have a switch, then we will show that $w(E_2') \leq 4 \cdot \sum_{i=1}^{\tau} r(C_i^*)$. Hence, the moreover part in Lemma 4 also follows. It is left to show the existence of such E_1' and E_2'.

3.4 Construction of E'_1 and E'_2

Two subgraphs G_1 and G_2 of G^* are called *0-1-edge-disjoint* if for any edge e_{η^1} of G_1 and e_{η^2} of G_2, the corresponding edges in E' are distinct. Thus, if G_1 contains a 0-edge (v_i, v_j), and G_1 and G_2 are 0-1-edge-disjoint, then G_2 cannot contain the 0-edge (v_i, v_j) and the 1-edge (v_j, v_i). Similarly, if G_1 contains a 1-edge (v_i, v_j), and G_1 and G_2 are 0-1-edge-disjoint, then G_2 cannot contain the 1-edge (v_i, v_j) and the 0-edge (v_j, v_i). Let $j^1 < j^2 < \ldots < j^\lambda$ be the indexes of the vertices on $\pi^* = \{u_1, \ldots, u_l\}$ where the switches occur. Note that $j^1 > 1, j^\lambda < l$. Denote the switch that occurs at u_{j^h} by b^h for all $1 \leq h \leq \lambda$ (i.e., b^h is the parity of (u_{j^h-1}, u_{j^h})). Let b^0 be the parity of (u_1, u_2) and $b^{\lambda+1}$ be the parity of (u_{l-1}, u_l).

First, we consider the simple case when the parity of (u_{l-1}, u_l) is 0 (resp. 1) and there is a 0-path (resp. 1-path) from u_l to u_1 in G^* that is 0-1-edge-disjoint from π^*. Let us denote the latter path by $\pi(l)$. Note that π^* is a path having the minimum number of switches and the existence of $\pi(l)$ ensures that π^* does not have a switch. Let $U_0 \subseteq E^*$ be the subset of edges that lie on the paths in $\{\pi^*\} \cup \{\pi(l)\}$. Next, we define a subset $E'_1 \subseteq E_1$ that has a one-to-one mapping with U_0. In particular, consider any edge (v_i, v_j) in U_0. Note that if it is a 0-edge, it was added due to an edge $\{p, q\}$ in E_1 such that $p \in P_1 \cap C^*_i$ and $q \in P_2 \cap C^*_j$. We add the edge $\{p, q\}$ to E'_1. Otherwise, if (v_i, v_j) is a 1-edge, it was added due to an edge $\{p, q\}$ in E_1 such that $p \in P_2 \cap C^*_i$ and $q \in P_1 \cap C^*_j$. In this case, we add the edge $\{p, q\}$ to E'_1.

Next, we show the construction of E'_2. Wlog, let us assume that π^* is a 0-path. The other case is symmetric. Note that then $\pi(l)$ is also a 0-path as per our assumption. First, we describe the process of adding the replacement edges for the path π^*. Consider any intermediate vertex (if any) $v_{j'}$ on this path. Then, there are exactly two points in $C^*_{j'}$ corresponding to the edges on π^*, which are of opposite colors. We add an edge between these two points in E'_2 (see Figure 1). Removal of the edges of E'_1 corresponding to π^* and the addition of this edge do not change the degree of the two points in $C^*_{j'}$. Similarly, we add edges to E'_2 corresponding to the intermediate vertices of $\pi(l)$. Next, consider the vertex $u_l = v_i$. There is an incoming 0-edge on π^* and an outgoing 0-edge on $\pi(l)$ that are incident on v_i. Thus, there are exactly two points in C^*_i of opposite colors corresponding to these two edges. We add an edge between these two points in E'_2. Removal of the edges of E'_1 corresponding to those two edges, and the addition of this edge does not change the degree of the two points in C^*_i. Similarly, consider the vertex $u_1 = v'_i$. There is an outgoing 0-edge on π^* and an incoming 0-edge on $\pi(l)$ that are incident on v'_i. Thus, there are exactly two points in $C^*_{i'}$ of opposite colors corresponding to these two edges. We add an edge between these two points in E'_2. Again, the removal of the edges of E'_1 corresponding to those two edges, and the addition of this edge does not change the degree of the two points in $C^*_{i'}$. See Figure 1 for an illustration.

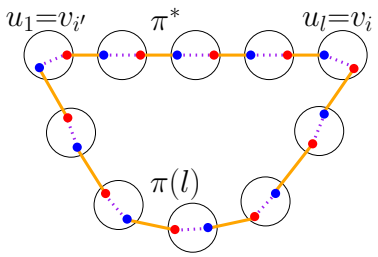


Figure 1 Figure illustrating the construction of E'_2 for $\{\pi^*\} \cup \{\pi(l)\}$. The bold (orange) edges are in E'_1 and the dashed (purple) edges are in E'_2.

By our construction, the set of edges $(E' \setminus E'_1) \cup E'_2$ form a valid degree-constrained subgraph of G on the set of vertices $P_1 \cup P_2$. The way we add the edges to E'_2, both endpoints of each edge lie in a cluster C^*_j such that the vertex v_j corresponding to the cluster lies on a path in $\{\pi^*\} \cup \{\pi(l)\}$. Now, v_j can lie either on one such path or on two paths. Thus, we add at most two edges to E'_2 corresponding to v_j. The sum of the weights of these two edges is at most 2 times the diameter of C^*_j. Hence, by Lemma 5, we obtain $\sum_{e \in \pi^*} \omega_e \leq 4 \cdot \sum_{i=1}^{\tau} r(C^*_i)$.

Next, we consider the remaining case when the parity of (u_{l-1}, u_l) is 0 (resp. 1) and there is no 0-path (resp. 1-path) from u_l to u_1 in G^* that is 0-1-edge-disjoint from π^*. Thus, there is no $b^{\lambda+1}$-path from u_l to u_1 in G^* that is 0-1-edge-disjoint from π^*.

Consider a path π and let v_i denote its start vertex. Also, consider a cycle O such that π and O have exactly one vertex v_j in common. Note that π might not have an edge, in which case $v_j = v_i$. Let D be the graph formed by the union of π and O, i.e., by gluing them together at v_j. We refer to such a graph D as a *hanging cycle* for v_i with v_j being the *join* vertex. D is called a b-hanging cycle if all the edges of π and O are b-edges. Let p be the point in the cluster C^*_j corresponding to the edge of the cycle O incoming to v_j. Additionally, D is called *special* if the degree of p in H_1 is at least 2, and p is called the *special* point of D (see Figure 2).

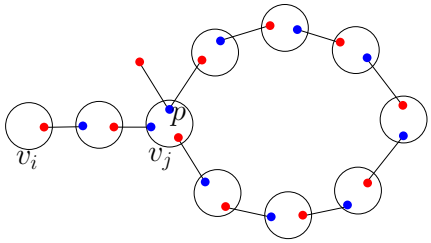


Figure 2 A special hanging cycle with the special point p.

In the current case, we need the following lemmas.

▶ **Lemma 6** (∗). *Suppose the parity of (u_{l-1}, u_l) is 0 (resp. 1), and there is no 0-path (resp. 1-path) from u_l to u_1 in G^* that is 0-1-edge-disjoint from π^*. Moreover, suppose there is no special 0-hanging cycle (resp. 1-hanging cycle) in G^* for u_l that is 0-1-edge-disjoint from π^*. Then, there exists a 0-path (resp. 1-path) π_1 in G^* from u_l to a vertex v_j, such that π_1 is 0-1-edge-disjoint from π^* and one of the following is true: (i) the degree of b_η (resp. r_η) in H_1 is at least 2, where e_η is the last edge on π_1 if it has an edge or (u_{l-1}, u_l) otherwise; or (ii) C^*_j has a red (resp. blue) point whose degree in H_1 is at most $t-1$.*

▶ **Lemma 7** (∗). *Suppose the parity of (u_1, u_2) is 1 (resp. 0), and there is no 1-path (resp. 0-path) from u_l to u_1 in G^* that is 0-1-edge-disjoint from π^*. Moreover, suppose there is no special 0-hanging cycle (resp. 1-hanging cycle) in G^* for u_1 that is 0-1-edge-disjoint from π^*. Then, there exists a 0-path (resp. 1-path) π_1 from u_1 to a vertex v_j, such that π_1 is 0-1-edge-disjoint from π^* and one of the following is true: (i) the degree of b_η (resp. r_η) in H_1 is at least 2, where e_η is the last edge on π_1 if it has an edge or (u_2, u_1) otherwise, and (ii) C^*_j has a red (resp. blue) point whose degree in H_1 is at most $t-1$.*

Next, we apply the above lemmas to show the construction of E'_1 and E'_2. **Recall that in this case there is no $b^{\lambda+1}$-path from u_l to u_1 in G^* that is 0-1-edge-disjoint from π^*.** If π^* has a switch, then there is no 0-path or 1-path from u_l to u_1 in G^* that is

0-1-edge-disjoint from π^*. Otherwise, it must be that $b^0 = b^{\lambda+1}$, and hence by our assumption, there is no b^0-path from u_l to u_1 in G^* that is 0-1-edge-disjoint from π^*. We conclude that in this case ($b^0 = b^{\lambda+1}$), there is no b^0-path from u_l to u_1 in G^* that is 0-1-edge-disjoint from π^*. Then, by Lemma 7 it follows that, either there is a $(1-b^0)$-hanging cycle for u_1 in G^* 0-1-edge-disjoint from π^*, or a $(1-b^0)$-path starting from u_1 in G^* with special properties. This is true, as the parity of (u_1, u_2) is b^0. We denote this structure by $\pi(0)$. Additionally, if $\pi(0)$ is a path, we call b_η (resp. r_η) an *anchor* point if its degree in H_1 is at least 2. Similarly, by Lemma 6 it follows that, either there is a $b^{\lambda+1}$-hanging cycle for u_l in G^* 0-1-edge-disjoint from π^*, or a $b^{\lambda+1}$-path starting from u_l in G^* with special properties. This is true, as (u_{l-1}, u_l) is a $b^{\lambda+1}$-edge. We denote this structure by $\pi(\lambda+1)$. Additionally, if $\pi(\lambda+1)$ is a path, we call b_η (resp. r_η) an *anchor* point if its degree in H_1 is at least 2.

Now, if $\mathbf{1 - b^0 \neq b^{\lambda+1}}$, then $\pi(0)$ and $\pi(\lambda+1)$ must be vertex-disjoint. If they are not vertex-disjoint, there exists a $b^{\lambda+1}$-path from u_l to u_1 in G^* that is 0-1-edge-disjoint from π^*: take the edges on $\pi(\lambda+1)$ from u_l to a common vertex and the reverse of $\pi(0)$, from the common vertex to u_1. These reverse edges have parity opposite of $1-b^0$, i.e., the same as $b^{\lambda+1}$. But, by our assumption, such a $b^{\lambda+1}$-path does not exist. Hence, $\pi(0)$ and $\pi(\lambda+1)$ are vertex-disjoint.

In the other case, $\mathbf{1 - b^0 = b^{\lambda+1}}$. We note that if π^* has no switch, $b^0 = b^{\lambda+1}$. Thus, if $1 - b^0 = b^{\lambda+1}$, then we can safely assume that π^* has at least one switch. In this case, suppose there are a $(1-b^0)$-hanging cycle for u_1 and a $b^{\lambda+1}$-hanging cycle for u_l in G^*, such that both are 0-1-edge-disjoint, each of the hanging cycles is 0-1-edge-disjoint from π^*, and either the special vertices of both are distinct or the special points are the same and the degree of that point in H_1 is at least 3. Then, we take the hanging cycle for u_1 as $\pi(0)$ and the one for u_l as $\pi(\lambda+1)$. Otherwise, if there is a $(1-b^0)$-hanging cycle for u_1 in G^* 0-1-edge-disjoint from π^* or a $b^{\lambda+1}$-hanging cycle for u_l in G^* 0-1-edge-disjoint from π^*, we consider one of those. Assume that the former holds. The other case is symmetric. We take such a hanging cycle for u_1 as $\pi(0)$. Then, one can prove that there is a $b^{\lambda+1}$-path from u_l in G^* with special properties (Lemma 8 in the full version). We take this $b^{\lambda+1}$-path as $\pi(\lambda+1)$. Otherwise, there is neither a $(1-b^0)$-hanging cycle for u_1 in G^* 0-1-edge-disjoint from π^* nor a $b^{\lambda+1}$-hanging cycle for u_l in G^* 0-1-edge-disjoint from π^*. Then, one can prove that there are two 0-1-edge-disjoint paths with parity $1 - b^0 = b^{\lambda+1}$, from u_1 and u_l, respectively, such that both are also 0-1-edge-disjoint from π^* (Lemma 9 in the full version). In this case, we take the path from u_1 as $\pi(0)$ and the path from u_l as $\pi(\lambda+1)$.

For all $1 \leq h \leq \lambda$, if there are two 0-1-edge-disjoint b^h-hanging cycles for u_{j^h} in G^* that are 0-1-edge-disjoint from π^* and have distinct special points or the same special point of degree at least 3 in H_1, denote them by $\pi_1(h)$ and $\pi_2(h)$. Otherwise, if there is one b^h-hanging cycle for u_{j^h} in G^* that is 0-1-edge-disjoint from π^*, denote it by $\pi_1(h)$. Now, (u_{j^h-1}, u_{j^h}) is a b^h-edge and (u_{j^h}, u_{j^h+1}) is a $(1-b^h)$-edge. Then, one can prove that there is a b^h-path starting from u_{j^h} in G^* with special properties (Lemma 8 in the full version). Denote this path by $\pi_2(h)$. Otherwise, there is no b^h-hanging cycle for u_{j^h} in G^* that is 0-1-edge-disjoint from π^*. In this case, one can prove that there are two b^h-paths starting from u_{j^h} in G^* with special properties. Denote them by $\pi_1(h)$ and $\pi_2(h)$. Note that in all the cases, $\pi()$, $\pi_1()$ or $\pi_2()$ can either be a path or a hanging cycle.

Construction of E'_1. Let $U \subseteq E^*$ be the subset of edges that lie on the structures in $\mathcal{S} = \cup_{i=1}^{\lambda}(\{\pi_1(i)\} \cup \{\pi_2(i)\}) \cup \{\pi(0), \pi(\lambda+1), \pi^*\}$. Next, we define the subset E'_1 of E' that has a one-to-one mapping with U. In particular, consider any edge (v_i, v_j) in U. Note that if it is a 0-edge, it was added due to an edge $\{p, q\}$ in E_1 such that $p \in P_1 \cap C_i^*$ and $q \in P_2 \cap C_j^*$. We add the edge $\{p, q\}$ to E'_1. Otherwise, if (v_i, v_j) is a 1-edge, it was added due to an edge $\{p, q\}$ in E_1 such that $p \in P_2 \cap C_i^*$ and $q \in P_1 \cap C_j^*$. We again add the edge $\{p, q\}$ to E'_1.

Our proof is completed by the following two lemmas.

▶ **Lemma 8** (∗). *There is a subset of edges $E'_2 \subset E$, such that the set of edges $(E' \setminus E'_1) \cup E'_2$ form a valid degree-constrained subgraph of G on the set of vertices $P_1 \cup P_2$.*

4 The Algorithm for Balanced Sum-of-Radii Clustering

In this section, we prove Theorem 2. Recall that we are given ℓ disjoint groups $P_1, \ldots, P_\ell$ having n points in total in a metric space $(\Omega = \cup_{i=1}^{\ell} P_i, d)$, such that $|P_1| = |P_2| = \ldots = |P_\ell|$.

Our algorithm is as follows.

The Algorithm

1. For each $2 \leq i \leq \ell$, construct a graph $G_i = (V_i, E_i)$ where $V_i = P_1 \cup P_i$ and $E_i = \{\{p, q\} \mid p \in P_1, q \in P_i\}$. Define the weight function w_i such that for each edge $e = \{p, q\}$, $w_i(e) = d(p, q)$. Compute a minimum-weight (w.r.t. w_i) perfect matching M_i of G_i. For each $p \in P_1$, let S_p be the union of $\{p\}$ and the points from $P_2, \ldots, P_\ell$ that are matched to p in $M = \cup_{i=2}^{\ell} M_i$.
2. Construct an edge-weighted graph G' in the following way: For each $p \in \Omega$, add a vertex to G'; For each $p \in P_1$, add a vertex corresponding to S_p to G', which we also call by S_p; For each $p, q \in \Omega$, add the edge $\{p, q\}$ to G' with weight $d(p, q)$; For all $p' \in \Omega$ and $p \in P_1$, add the edge $\{p', S_p\}$ to G' with weight $\max_{q \in S_p} d(p', q)$. Let d' be the shortest path metric in G'. Construct the metric space (Ω', d') where Ω' is the subset of vertices $\{S_p \mid p \in P_1\}$ in G'.
3. Compute a sum of radii clustering $X = \{X_1, \ldots, X_k\}$ of the points in Ω' using the Algorithm of Buchem et al. [14] (with Ω' also being the candidate set of centers).
4. Compute a clustering X' of the points in $\cup_{i=1}^{\ell} P_i$ using X in the following way. For each cluster X_i, add the cluster $\cup_{q \in S_p | S_p \in X_i} \{q\}$ to X'. Return X'.

Let $\mathcal{C}^* = \{C_1^*, C_2^*, \ldots, C_k^*\}$ be a fixed optimal balanced clustering.

We have the following lemma. Our main result follows as a corollary.

▶ **Lemma 9** (∗). *Consider the clustering X of Ω' constructed in Step 3 of the algorithm. Then $cost_{(\Omega', d')}(X) \leq (\mathbf{60} + \epsilon) \cdot \sum_{i=1}^{k} r_{(\Omega, d)}(C_i^*)$.*

▶ **Corollary 2** (∗). *Consider the clustering X' of $\cup_{i=1}^{\ell} P_i$ constructed in Step 3 of the algorithm. Then $cost_{(\Omega, d)}(X') \leq (\mathbf{180} + \epsilon) \cdot \sum_{i=1}^{k} r_{(\Omega, d)}(C_i^*)$. Thus, our algorithm is a $(\mathbf{180} + \epsilon)$-approximation algorithm.*

References

1 Anders Aamand, Justin Y. Chen, Allen Liu, Sandeep Silwal, Pattara Sukprasert, Ali Vakilian, and Fred Zhang. Constant approximation for individual preference stable clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

2 Mohsen Abbasi, Aditya Bhaskara, and Suresh Venkatasubramanian. Fair clustering via equitable group representations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAccT)*, pages 504–514, 2021. doi:10.1145/3442188.3445913.

3 Sara Ahmadian and Chaitanya Swamy. Approximation Algorithms for Clustering Problems with Lower Bounds and Outliers. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 69:1–69:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2016.69.

4 Georg Anegg, Haris Angelidakis, Adam Kurpisz, and Rico Zenklusen. A technique for obtaining true approximations for k-center with covering constraints. In *International conference on integer programming and combinatorial optimization*, pages 52–65. Springer, 2020. doi:10.1007/978-3-030-45771-6_5.

5 Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In *International Conference on Machine Learning*, pages 405–413, 2019. URL: http://proceedings.mlr.press/v97/backurs19a.html.

6 Sayan Bandyapadhyay, Eden Chlamtáč, Yury Makarychev, and Ali Vakilian. A polynomial-time approximation for pairwise fair k-median clustering. *arXiv preprint arXiv:2405.10378*, 2024. doi:10.48550/arXiv.2405.10378.

7 Sayan Bandyapadhyay, Tanmay Inamdar, Shreyas Pai, and Kasturi Varadarajan. A constant approximation for colorful k-center. In *27th Annual European Symposium on Algorithms (ESA 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

8 Sayan Bandyapadhyay, William Lochet, and Saket Saurabh. FPT constant-approximations for capacitated clustering to minimize the sum of cluster radii. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA*, volume 258 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SOCG.2023.12.

9 Babak Behsaz and Mohammad R. Salavatipour. On minimum sum of radii and diameters clustering. *Algorithmica*, 73(1):143–165, 2015. doi:10.1007/s00453-014-9907-3.

10 Suman Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *Advances in Neural Information Processing Systems*, pages 4954–4965, 2019.

11 Ioana O Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.18.

12 Matteo Böhm, Adriano Fazzone, Stefano Leonardi, and Chris Schwiegelshohn. Fair clustering with multiple colors. *arXiv preprint arXiv:2002.07892*, 2020. arXiv:2002.07892.

13 B Brubach, D Chakrabarti, J Dickerson, A Srinivasan, and L Tsepenekas. Fairness, semi-supervised learning, and more: A general framework for clustering with stochastic pairwise constraints. In *Proc. Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

14 Moritz Buchem, Katja Ettmayr, Hugo KK Rosado, and Andreas Wiese. A $(3+\epsilon)$-approximation algorithm for the minimum sum of radii problem with outliers and extensions for generalized lower bounds. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1738–1765. SIAM, 2024.

15 Lena Carta, Lukas Drexler, Annika Hennes, Clemens Rösner, and Melanie Schmidt. FPT Approximations for Fair k-Min-Sum-Radii. In Julián Mestre and Anthony Wirth, editors, *35th International Symposium on Algorithms and Computation (ISAAC 2024)*, volume 322 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2024.16.

16 Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *J. Comput. Syst. Sci.*, 68(2):417–441, 2004. doi:10.1016/j.jcss.2003.07.014.

17 Danny Z Chen, Jian Li, Hongyu Liang, and Haitao Wang. Matroid and knapsack center problems. *Algorithmica*, 75(1):27–52, 2016. doi:10.1007/S00453-015-0010-1.

18 Xianrun Chen, Dachuan Xu, Yicheng Xu, and Yong Zhang. Parameterized approximation algorithms for sum of radii clustering and variants. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20666–20673, 2024. doi:10.1609/AAAI.V38I18.30053.

19 Xingyu Chen, Brandon Fain, Liang Lyu, and Kamesh Munagala. Proportionally fair clustering. In *International Conference on Machine Learning*, pages 1032–1041, 2019. URL: http://proceedings.mlr.press/v97/chen19d.html.

20 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems*, pages 5029–5037, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/978fce5bcc4eccc88ad48ce3914124a2-Abstract.html.

21 Ashish Chiplunkar, Sagar Kale, and Sivaramakrishnan Natarajan Ramamoorthy. How to solve fair k-center in massive data models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1877–1886, 2020.

22 Eden Chlamtáč, Yury Makarychev, and Ali Vakilian. Approximating fair clustering with cascaded norm objectives. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2664–2683, 2022. doi:10.1137/1.9781611977073.104.

23 Zhen Dai, Yury Makarychev, and Ali Vakilian. Fair representation clustering with several protected classes. In *FAccT '22: 2022 ACM Conference on Fairness, Accountability, and Transparency, Seoul, Republic of Korea, June 21 - 24, 2022*, pages 814–823. ACM, 2022. doi:10.1145/3531146.3533146.

24 Lukas Drexler, Annika Hennes, Abhiruk Lahiri, Melanie Schmidt, and Julian Wargalla. Approximating fair k-min-sum-radii in euclidean space. In *International Workshop on Approximation and Online Algorithms*, pages 119–133. Springer, 2023. doi:10.1007/978-3-031-49815-2_9.

25 Arnold Filtser and Ameet Gadekar. Fpt approximations for capacitated sum of radii and diameters. *arXiv preprint arXiv:2409.04984*, 2024. doi:10.48550/arXiv.2409.04984.

26 Zachary Friggstad and Mahya Jamshidian. Improved Polynomial-Time Approximations for Clustering with Minimum Sum of Radii or Diameters. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.56.

27 Harold N Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 448–456, 1983. doi:10.1145/800061.808776.

28 Mehrdad Ghadiri, Samira Samadi, and Santosh S. Vempala. Socially fair k-means clustering. In Madeleine Clare Elish, William Isaac, and Richard S. Zemel, editors, *FAccT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3-10, 2021*, pages 438–448. ACM, 2021. doi:10.1145/3442188.3445906.

29 Mehrdad Ghadiri, Mohit Singh, and Santosh S Vempala. Constant-factor approximation algorithms for socially fair k-clustering. *arXiv preprint arXiv:2206.11210*, 2022. doi:10.48550/arXiv.2206.11210.

30 Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi R. Varadarajan. On metric clustering to minimize the sum of radii. *Algorithmica*, 57(3):484–498, 2010. doi:10.1007/s00453-009-9282-7.

31 Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi R. Varadarajan. On clustering to minimize the sum of radii. *SIAM J. Comput.*, 41(1):47–60, 2012. doi:10.1137/100798144.

32 Swati Gupta, Jai Moondra, and Mohit Singh. Which l_p norm is the fairest? approximations for fair facility location across all "p", 2022. arXiv:2211.14873.

33 Pinar Heggernes and Daniel Lokshtanov. Optimal broadcast domination in polynomial time. *Discret. Math.*, 306(24):3267–3280, 2006. doi:10.1016/j.disc.2006.06.013.

34 Sedjro Salomon Hotegni, Sepideh Mahabadi, and Ali Vakilian. Approximation algorithms for fair range clustering. In *International Conference on Machine Learning*, pages 13270–13284. PMLR, 2023.

35 Tanmay Inamdar and Kasturi R. Varadarajan. Capacitated sum-of-radii clustering: An FPT approximation. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 62:1–62:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.62.

36 Ragesh Jaiswal, Amit Kumar, and Jatin Yadav. FPT approximation for capacitated sum of radii. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pages 65:1–65:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ITCS.2024.65.

37 Xinrui Jia, Kshiteej Sheth, and Ola Svensson. Fair colorful k-center clustering. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 209–222. Springer, 2020. doi:10.1007/978-3-030-45771-6_17.

38 Christopher Jung, Sampath Kannan, and Neil Lutz. A center in your neighborhood: Fairness in facility location. In *Proceedings of the Symposium on Foundations of Responsible Computing (FORC)*, pages 5:1–5:15, 2020.

39 Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. Fair k-center clustering for data summarization. In *36th International Conference on Machine Learning, ICML 2019*, pages 5984–6003. International Machine Learning Society (IMLS), 2019.

40 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 646–659, 2018. doi:10.1145/3188745.3188882.

41 Yury Makarychev and Ali Vakilian. Approximation algorithms for socially fair clustering. In *Conference on Learning Theory (COLT)*, pages 3246–3264. PMLR, 2021. URL: http://proceedings.mlr.press/v134/makarychev21a.html.

42 Evi Micha and Nisarg Shah. Proportionally fair clustering revisited. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2020. doi:10.4230/LIPIcs.ICALP.2020.85.

43 Maryam Negahbani and Deeparnab Chakrabarty. Better algorithms for individually fair k-clustering. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:13340–13351, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/6f221fcb5c504fe96789df252123770b-Abstract.html.

44 Sina Bagheri Nezhad, Sayan Bandyapadhyay, and Tianzhi Chen. Polynomial-time constant-approximation for fair sum-of-radii clustering, 2025. doi:10.48550/arXiv.2504.14683.

45 Fatemeh Rajabi-Alni and Alireza Bagheri. Computing a many-to-many matching with demands and capacities between two sets using the hungarian algorithm. *Journal of mathematics*, 2023(1):7761902, 2023.

46 Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k-means. In *International Workshop on Approximation and Online Algorithms*, pages 232–251. Springer, 2019. doi:10.1007/978-3-030-39479-0_16.

47 Ali Vakilian and Mustafa Yalçıner. Improved approximation algorithms for individually fair clustering. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 8758–8779. PMLR, 2022.

Connected k-Median with Disjoint and Non-Disjoint Clusters

Jan Eube
University of Bonn, Germany

Kelin Luo
University at Buffalo, NY, USA

Dorian Reineccius
Deutsches Zentrum für Luft- und Raumfahrt, Bonn, Germany

Heiko Röglin
University of Bonn, Germany

Melanie Schmidt
Heinrich-Heine Universität Düsseldorf, Germany

Abstract

The connected k-median problem is a constrained clustering problem that combines distance-based k-clustering with connectivity information. The problem allows to input a metric space and an unweighted undirected connectivity graph that is completely unrelated to the metric space. The goal is to compute k centers and corresponding clusters such that each cluster forms a connected subgraph of G, and such that the k-median cost is minimized.

The problem has applications in very different fields like geodesy (particularly districting), social network analysis (especially community detection), or bioinformatics. We study a version with overlapping clusters where points can be part of multiple clusters which is natural for the use case of community detection. This problem variant is $\Omega(\log n)$-hard to approximate, and our main result is an $\mathcal{O}(k^2 \log n)$-approximation algorithm for the problem. We complement it with an $\Omega(n^{1-\epsilon})$-hardness result for the case of disjoint clusters without overlap with general connectivity graphs, as well as an exact algorithm in this setting if the connectivity graph is a tree.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Clustering, Connectivity constraints, Approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.63

Related Version *Full Version*: `https://arxiv.org/abs/2507.02774` [7]

Funding This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 390685813; 459420781 and by the Lamarr Institute for Machine Learning and Artificial Intelligence lamarr-institute.org.

1 Introduction

Clustering problems, like the k-center and the k-median problem, are classical optimization problems that have been widely studied both in theory and practice. In a typical center-based clustering problem, the input consists of a given set of data points in some metric space and a positive integer k. The goal is to partition the data points into k groups (clusters) and to find a center for each of these groups so as to optimize some objective: In the k-center problem, the goal is to minimize the maximal distance of any point to the center of its cluster; in the k-median problem, the goal is to minimize the sum of the distances of the points to their corresponding cluster centers.


© Jan Eube, Kelin Luo, Dorian Reineccius, Heiko Röglin, and Melanie Schmidt;
licensed under Creative Commons License CC-BY 4.0



Both the k-center and the k-median problem are NP-hard but various constant-factor approximation algorithms are known. For the k-center problem, there are 2-approximation algorithms [9, 13], and this is best possible [14]. The best known approximation algorithm for the k-median problem achieves a $(2.675 + \epsilon)$-approximation [2], while it is known that no approximation factor better than $1 + 2/e \approx 1.735$ can be achieved, unless P=NP [10].

Beyond these vanilla clustering problems, there has been a lot of interest in variants of clustering problems motivated by specific applications. These variants often introduce additional constraints. For example, in capacitated clustering problems, constraints are added that limit the number of data points that can be assigned to each cluster [17]. There are also models to address outliers in data sets [18], and in the last decade, different variants of fair clustering problems have been studied [4, 20, 21]. These are just a few examples of the numerous variants of clustering problems that have been studied to address diverse challenges in different application domains. Each of these variants requires specialized algorithms, often inspiring novel methodologies and insights in the broader field of clustering and data analysis.

In this paper, we study a *connectivity* constraint. The basic idea of this constraint is that clusters have to correspond to connected subgraphs of a given undirected and unweighted *connectivity graph*. This graph is independent of the metric that defines the distances between the input points.

This model has been studied in different contexts. One line of research, see Ge et al. [8], considers the connected k-center problem to allow for a joint analysis of two type of data: attribute data and relationship data. Attribute data can be used for clustering by finding a suitable metric distance function on the attribute vectors and then applying a popular clustering method like k-means or k-center. Relationship data arises in the context of social networks or other network structures and gives information on pairwise relationships. Clustering on relationship data can be done via methods like correlation clustering. Both types of data thus allow for well studied clustering methods, but the focus by Ge et al. [8] is a joint analysis. Connected clustering gives one option to combine relationship and attribute data by defining a constraint based on the relationships and then optimizing over the attribute vector distances.

More precisely, connected clustering expects as input a set of points in a metric space (stemming from the attribute data) and an unweighted graph G (representing the relationship data), and the goal is to cluster with respect to the metric distance function but under the constraint that clusters are connected in G. We call G the *connectivity graph* in the following.

Ge et al. [8] study the connected k-center problem. For this problem, the goal is to compute k centers $c_1, \ldots, c_k$ and corresponding clusters $C_1, \ldots, C_k$ such that all C_i are connected in the connectivity graph G, and such that the largest radius is minimized (the radius of C_i is $\max_{x \in C_i} d(x, c_i)$). They provide an optimal algorithm via dynamic programming for the special case of a tree connectivity graph which was rediscovered by Drexler et al. [6] in a later publication. Ge et al. also present algorithms for the general case with unknown approximation guarantee[1] and provide an experimental evaluation.

The best-known approximation guarantee for the connected k-center problem is a $O(\log^2 k)$-approximation given by Drexler et al. [6]. For Euclidean metrics with constant dimension and metrics with constant doubling dimension, they provide an $O(1)$-approximation algorithm. Their paper studies the problem from a very different context: Motivated by an application from geodesy and sea level analysis, they use the distance function to model differences in tide gauge measurements and the connectivity graph to model geographic closeness, and their modelling results in the same problem definition as Ge et al. [8].

[1] There is a proof in the paper that claims a guarantee of 6, but [6] gives a counterexample.

In this paper, we study the connected k-*median* problem, i.e., the related problem with the k-median objective instead of the k-center objective. We give a formal definition now.

▶ **Definition 1** (Connected k-Median Problem (disjoint))**.** *In an instance of the (disjoint) connected k-median problem, we are given a set V of points, a metric $d : V \times V \to \mathbb{R}_{\geq 0}$ on V, a positive integer $k \geq 2$, and an unweighted and undirected connectivity graph $G = (V, E)$. A feasible solution is a partition of V into k disjoint clusters $\mathcal{V} = \{V_1, \dots, V_k\}$ with corresponding centers $C = \{c_1, c_2, \dots, c_k\} \subseteq V$ which satisfies that for every $i \in [k] := \{1, \dots, k\}$ the subgraph of G induced by V_i is connected and $c_i \in V_i$. The* connected k-median problem *aims to find a feasible solution $(C, \mathcal{V})$ that minimizes*

$$\Phi_{med}(C, \mathcal{V}) := \sum_{i \in [k]} \sum_{v \in V_i} d(c_i, v).$$

The connected k-median has also appeared in different contexts. For example, Validi et al. [22] study connected k-median for a districting problem. Here the goal is to partition given administrative or geographic units into districts. The clusters should usually correspond to connected areas on the map which can be formulated via introducing centers for each unit and a graph to model which units are neighbors on the map. The k-median objective is one way to model compactness of districts. Validi et al. [22] study various districting methods, and connected k-median is one of them. They solve it via an integer linear programming approach, i.e., to optimality but with exponential worst-case running time. A related variant is the connected k-means problem, which for example has been considered in [19] where a heuristic approach to minimize it is employed.

Gupta et al. [12] study k-median/k-means from the point of view of approximation algorithms. They consider the special case where the number of clusters is $k = 2$ and the connectivity graph is star-like (there exists a point that is connected to all other points). The paper shows that even this case is $\Omega(\log n)$-hard to approximate for both problems and proposed an $O(\log n)$-approximation algorithm for this special case. There is nothing known about the general case, and our work is motivated by the following question:

Can the connected k-median problem be approximated for general connectivity graphs?

Unfortunately, we quickly arrive at the conclusion that no reasonable approximation guarantee is possible if $P \neq NP$ (see Thm 4). We do derive a dynamic programming based exact algorithm for trees that runs in time $O(n^2k^2)$ (see Thm 6), thus extending the landscape for the problem, but the answer to our main question is still negative.

However, our inapproximability proof depends on a subtlety of the problem definition: The clusters V_i have to be disjoint. This is a natural assumption in clustering, and in clustering without constraints, clusters can be assumed to be disjoint (adding a point to multiple clusters will in general only increase the objective). But here, if we allow for overlapping clusters, then satisfying connectivity can become easier because we can add critical vertices like articulation points to multiple clusters. This motivates the definition of *connected k-median with non-disjoint clusters.*

▶ **Definition 2** (Disjoint vs. non-disjoint clusters)**.** *In the* connected k-median problem with disjoint clusters*, we require the clusters $V_1, \dots, V_k$ to be pairwise disjoint. In the* connected k-median problem with non-disjoint clusters*, the clusters are allowed to overlap and we only require $V = \bigcup_{i \in [k]} V_i$.*

One may note that if a vertex appears in multiple clusters it will also contribute to the objective function multiple times. Thus also in the non-disjoint variant, we are incentivized to avoid assigning vertices unnecessarily often.

The non-disjoint version has also been studied previously. For example, Drexler et al. [6] study it for k-center and give a 2-approximation for this case. From the application side, it depends on the interpretation whether disjoint or non-disjoint clusters are preferred. In districting, units should certainly not be in multiple clusters. But for the application of community detection in social networks the modeling is natural since individuals can belong to different groups. We thus study the following question.

Can the connected k-median problem with non-disjoint clusters be approximated for general connectivity graphs?

The proof by Gupta et al. [12] gives a lower bound of $\Omega(\log n)$ for the disjoint case. For the sake of completeness, we provide in the full version of this paper an adapted version that also works if the clusters may overlap [7]. Our main result is a $O(k^2 \log n)$-approximation for the connected k-median problem with non-disjoint clusters.

Before we summarize our results, we need one more definition. For a clustering problem with constraints, the corresponding *assignment* version of the problem asks for assigning points to given centers while respecting the constraints and minimizing the cost. In our case, we get the following problem.

▶ **Definition 3** (Assignment version)**.** *In the* assignment version *of the connected k-median problem, we assume that k centers, denoted as $C \subseteq V$, are given and the goal is to find a feasible assignment of the points in V to the centers C such that the k-median costs are minimized. This version can be defined both for the disjoint and the non-disjoint variant of the connected k-median problem.*

Without a constraint, the assignment version of the k-median problem can be solved optimally by assigning every point to its closest center. For clustering with constraints, assignment problems can be difficult: For example, the assignment problem for fair k-center is NP-hard [1], and the connected k-center with disjoint clusters is also NP-hard [6] (both problems are even hard to approximate better than 3). We will establish an even higher lower bound for the assignment problem in the connected k-median setting.

1.1 Our results

Table 1 Overview of the results presented in this work.

(a) Our results in the non-disjoint setting

	Assignment version	Regular version
Approximation	$O(k \log(n))$ (Theorem 8)	$O(k^2 \log(n))$ (Theorem 9)
Hardness($k = 2$)	$\Omega(\log(n))$ (Theorem 7)	

(b) Our results in the disjoint setting. All results also hold for the assignment version.

	General Graph	Tree graph
Approximation	$O(n \log^2(k))$ (Theorem 5)	1 (Theorem 6)
Hardness($k = 2$)	$\Omega(n^{1-\epsilon})$ (Theorem 4)	–

We prove new approximation and hardness results for the connected k-median problem, both for the disjoint and the non-disjoint variant. Table 1 provides a brief overview of them. First we establish a strong hardness of approximation result for the connected k-median problem with disjoint clusters, which even holds in the special case that $k = 2$.

▶ **Theorem 4.** *For any $\epsilon > 0$, there is no polynomial-time $O(n^{1-\epsilon})$-approximation algorithm for the connected k-median problem with disjoint clusters even if $k = 2$, unless $P = NP$. For the assignment version, this lower bound also holds.*

For completeness we point out that one can almost match this lower bound using the connected k-center algorithm by Drexler et al. [6]:

▶ **Theorem 5.** *For the disjoint connected k-median problem, there is a polynomial-time algorithm with approximation factor $O(n \log^2 k)$.*

Another important special case that has also been studied for the connected k-center problem is the case that the connectivity graph is a tree. For this special case, we obtain a polynomial-time algorithm based on dynamic programming.

▶ **Theorem 6.** *When the connectivity graph G is a tree, then the connected k-median problem with disjoint clusters can be solved optimally in time $O(n^2k^2)$. This is true even if the distances are not a metric or the centers are given.*

Motivated by applications like community detection, in which the clusters do not necessarily have to be disjoint, we also study the connected k-median problem with non-disjoint clusters. This problem is $\Omega(\log n)$-hard to approximate even if $k = 2$.

▶ **Theorem 7.** *There is a constant $c > 0$ such that for all sufficiently large n, the non-disjoint connected k-median problem cannot be approximated within a factor of $c \cdot \log n$ unless P=NP, even when $k = 2$. The same is true for the assignment variant.*

Our main result are approximation algorithms for the clustering and the assignment problem whose approximation factors match this lower bound in terms of n.

▶ **Theorem 8.** *For the assignment version of the non-disjoint connected k-median problem, there is a polynomial-time algorithm with approximation factor $O(k \log n)$. This is true even if the distances are not a metric.*

▶ **Theorem 9.** *For the non-disjoint connected k-median problem, there is a polynomial-time algorithm with approximation factor $O(k^2 \log n)$.*

1.2 Outline

The remainder of this paper focuses on the non-disjoint variant. For the disjoint results we refer to the full version [7]. In Section 2.1 we present our algorithm for the assignment version and in Section 2.2 we give an overview on how the centers can be found. The full description and analysis of the latter algorithm as well as the hardness proof for this setting can again be found in the full version of this work.

2 Non-disjoint Connected k-Median Problem

2.1 Assignment Version

The high-level idea to solve the assignment version of the connected k-median problem is to first define an LP formulation for it and compute an optimum fractional solution for the LP relaxation in polynomial time. Using an observation that the connected clusters are similar to so called node weighted Steiner trees we can use this LP solution to obtain k instances of

the node weighted Steiner tree problem such that every node needs to be contained in at least one Steiner tree and there exist a fractional solution for each of these instances such that the total cost of these solutions is bounded by k times the optimum LP solution. Using a primal-dual algorithm by Klein and Ravi [15] we are able to find integral Steiner trees that cost at most $O(\log n)$ times as much as the respective fractional solutions and which directly form k connected clusters assigning each node to at least one center. In total, this gives us an approximation factor of $O(k \log n)$.

Now we will define the LP formulation. For that, we use the following definition of cuts:

▶ **Definition 10** (Vertex Cuts). *For two arbitrary nodes $v, w \in V$, a (v,w)-*cut *is a subset $S \subseteq V$ such that all paths from v to w in G contain at least one node in S. The set of all (v,w)-cuts is denoted by $\mathcal{S}_{v,w}$.*

Using this, we can reformulate the requirement that a node v must be connected to its assigned center c by ensuring that for any (v,c)-cut $S \in \mathcal{S}_{v,c}$, at least one node in S is also assigned to c. This gives us the following LP-formulation of the non-disjoint connected k-median problem:

$$\begin{aligned}
\min \quad & \sum_{c \in C, v \in V} d(v,c) x_v^c & & \\
\text{s.t.} \quad & \sum_{c \in C} x_v^c \geq 1, & & \forall v \in V, \\
& \sum_{v' \in S} x_{v'}^c \geq x_v^c, & & \forall v \in V, c \in C, S \in \mathcal{S}_{v,c}, \\
& x_v^c \in \{0,1\}, & & \forall c \in C, v \in V,
\end{aligned} \tag{1}$$

where for all $v \in V$, $c \in C$ the variable x_v^c indicates whether v is assigned to c ($x_v^c = 1$) or not ($x_v^c = 0$). While the first constraint ensures, that each point v must belong to at least one cluster, the second constraint ensures that the clusters are connected, i.e., if v is connected to center c, then in each (v,c)-cut, there must be at least one point assigned to c. To relax this problem we drop the integrality constraint and only require that $x_v^c \geq 0$ for all $v \in V$, $c \in C$.

It is worth noting that the linear program may have exponential size, as there can be exponentially many (v,c)-cuts for a given pair v, c. However, since the separation problem can be solved in polynomial time and there also exists an alternative flow-based formulation of the LP with polynomial size [7], it remains possible to find the optimum fractional LP-solution in polynomial time.

Let $\tilde{x}$ be such an optimum fractional solution. By Markov's inequality it is easy to verify that for any $v \in V$ there exists at least one $c \in C$ such that $\tilde{x}_v^c \geq \frac{1}{k}$. By multiplying all values with k, we get a new LP solution x with $x_v^c := k\tilde{x}_v^c$ for all $v \in V, c \in C$ where each node is assigned to at least one center in its entirety. But even if for a $v \in V, c \in C$ it holds that $x_v^c \geq 1$ there may still exist (v,c)-cuts not containing any nodes fully assigned to c. To deal with this problem we need some results shown for node weighted Steiner trees.

▶ **Definition 11** (Node Weighted Steiner Tree Problem). *In an instance of the* node weighted Steiner tree problem *we are given a graph $G = (V,E)$, a set of terminal nodes $T \subseteq V$ and a weight function $w : V \to \mathbb{R}_{\geq 0}$. The goal is to find a subset $S \subseteq V$ such that $T \subseteq S$, the nodes of S form a connected subgraph of G, and the sum $\sum_{v \in S} w(v)$ is minimized.*

Similar as for the connected k-median problem there also exists an LP relaxation for the node weighted Steiner tree problem:

$$\begin{aligned} \min \quad & \sum_{v \in V} w(v) x_v \\ \text{s.t.} \quad & \sum_{v' \in S} x_{v'} \geq 1, \quad \forall u, v \in T, S \in \mathcal{S}_{u,v}, \\ & x_v^c \geq 0, \quad \forall v \in V, \end{aligned} \tag{2}$$

One might note that the assignment version of the connected k-median problem is closely related to the node weighted Steiner tree problem as one could reinterpret the problem as finding k node weighted Steiner trees such that each node is contained in at least one Steiner tree. The distances of the nodes to the respective centers can in this context be interpreted as k different node weight functions for the different trees. Using the solution x we will now define for any center $c \in C$ the set $T_c := \{c\} \cup \{v \mid x_v^c \geq 1\}$. Then the solution x can be subdivided into k solutions for the Steiner tree LP using the respective sets $(T_c)_{c \in C}$ as terminals.

▶ **Lemma 12.** *Let $(x_v^c)_{v \in V, c \in C}$ be a solution for LP (1). Then for any center $c \in C$ the values $(x_v^c)_{v \in V}$ form a solution for LP (2) with $T := T_c$ and $w(v) := d(v, c)$.*

Proof. This lemma is proved by contradiction. Assume that $(x_v^c)_{v \in V, c \in C}$ is a solution for LP (1) that violates some constraint of LP (2). Then there exist two terminals $u, v \in T_c$ and an (u, v)-cut S such that

$$\sum_{v' \in S} x_{v'}^c < 1.$$

Now we distinguish two cases. In the first case let c be one of the nodes $\{u, v\}$, w.l.o.g. $u = c$. Then we can use the definition of LP (1) and T_c to obtain

$$\sum_{v' \in S} x_{v'}^c \geq x_v^c = 1,$$

which contradicts the assumption.

In the second case both $u \neq c$ and $v \neq c$. Then by the previous argument S can neither separate u nor v from c because then its weight would be at least 1. Thus there exists a path from u to c and from c to v not containing any nodes in S. But then there also exists a path from u to v not using any nodes in S and S would not be a (u, v)-cut. ◀

This lemma is crucial because for the node weighted Steiner tree problem there exists an $O(\log n)$-approximation algorithm by Klein and Ravi [15] for which Guha et al. [11] later proved that it is implicitly a primal-dual algorithm and that the cost of the solution is at most $O(\log n)$ times worse than any fractional solution of LP (2). Alternatively Könemann et al provided a more detailed primal-dual algorithm for a generalized version of the node weighted Steiner tree problem ensuring the same guarantees [16]. Using these algorithms and Lemma 12 we are able to subdivide x into k Steiner tree LP solutions that can each be transformed into a feasible cluster.

▶ **Theorem 8.** *For the assignment version of the non-disjoint connected k-median problem, there is a polynomial-time algorithm with approximation factor $O(k \log n)$. This is true even if the distances are not a metric.*

Proof. In the first step of the algorithm we will first calculate an optimum solution $\tilde{x}$ for the relaxed version of LP (1) which as argued above can be done in polynomial time. Since every assignment solution for the connected k-median problem is also an LP solution, the costs of $\tilde{x}$ are upper bounded by the optimum value OPT. After multiplying the values by k, the cost of solution x is then again bounded by $k \cdot \mathrm{OPT}$.

Consider T_c as defined above, by Markov's inequality for every node v there exists a center c such that $\tilde{x}_v^c \geq \frac{1}{k}$ which means that $v \in T_c$. Thus we have $\bigcup_{c \in C} T_c = V$. By Lemma 12 the fractional solution x can be split up into k solutions of the Steiner tree LP (2) with the respective terminal sets $(T_c)_{c \in C}$ such that the total sum of the objective value is exactly the cost of x.

By applying the algorithm by Klein and Ravi [15] with $T = T_c$ and $w(v) = d(v, c)$ for every $c \in C$ separately, one obtains a Steiner tree S_c with $T_c \subseteq S_c$ whose cost is upper bounded by $O(\log n)$ times the cost of the respective LP solution. Thus one obtains in polynomial time k Steiner trees each connecting the respective terminal set T_c and total cost bounded by $O(\log n)$ times the objective value of x which results in an upper bound of $O\left(k \log n\right) \mathrm{OPT}$. Since every node is contained in at least one of these Steiner trees, this directly yields the desired solution for the assignment version of the non-disjoint connected k-median problem.

This proof works even if the distances are not a metric because it does not use any of their properties. ◀

One may note that this approach works with any approximation algorithm for the node weighted Steiner tree problem whose cost is bounded by the optimum LP solution. For some restricted graph classes, including planar graphs, Demaine et al. [5] provided a primal dual algorithm calculating a constant approximation. Thus the approach can also be used to get an $O(k)$-approximation (and $O(k^2)$ if the centers are not given) for these kind of connectivity graphs.

2.2 Finding Centers

In this section we give an overview how suitable centers for the non-disjoint connected k-median problem can be found. For a more detailed description of the procedure and its analysis we refer to the full version of this paper [7]. First we will again provide a suitable ILP formulation:

$$
\begin{aligned}
\min \quad & \sum_{v,c \in V} d(v,c) x_v^c & & \\
\text{s.t.} \quad & \sum_{c \in V} x_v^c \geq 1, & & \forall v \in V, \\
& \sum_{c \in V} x_c^c \leq k, & & \\
& \sum_{v' \in S} x_{v'}^c \geq x_v^c, & & \forall v, c \in V, S \in \mathcal{S}_{v,c}, \\
& x_v^c \in \{0,1\}, & & \forall v, c \in V,
\end{aligned} \tag{3}
$$

where $\mathcal{S}_{v,c}$ denotes the set of vertex cuts, separating v and c, defined as in the previous section.

The main difference to the assignment variant is that we do now allow to assign nodes to any other nodes and not only to some predefined centers. However, by definition of $\mathcal{S}_{v,c}$, it holds that $\{c\} \in \mathcal{S}_{v,c}$ for any $v, c \in V$. Thus x_v^c will automatically be limited by x_c^c. This naturally leads to the property that nodes can only be assigned to c if $x_c^c = 1$. Thus limiting $\sum_{c \in V} x_c^c$ by k directly ensures that any solution of the ILP uses at most k centers.

If we relax the integrality constraint and only require that $x_v^c \geq 0$ for all $v, c \in V$ also LP (3) can be solved in polynomial time despite its exponential size. The drawback of this is that for arbitrarily many nodes $c' \in V$ it can hold that $x_{c'}^{c'} > 0$ if the respective values are sufficiently small. This means that other nodes can at least partially be assigned to them and we will call the value $x_{c'}^{c'}$ the *opening* of c'.

The main focus of this section is to move these openings to a limited set of k centers that will be determined by our algorithm such that reassigning the nodes accordingly is not too expensive. To do this, we follow a similar approach as the LP rounding algorithm for k-median by Charikar et al. [3]. In the first step, we will find a solution in which every node is opened either by at least $\frac{1}{2}$ or not opened at all (limiting the number of centers to $2k$). In the latter steps these openings will then be made integral by first finding possible replacements for some of the centers, which possibly allows us to close the respective centers, and then choosing a subset of the centers of size k such that for every center itself or its replacement is contained in this subset.

We will relax the connectivity constraint during the algorithm. We say for an $\alpha \in [0, 1]$ that an assignment $(x_v^c)_{v,c \in V}$ is an α-connected solution if $\sum_{c \in V} x_c^c \leq k$, $\forall v \in V : \sum_{c \in V} x_v^c \geq 1$ and for all $v, c \in V$ and all cuts $S \in \mathcal{S}_{v,c}$ separating v and c it holds that $\sum_{v' \in S} x_{v'}^c \geq \min(\alpha, x_v^c)$. To simplify notation, we will denote the weight of the minimum cut between two nodes v and c (with respect to the values $(x_{v'}^c)_{v' \in V}$) by $\mathrm{sep}^x(v, c) = \min_{S \in \mathcal{S}_{v,c}} \sum_{v' \in S} x_{v'}^c$. Then the last requirement can be reformulated as $\mathrm{sep}^x(v, c) \geq \min(\alpha, x_v^c)$. It is easy to verify that an assignment corresponds to a fractional solution for LP (3) if and only if it is 1-connected.

Now we give an overview over the different steps of the algorithm. Our main focus will lie on the first step, which ensures that all centers will be half opened afterwards. This is due to the fact that it introduces the largest amount of novel ideas and necessary modifications compared to the existing algorithm by Charikar et al.:

Step 1. To obtain half-opened centers, let an optimal (fractional) LP solution $\tilde{x}$ be given. For each v, let its mass be defined as $m_v = \sum_{c \in V} \tilde{x}_v^c$, its cost (or price) as $p_v = \sum_{c \in V} \tilde{x}_v^c \cdot d(v, c)$ and its (average) radius as $r_v = \frac{p_v}{m_v}$. We say that an assignment of a node v to another node c is *good* if $d(v, c) \leq 4r_v$. Since $m_v \geq 1$ for all nodes $v \in V$, we may observe that because of Markov's inequality at least $\frac{3}{4}$ of v gets assigned via good assignments, i.e., $\sum_{c \in V : d(v,c) \leq 4r_v} \tilde{x}_v^c \geq \frac{3}{4}$.

We will simultaneously maintain two assignments $(x_v^c)_{v,c \in V}$ and $(y_v^c)_{v,c \in V}$ such that for all $v, c \in V$ initially $x_v^c = \tilde{x}_v^c$ and $y_v^c = 0$ and that at the end of the algorithm $x_v^c = 0$. The final values of y_v^c will be the assignment using half opened centers. During the algorithm, the combination of x and y will always be a $\frac{1}{8k}$-connected solution, which means that the following inequalities stay satisfied throughout the entire process:

$$\sum_{c \in V} (x_v^c + y_v^c) \geq 1 \qquad \forall v \in V \tag{4}$$

$$\sum_{c \in V} (x_c^c + y_c^c) \leq k \tag{5}$$

$$\sum_{v' \in S} x_{v'}^c \geq x_v^c \qquad \forall v, c \in V, S \in \mathcal{S}_{v,c} \tag{6}$$

$$\sum_{v' \in S} y_{v'}^c \geq \min\left(\frac{1}{8k}, y_v^c\right) \qquad \forall v, c \in V, S \in \mathcal{S}_{v,c} \tag{7}$$

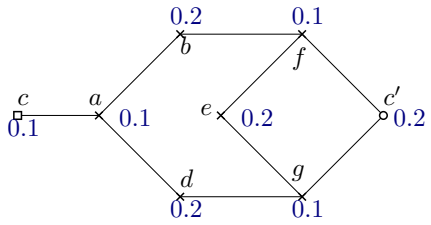


Figure 1 A sketch depicting a situation in which, at least if we require 1-connectivity, the total assignment needs to increase if we apply a shift from c' to c. The blue values denote the assignments of the respective nodes to c' before the shift, i.e., the $x_v^{v'}$-values. We assume that $x_v^c = 0$ for all nodes v. Now if we apply a shift from c' to c by $x_c^{c'} = 0.1$, both c' and c have an opening of 0.1. Thus the nodes b, d, e all need to be assigned to c and c' with value 0.1. If we would try to find assignments of a, f and g such that for each node its total assignment to c and c' is exactly 0.1 we would need to set $x_g^{c'} = x_f^{c'} = 0.1$ to ensure the connectivity of b and d to c'. However this would mean that $y_f^c = y_g^c = 0$ which would separate e from c. Thus to ensure connectivity, for at least one node the total assignment after the shift needs to be larger than the assignment before the shift.

The algorithm will now maintain an initially empty center set C and repeat in an alternating order the following two steps: First it decides for an additional node whether it gets added to the center set or not (in the latter case it will be added to a so called environment of an already existing center). Afterwards it shifts some openings in x (the x_c^c that are greater than 0) to already chosen centers c in C.

To get an intuition regarding the shifts, it can be beneficial to sometimes think about the connectivity constraints not from a cut-perspective but to rather interpret the assignments as flows. Using the max-flow min-cut theorem, it is easy to see that if we interpret the values $(x_{v'}^{c'})_{v' \in V}$ as node capacities then there exists a flow from a node v to a node c' of value f exactly if $\text{sep}^x(v, c') \geq f$. Thus our connectivity constraint implies that there always exists a flow from v to c' of value $x_v^{c'}$. By basically reverting this flow for a center $c \in C$ and a node $c' \in V$ we are able to shift an opening of value $x_c^{c'}$ from c' to c. One important question is now how the nodes that were previously assigned to c' get assigned. If $x_c^{c'}$ was equal to $x_{c'}^{c'}$, the entire opening of c' can be shifted and we can simply reassign all nodes to c by setting $y_v^c = y_v^c + x_v^{c'}$ and $x_v^{c'} = 0$ for any $v \in V$. However, if $x_c^{c'}$ was smaller than $x_{c'}^{c'}$ only a fraction of the opening of c' gets shifted and we basically have to split up the previous assignments to c' between c and c'. As Figure 1 shows this can lead to situations in which the total assignment to c and c' after this shift actually needs to be larger than the assignment before this shift. Since it is actually rather involved to determine the new values in this situation, a formal description will only be provided in the full version of this work. To get an intuition regarding the algorithm it is for now sufficient to only consider the first case.

Since either way also the assignments of nodes assigned to c' have to be shifted in this situation, we have to make sure that the distances of the respective shifts are not too large which the algorithm will ensure by only allowing shifts with a rather small distance between c and c' in the beginning and increasing this distance over time. More details will follow later.

While the shifts depend on the chosen centers C, the choice whether a node v will be added to C in turn depends on the shifts that have already been performed before v gets considered by the algorithm. The algorithm terminates once it has decided for every node whether it gets added to C and all openings in x have been shifted.

To determine which nodes should be added to C, we iterate over all nodes ordered by their radii in increasing order and check whether we can open them by at least $\frac{1}{2}$. To be more precise, a node v will be chosen as a center if the sum of the remaining good assignments $(x_v^{c'})_{c' \in V}$ is at least $\frac{1}{2}$. The construction of the algorithm will ensure that it will shift at least

all remaining openings belonging to good assignments of v to v itself which means that v will be at least half opened. When v is added to the center set we also create its environment $S_v = \{v\}$, which initially only contains v itself.

In the case that v does not get chosen as a center, we know that there were at least $\frac{1}{4}$ of good assignments of v that have already been shifted to other centers in C. Since every center in C gets at least half opened and the entire opening is limited by k, we know that $|C| \leq 2k$ and thus there exists at least one center to which $\frac{1}{8k}$ of the good assignments of v have been shifted. One critical property of the algorithm will be that before v gets considered, the distance between the source c' and target c of any shift will be at most $4r_v$. Thus for any good assignment to a node z whose opening has been shifted to a center c we have that $d(v,c) \leq d(v,z) + d(z,c) \leq 8r_v$. As a result we may conclude that there exists a center $c \in C$ such that $y_v^c \geq \frac{1}{8k}$ while $d(v,c) \leq 8r_v$ and we add v to the environment S_c of the closest center c fulfilling these requirements.

Adding v to this environment allows us to shift some openings of a node c' with $x_v^{c'} > 0$ over the node v to c even if there does not exist a direct flow from c to c' (i.e., c is not assigned to c'). The intuition behind this is that we first revert the flow from v to c' and thus are already shifting the opening to v. Now by combining $y_v^c \geq \frac{1}{8k}$ and Property (7) we know that already before this shift the minimum cut between v and c has value at least $\frac{1}{8k}$. Thus if we are moving the opening (and the assignments to it) to c instead of v itself we know that the solution still stays $\frac{1}{8k}$-connected. Furthermore since v and c are not too far apart, we know that the distances of these shifts does not increase too much compared to a direct shift to v. This will help us ensuring that if the openings of some nodes, to which v got assigned, get shifted to some other centers then the distance added by this shift will not be too large.

Every time after a new node v gets added to either C or to an environment S_c of a center $c \in C$ we will go over all possible shifts ordered by their distances in increasing order, and perform all shifts with distance at most $4r_{\text{next}}$, where r_{next} denotes the radius of the next node that will be considered by the algorithm. Note that any shift with radius $\leq 4_{r_v}$ can only result from the addition of v to S_c or C because any shift not involving v would already have been applied in the last iteration (as r_v is the previous value of r_{next}). As a result all those shifts will be performed before any shift not involving v will be executed. This directly guarantees that if v is added to C then all openings belonging to good assignments will be shifted to v, which ensures that v is half opened.

An important property of the shifts is that if we shift some opening of a node $c' \in V$ over a node $v \in V$ then afterwards $x_v^{c'} = 0$ which ensures that for all $v, c' \in V$ we will only shift some opening of c' over v at most once, limiting the number of shifts by $|V|^2$. Additionally after we have added v to C or a set S_c there always exists a possible shift while $x_v^{c'} > 0$ for a $c' \in V$. Thus, it is possible to execute the described algorithm for a polynomial time until for every $v, c' \in V$, $x_v^{c'} = 0$ and then return $(y_v^c)_{v \in V, c \in C}$. A pseudocode summarizing the structure of the described procedure can be found in Algorithm 1. The cost of the resulting $\frac{1}{8k}$-connected solution is bounded by $O(k)$ times the cost of the LP solution $\tilde{x}$.

Step 2. Now we would like to reduce the number of centers to at most k. However, if we want to reduce the opening of a center $c \in C$ to 0, we will need to find a suitable replacement, i.e., another center $\tilde{c}$ such that reassigning all nodes assigned to c to $\tilde{c}$ neither costs too much nor violates our connectivity constraints (which is further relaxed to $\frac{1}{16k}$-connectivity). To do this we will need two properties of Algorithm 1:

- For any node v we know that if a shift from another node c' to a center c caused v to be (partially) assigned to c then the radius r_c of c can be bounded by the radius r_v of v and the distance from v to c'. To be more precise $r_c \leq d(v,c') + 2r_v$. Intuitively, if this

Algorithm 1 The general structure of the first step.

```
Order v_1, ..., v_n according to their radii (r_v)_{v∈V} in increasing order
for v_1, ..., v_n do
    Decide whether v_i is added to the center set C or to an environment S_c of a
     center c ∈ C
    while A shift from a node v to a center c ∈ C with d(v, c) ≤ r_{v_{i+1}} is possible do
        Apply the cheapest shift
    end
end
```

was not the case, then the algorithm would have shifted some opening of c' either to v itself (if v gets chosen as a center) or over v to another center $\tilde{c} \in C$ with $v \in S_{\tilde{c}}$ before c was even added to the center set which would have prevented v from being assigned to c. The main consequence of this property is that if we can find another center $\tilde{c}$ that is only a constant times r_c away from c then reassigning all nodes that are currently assigned to c, to $\tilde{c}$ instead costs not too much.

- Additionally for any center c it holds that if c got assigned to another center $\tilde{c}$ because some opening got shifted from a node c' to c then it holds that $d(c, \tilde{c})$ is bounded by $d(c, c') + \max(4r_c, d(c, c'))$ because otherwise a shift from c' to c would have happened before the shift to $\tilde{c}$. Since the average length of the original assignments of c was r_c, one can use this to identify a center $\tilde{c}$ that is not too far apart from c.

Using these two properties, we are able to split up C into two sets C_1 and $C_{1/2}$ such that for every center $c \in C_{1/2}$ we have a successor $s(c) \in C$ such that the total cost that occurs if we replace any center in $C_{1/2}$ by its successor would not be too large and $|C_1| + \frac{1}{2}|C_{1/2}| \leq k$. Later we will then open all nodes in C_1 as well as at most half the nodes in $C_{1/2}$ to obtain a integral center set C^* such that for all nodes in $C_{1/2}$ either the node itself or its successor is contained in C^*.

To obtain these sets, we consider an arbitrary center $c \in C$ and define $a_c = 1 - y_c^c$. Then at least a total value of a_c of the openings corresponding to the initial assignments of c (the values $\tilde{x}_c^{c'}$ for any $c' \in V$) were shifted to other centers. If a_c has value at least $\frac{1}{4}$, we are able to use Markov's inequality together with the second property to identify another center $\tilde{c}$ such that $d(c, \tilde{c})$ lies within $O(r_c)$ and c is already sufficiently connected to $\tilde{c}$ (i.e., $y_c^{\tilde{c}} \geq \frac{1}{16k}$). We thus can set $s(c) = \tilde{c}$ and add c to $C_{1/2}$.

However, if a_c is rather small then it can be that all assignments of c to other centers $\tilde{c}$ only resulted from shifts of nodes c' to $\tilde{c}$ that are really far apart from c which means that $\tilde{c}$ can be even further away. Additionally the assignments $y_c^{\tilde{c}}$ might also be quite small which means that we would need to increase some existing assignments to $\tilde{c}$ to ensure $\frac{1}{16k}$-connectivity if we replace c by $\tilde{c}$. However, if there are multiple centers where the respective value a_c is small then we know that since $\sum_{c \in C}(1 - a_c) = \sum_{c \in C} y_c^c \leq k$ that if we add one of these centers to $C_{1/2}$ we can add multiple others to C_1. Since we do not need a replacement for the centers in C_1, we can use a careful potential argument to redistribute parts of the replacement cost of the centers added to $C_{1/2}$ to the centers in C_1 to bound the additional cost among all nodes.

Step 3. To decide which centers in $C_{1/2}$ can be closed we have a look at the directed Graph $G_{1/2} = (C_{1/2}, S)$, where $S = \{(c, s(c)) \mid c, s(c) \in C_{1/2}\}$ contains for any center in $C_{1/2}$ an edge to its successor if the latter is again contained in $C_{1/2}$. If this graph is bipartite we can simply take the smaller half of this partition and add this to the center set C^*. Then for every

edge one of its two endpoints is contained in C^* which ensures that for every center either itself or its successor is contained in C^*. However, the graph might not directly be bipartite because it can contain cycles of uneven length. But we can ensure by being a little bit more careful in step 2 that any of these cycles contains at most one center c with $a_c < \frac{1}{4}$. Using this, we can remove the respective cycles and thus obtain a bipartite graph. For all centers in $C_{1/2}$ not added to C^*, we reassign all nodes assigned to them to the respective successor $s(c)$. Since we bounded the cost of replacing all centers in $C_{1/2}$ and are only replacing a subset of them, the total cost of this is again bounded. The entire cost still stays in $O(k)$ times the cost of the fractional LP solution $\tilde{x}$.

Step 4. After step 3, we have an $\frac{1}{16k}$-connected solution only using integral centers. However the assignments might still be fractional. By multiplying the assignments by $16k$ the solution becomes 1-connected and similarly as for the assignment version in Section 2.1 we obtain that for every node v there exists at least one center c such that $x_v^c \geq 1$. Again applying the primal dual algorithm for node weighted Steiner trees, we obtain an entirely integral solution:

▶ **Theorem 9.** *For the non-disjoint connected k-median problem, there is a polynomial-time algorithm with approximation factor $O(k^2 \log n)$.*

References

1. Ioana Oriana Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R. Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. In Dimitris Achlioptas and László A. Végh, editors, *APPROX/RANDOM 2019*, volume 145 of *LIPIcs*, pages 18:1–18:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.18.
2. Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Transactions on Algorithms (TALG)*, 13(2):1–31, 2017. doi:10.1145/2981561.
3. Moses Charikar, Sudipto Guha, Éva Tardos, and David B Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 1–10, 1999.
4. Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. *Advances in neural information processing systems*, 30, 2017.
5. Erik D Demaine, MohammadTaghi Hajiaghayi, and Philip N Klein. Node-weighted steiner tree and group steiner tree in planar graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 328–340. Springer, 2009. doi:10.1007/978-3-642-02927-1_28.
6. Lukas Drexler, Jan Eube, Kelin Luo, Dorian Reineccius, Heiko Röglin, Melanie Schmidt, and Julian Wargalla. Connected k-center and k-diameter clustering. *Algorithmica*, September 2024. doi:10.1007/s00453-024-01266-9.
7. Jan Eube, Kelin Luo, Dorian Reineccius, Heiko Röglin, and Melanie Schmidt. Connected k-median with disjoint and non-disjoint clusters, 2025. arXiv:2507.02774.
8. Rong Ge, Martin Ester, Byron J Gao, Zengjian Hu, Binay Bhattacharya, and Boaz Ben-Moshe. Joint cluster analysis of attribute data and relationship data: The connected k-center problem, algorithms and applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(2):1–35, 2008. doi:10.1145/1376815.1376816.
9. Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.
10. Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. In Howard J. Karloff, editor, *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California, USA*, pages 649–657. ACM/SIAM, 1998. URL: http://dl.acm.org/citation.cfm?id=314613.315037.

11 Sudipto Guha, Anna Moss, Joseph (Seffi) Naor, and Baruch Schieber. Efficient recovery from power outage (extended abstract). In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99, pages 574–582, New York, NY, USA, 1999. Association for Computing Machinery. doi:10.1145/301250.301406.

12 Neelima Gupta, Aditya Pancholi, and Yogish Sabharwal. Clustering with internal connectedness. In *WALCOM: Algorithms and Computation: 5th International Workshop, WALCOM 2011, New Delhi, India, February 18-20, 2011. Proceedings 5*, pages 158–169. Springer, 2011. doi:10.1007/978-3-642-19094-0_17.

13 Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Math. Oper. Res.*, 10(2):180–184, 1985. doi:10.1287/moor.10.2.180.

14 Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discret. Appl. Math.*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.

15 P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *Journal of Algorithms*, 19(1):104–115, 1995. doi:10.1006/jagm.1995.1029.

16 Jochen Könemann, Sina Sadeghian, and Laura Sanita. An lmp o (log n)-approximation algorithm for node weighted prize collecting steiner tree. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 568–577. IEEE, 2013.

17 Shi Li. Approximating capacitated k-median with (1+ϵ) k open facilities. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 786–796. SIAM, 2016.

18 Shi Li and Xiangyu Guo. Distributed k-clustering for data with heavy noise. *Advances in Neural Information Processing Systems*, 31, 2018.

19 Zhung-Xun Liao and Wen-Chih Peng. Clustering spatial data with a geographic constraint: exploring local search. *Knowl. Inf. Syst.*, 31(1):153–170, 2012. doi:10.1007/s10115-011-0402-8.

20 Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k-means. In *Approximation and Online Algorithms: 17th International Workshop, WAOA 2019, Munich, Germany, September 12–13, 2019, Revised Selected Papers 17*, pages 232–251. Springer, 2020. doi:10.1007/978-3-030-39479-0_16.

21 Ali Vakilian and Mustafa Yalciner. Improved approximation algorithms for individually fair clustering. In *International Conference on Artificial Intelligence and Statistics*, pages 8758–8779. PMLR, 2022.

22 Hamidreza Validi, Austin Buchanan, and Eugene Lykhovyd. Imposing contiguity constraints in political districting models. *Oper. Res.*, 70(2):867–892, 2022. doi:10.1287/opre.2021.2141.

A Dynamic Piecewise-Linear Geometric Index with Worst-Case Guarantees

Emil Toftegaard Gæde
Technical University of Denmark, Lyngby, Denmark

Ivor van der Hoog
IT University of Copenhagen, Denmark

Eva Rotenberg
IT University of Copenhagen, Denmark

Tord Stordalen
Technical University of Denmark, Lyngby, Denmark

Abstract

Indexing data is a fundamental problem in computer science. The input is a set S of n distinct integers from a universe $\mathcal{U}$. Indexing queries take a value $q \in \mathcal{U}$ and return the `membership`, `predecessor` or `rank` of q in S. A `range` query takes two values $q, r \in \mathcal{U}$ and returns the set $S \cap [q, r]$.

Recently, various papers study a special case where the the input data behaves in an approximately piece-wise linear way. Given the sorted (rank,value) pairs, and given some constant ε, one wants to maintain a small number of axis-disjoint line-segments such that, for each rank, the value is within $\pm\varepsilon$ of the corresponding line-segment. Ferragina and Vinciguerra (VLDB 2020) observe that this geometric problem is useful for solving indexing problems, particularly when the number of line-segments is small compared to the size of the dataset.

We study the dynamic version of this geometric problem. In the dynamic setting, inserting or deleting just one data point may cause up to three line-segments to be merged, or one line-segment to be split at most three-way. To determine and compute this, we use techniques from dynamic maintenance of convex hulls, and provide new algorithms with worst-case guarantees, including an $O(\log n)$ algorithm to compute a separating line between two non-intersecting convex hulls – an operation previously missing from the literature.

We then use our fully-dynamic geometry-based subroutine in an indexing data structure, combining it with a natural hashing technique. The resulting indexing data structure has theoretically efficient worst-case guarantees in expectation. We compare its practical performance to the solution of Ferragina and Vinciguerra, which was shown to perform better in certain structured settings [Sun, Zhou, Li VLDB 2023]. Our empirical analysis shows that our solution supports more efficient range queries in the special case where the update sequence contains many deletions.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Algorithms Engineering, Data Structures, Indexing, Convex Hulls

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.64

Related Version *Full Version*: `https://arxiv.org/abs/2503.05007`

Supplementary Material
Software (Source Code): `https://github.com/Sgelet/DynamicLearnedIndex` [18]
archived at `swh:1:dir:b8763eb0504d33beb81ee89d230a30dca8ab0b66`
Software (Test Bed): `https://github.com/Sgelet/LearnedIndexBench` [19]
archived at `swh:1:dir:07ea25cfc176438933c1a5507bfdad3ba9461ab6`

Funding This work was supported by the Carlsberg Foundation Fellowship CF21-0302 "Graph Algorithms with Geometric Applications", the VILLUM Foundation grant (VIL37507) "Efficient Recomputations for Changeful Problems", and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987.

Acknowledgements We thank Linda Kleist for reading this paper and correcting our constants.



© Emil Toftegaard Gæde, Ivor van der Hoog, Eva Rotenberg, and Tord Stordalen;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

We investigate the use of *learned indices* for the design of *dynamic indexing data structures*.

Indexing data structures. An indexing data structure maintains a set S of n distinct integers from a universe $\mathcal{U}$. Let $\text{RANK} : S \to [n]$ be the function mapping each $s \in S$ to its index in the sorted order of S. The objective is to support the following indexing queries:

- `member(`q`)` returns `true` if $q \in S$.
- `predecessor(`q`)` returns $\max\{t \in S \mid t < q\}$. (We allow $q \notin S$.)
- `rank(`q`)` returns $\text{RANK}(\texttt{predecessor}(q)) + 1$. (We allow $q \notin S$.)

Additionally, we consider *range queries*, where k denotes the output size:

- `range(`q`, `t`)` returns $S \cap [q, t]$. (We allow $q, t \notin S$.)

Static indexing data structures fall into three broad categories: *Tree-based solutions* store S in a sorted array A, requiring no additional space but incurring logarithmic query costs. Tree traversals enable `predecessor`, `rank`, and `member` queries in $O(\log n)$ time and range queries in $O(\log n + k)$ time [1, 3, 32, 37]. *Map-based solutions* store S in sorted order and maintain a hash map $H : S \to [n]$ mapping each element to its rank [4, 23, 31]. This enables constant-time support for `member`, `predecessor`, and `rank` queries if queries are restricted to elements of S, and $O(k)$ time for `range` queries when both endpoints lie in S. The additional space is $O(n)$. A third category use what are called *learned indices*, a recently introduced term [24, 20, 16, 22, 8]. Given an integer parameter ε, a learned index is a function $h_\varepsilon : \mathcal{U} \to [0, n]$ such that

$$h_\varepsilon(q) \in [\texttt{rank}(q) - \varepsilon, \texttt{rank}(q) + \varepsilon].$$

The function h_ε is learned from S and used to guide search in a sorted array A storing S. Ferragina and Vinciguerra [16] interpret h_ε geometrically: each $s \in S$ maps to a point $(\text{RANK}(s), s)$ in the plane, and h_ε is learned as a piecewise-linear approximation to F_S.

A notable instance of learned indices is the *PGM index* [16], where h_ε is a y-monotone piecewise-linear function made of segments, with the property that each point in F_S lies within an ε-wide horizontal strip around some segment. Let $|h_\varepsilon|$ denote the number of segments. The data structure supports indexing queries in $O(\varepsilon + \log |h_\varepsilon|)$ time and range queries in $O(\varepsilon + k + \log |h_\varepsilon|)$ time. They also show how to construct a PGM index in linear time, such that there exists no PGM index h'_ε with $|h_\varepsilon| > 2|h'_\varepsilon|$. Ferragina and Vinciguerra argue that learned indices are the "best of both worlds" since:

- the supported queries are as general as those supported by tree-based solutions,
- the solution uses only $O(|h_\varepsilon|)$ additional space, and
- $O(\varepsilon + \log |h_\varepsilon|)$ is, for an appropriate choice of ε, efficient in practice.

Their performance has been empirically benchmarked in several studies [15, 24, 38, 37].

Dynamic indexing data structures. Due to their fundamental role, dynamic indexing structures have received extensive theoretical and practical attention. When S is dynamic, maintaining a sorted array becomes inefficient. Tree-based structures can be updated in $O(\log n)$ time with tree rotations. Map-based approaches allow constant-time `member` updates but are typically not extended to support other indexing queries. Learned indices offer a promising direction by exploiting structural properties of S, akin to parametrised algorithms. However, just as parametrised algorithms, data structures based on learned indices are not always efficient: if S lacks exploitable structure or access patterns are skewed, traditional

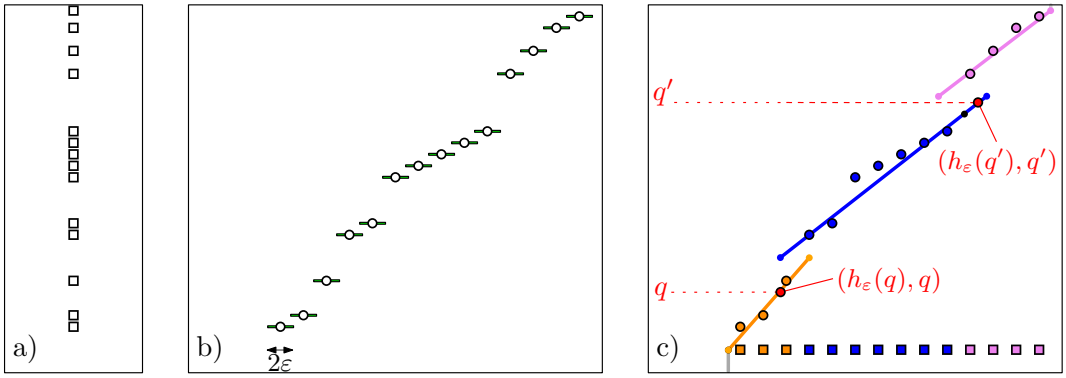


Figure 1 (a) a set of n values S. (b) S corresponds to an xy-monotone point set F_S. (c) The PGM index computes a y-monotone set of segments that starts and ends with a vertical halfline.

indexing data structures are preferred [33, 26]. Experimental studies have examined properties of learned indices [14, 13, 26, 33], in an effort to classify when they are appropriate to use. As a brief summary, learned indices pay a price in updates [26], and traditional indices are preferred if S or the access pattern is complex or skewed, or if concurrency is possible [33]. If there is sufficient structure, both space usage and access times can benefit from learned indices [13, 14], subject to the strategy employed by the learned index.

Dynamic learned indices through the logarithmic method. The logarithmic method of Overmars [29] provides an amortised way to maintain learned indices. S is partitioned into $\lceil \log n \rceil$ buckets B_i, each of size 2^i. Each bucket is either full or empty, and stores its contents in an array A_i in sorted order and maintains a learned index h^i_ε over A_i.

Let the learned index h_ε have a construction time of $T(n)$. This data structure can be maintained insertion-only in amortised $O(T(n) \log n)$ time. An insertion inserts a new value into B_0. Let j be the maximum integer such that all B_i for $i \in [0, j-1]$ are full. This approach empties these buckets, fills B_j in sorted order, and constructs (A_j, h^j_ε) in $O(T(2^j))$ time. Whenever we delete some $s \in S$, this approach instead inserts a *tombstone* s^*, which is a special copy of s. If an insertion fills a new bucket B_j, it first iterates over all elements. If B_j contains both s and s^*, it removes both elements. It then constructs (A_j, h^j_ε) twice. Once on all "normal" values, and once on all tombstones in B_j. This way, deletions take the same time as insertions do. In this paper, we consider the following open question:

"Can a learned index be dynamically be maintained with worst-case guarantees?"

Intermezzo: computing a line cover. The interpretation of learned indices by Ferragina and Vinciguerra [16] translates to a geometric problem where the goal is to (approximately) cover a monotone set of two-dimensional points by a set of line segments. Using the logarithmic method, and the static algorithm of O'Rourke [27] to approximately points, they dynamically maintain an ε-*cover*: a set of lines that are guaranteed to be within an ε horizontal distance from each point. We consider the problem of maintaining dynamic ε-covers to be an interesting geometric problem in its own right.

From learned indices to indexing data structures. Under the logarithmic method, indexing queries decompose naturally across the buckets:

- For `member`(q), $q \in S$ if and only if there exists an $i \in [\lceil \log n \rceil]$ with $q \in B_i$.
- For `predecessor`(q), the output is the maximum predecessor across B_i for $i \in [\lceil \log n \rceil]$.

- For `rank`(q), the rank is the sum of all ranks of q in B_i for $i \in [\lceil \log n \rceil]$.
- For `range`(q, t), the reported range is the union of all ranges in B_i for $i \in [\lceil \log n \rceil]$.

This way, indexing queries require only an additional factor $O(\log n)$ time. Indexing queries can be answered by combining queries to both the normal and tombstone structures. E.g., `rank`(q) is the rank of q in the "normal" data structure minus the rank of q in the tombstone structure. This approach has two downsides:

- First, approach has an amortised update time.
- Second, this approach does not support output-sensitive range queries – as there may be $O(n)$ values s that (together with their tombstones s^*) lie in between a query pair (q, t).

This leads to the following open question:

> "Can a dynamic learned index be converted into an output-sensitive dynamic indexing data structure with worst-case guarantees?"

Contribution and organization. We propose maintaining a dynamic ε-cover (and thereby a dynamic learned index) via dynamic convex hull techniques. Section 3 shows that deciding whether S admits an ε-cover of complexity 1 is equivalent to convex hull intersection testing (Figure 2). Section 4 shows a robust algorithm to compute the intersection between two convex hulls in $O(\log n)$ time. We adapt our algorithm to output a separating line (which is a learned index of complexity 1) in the negative case. Section 5 combines these with dynamic convex hull data structures to yield a dynamic learned index worst-case $O(\log^2 n)$ update time. We empirically compare our learned index to the PGM index from [16].

Section 6 introduces a novel hashing-based approach to convert learned indices into dynamic indexing data structures, using $O(\varepsilon^{-1})$ additional expected overhead. We compare our dynamic indexing structure to the amortised PGM index of [16] in terms of update time and index complexity. We do not benchmark against traditional indexing data structures – since the relation between learned and traditional indices is previously studied [33]. Instead, our goal is to push the theoretical limits and worst-case guarantees of learned indices.

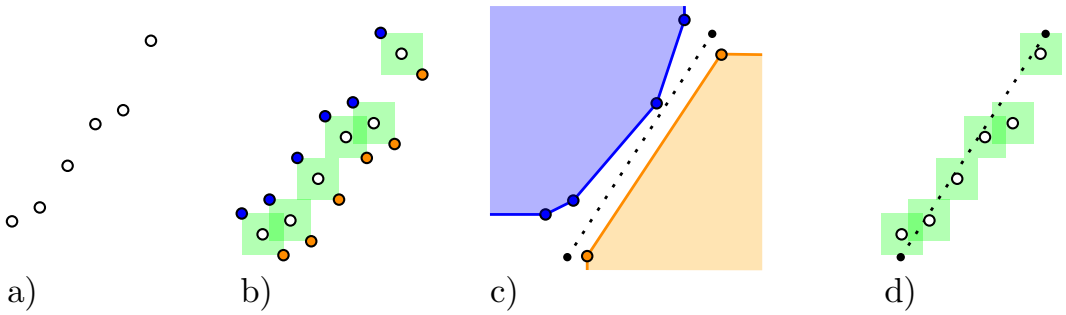


Figure 2 For any set F_S, we construct two convex hulls. We prove that there exists a segment ℓ within L_∞-distance ε of all points in F_S if and only if these hulls do not intersect. We adapt the convex hull intersection testing algorithm to find ℓ whenever these hulls are disjoint.

2 Preliminaries

The input is a dynamic set S of n distinct positive integers from some universe $\mathcal{U}$. For $a, b \in \mathbb{Z}$ with $a \leq b$, we define $S[a, b]$ as the set $S \cap [a, b]$. We denote by F_S the two-dimensional point set obtained by mapping each $s \in S$ to $(\text{RANK}(S), s)$. Throughout this paper, we distinguish between positions and strict positions. E.g., lying above or strictly above a line.

▶ **Definition 1** ([16]). *Let ε be a positive integer. A* PGM index h_ε *of S is defined as a y-monotone set of segments that together cover the y-axis. We regard h_ε as a map from y-coordinates to x-coordinates and require that for all $q \in \mathcal{U}$, $h_\varepsilon(q) \in [\texttt{rank}(q)-\varepsilon, \texttt{rank}(q)+\varepsilon]$.*

Ferragina and Vinciguerra [16, Lemma 1] wrongfully claim an $O(n)$-time algorithm to compute a minimum complexity PGM index h_ε. They invoke a streaming algorithm by O'Rourke [27] for fitting straight lines through data ranges. We show that this algorithm outputs a PGM index h_ε such that there exists no PGM index h'_ε with $|h_\varepsilon| > 2h'_\varepsilon$ (see the full version for details). Their algorithm restricts S to contain no duplicates. We assume the same setting and compute something slightly different as we define an ε-cover instead:

▶ **Definition 2.** *Let ε be a positive integer. We define an ε-cover f of S as a set of vertically separated segments with slope at least 1 where all $(r, s) \in F_S$ are within L_∞-distance ε of f.*

An ε-cover has a functionality and complexity similar to a learned index:

▶ **Observation 3.** *Let f be an ε-cover and Q be a horizontal line with height $q \in [\min S, \max S]$. Let (s, t) be the segment in f closest to q. Then $(line(s, t) \cap Q).x \in [\texttt{rank}(q)-2\varepsilon, \texttt{rank}(q)+2\varepsilon]$.*

▶ **Observation 4.** *For fixed ε, let k denote the minimum complexity of any PGM index of S. If f is an ε-cover of S of minimum complexity, then f contains at most $k-2$ edges.*

▶ **Definition 5.** *For any fixed ε-cover f of S, we define $\Lambda(f)$ as the set of pairwise interior-disjoint one-dimensional intervals that correspond to the y-coordinates of segments in f.*

Dynamic convex hulls. We dynamically maintain an ε-cover f of S of approximately minimum complexity. To this end, we use a result by Overmars and van Leeuwen [30] to dynamically maintain for all $[a, b] \in \Lambda(f)$ the convex hull of $F_{S[a,b]}$. For any point set F, denote by $CH(F)$ their convex hull. The data structure in [30] is a balanced binary tree over F, which at its root maintains a balanced binary tree over the edges $CH(F)$ in their cyclical ordering. It uses $O(n)$ space and has worst-case $O(\log^2 n)$ update time.

Rank-based convex hulls. For any update in S, up to n values in F_S may change their x-coordinate. This complicates the maintenance of a dynamic data structure over F. Gæde, Gørtz, van Der Hoog, Krogh, and Rotenberg [17] observe that all algorithmic logic in [30] requires only the *relative* x-coordinates between points. They adapt [30] to give an efficient and robust implementation of what they call *a rank-based convex hull* data structure $T(S)$ with $O(\log^2 n)$ update time. For ease of exposition, we overly simplify their functionality:

For each $[a, b] \in \Lambda(f)$, we store $S[a, b]$ in $T(S[a, b])$. $T(S[a, b])$ maintains a balanced binary tree $\gamma(S[a, b])$ storing the edges of $\mathrm{CH}(F_{S[a,b]})$ in their cyclical ordering. We use this data structure as a black box, using the following functions that take at most $O(\log^2 n)$ time:

- $T(S[a, b])$.`get_hull()` returns the tree $\gamma(S[a, b])$.
- $T(S[a, b])$.`split(`v`)` returns, for $v \in [a, b]$, $T(S[a, v])$ and $T(S[v, b])$.
- $T(S[a, b])$.`split(`$T([S[b, c])$`)` returns $T(S[a, c])$.
- $T(S[a, b])$.`update(`v`)` updates, for $v \in [a, b]$, the set S (deleting or inserting v).

3 Testing whether a set can be ε-covered by a single segment

We consider the following subproblem: given a parameter ε, a set S of n distinct integers, and the edges of $\mathrm{CH}(F_S)$ stored in a balanced binary tree, can we compute in $O(\log n)$ time whether there exists an ε-cover f of complexity 1? Formally, we seek a line ℓ of slope at least

1 such that all points in F_S lie within L_∞-distance ε of ℓ. Let L (resp. U) denote the set obtained by shifting each $p \in F_S$ downwards and rightwards (resp. upwards and leftwards) by ε, and adding the point $(\infty, -\infty)$ (resp. $(-\infty, \infty)$).

▶ **Lemma 6.** *Let ℓ be a line of slope at least 1. Then all points in F_S lie within L_∞-distance ε of ℓ if and only if ℓ lies below all points in U and above all points in L.*

Proof. Any line with positive slope lies above $(\infty, -\infty)$ and below $(-\infty, \infty)$. Consider a point $p \in F_S$ and the two corresponding points $l \in L$ and $u \in U$ and denote by C an axis-aligned square of radius ε centred at p. If ℓ lies below l then all points on ℓ left of l lie below C. If ℓ lies above u then all points on ℓ right of u lie above C. If ℓ lies above l and below u then because ℓ has positive slope, it must intersect C. The statement follows. ◀

▶ **Corollary 7.** *ℓ is an ε-cover of S iff it has a slope ≥ 1 and separates $CH(L)$ from $CH(U)$.*

Given $\mathrm{CH}(F_S)$, we can extract $\mathrm{CH}(L)$ and $\mathrm{CH}(U)$ in $O(\log n)$ time. Chazelle and Dobkin [6, Section 4.2] remark that, in the negative case, convex hull intersection testing can be modified to produce a separating line. In our setting, the hulls consist of segments with slope at least 1, and any such separator corresponds to an ε-cover. Thus, our problem reduces to the classical convex hull intersection problem and we are seemingly done.

However, the history of convex hull intersection testing is long and intricate. Both Chazelle and Dobkin [6] and Dobkin and Kirkpatrick [10] independently proposed the first $O(\log n)$-time algorithms. In 1987, Chazelle and Dobkin [5] presented a more detailed description of their method. Dobkin and Kirkpatrick revisited their own work in 1990 [11], proposing a unified $O(\log^2 n)$-time algorithm for polyhedron intersection, which O'Rourke later identified as incorrect [28]. He corrected the argument and provided a C-implementation. Further work by Dobkin and Souvaine [9] noted that earlier implementations lacked robustness. More recently, Barba and Langerman [2] observed that the community still lacked a complete, robust algorithm for polyhedral intersection. They proposed an alternative $O(\log n)$ algorithm based on polar transformations. Walther's master's thesis [35], supervised by Afshani and Brodal, implemented both this and earlier methods, but the source code is no longer available.

This 35-year history highlights the complexity and subtlety of convex hull intersection testing. Despite its history, no robust and modern $O(\log n)$-time implementation is available. Moreover, no published algorithm explicitly computes a separating line in the negative case.

Contribution. In the full version, we present a robust $O(\log n)$-time algorithm for convex hull intersection testing. Our algorithm is specialised to convex hulls composed of positively sloped segments and including the points $(\infty, -\infty)$ and $(-\infty, \infty)$. We formally prove its correctness and adapt it to compute a separating line in the negative case, thereby constructing an ε-cover of complexity 1 when it exists. The later adaption and its analysis are nontrivial, and arguably (partly) fill a gap in the existing literature on convex hull intersection testing.

▶ **Theorem 8.** *Let A and B be convex chains of edges with slope at least 1, stored in a balanced binary tree on their left-to-right order. There exists an $O(\log n)$ time to decide whether there exists a line that separates A and B. This algorithm requires only orientation-testing for ordered triangles and can output a separating line whenever it exists.*

4 Robustness

A geometric predicate is a function that takes geometric objects and outputs a Boolean. Our algorithms compute geometric predicates and use their output to branch along a decision tree. In F_S, consecutive points differ in x-coordinate by exactly 1 whilst their y-coordinate

may wildly vary. Consequently, any segment that ε-covers a subsequence of F_S is quite steep. This quickly leads to rounding errors when computing geometric predicates, which in turn creates robustness errors. To illustrate our point, we discuss one of our main algorithms:

Algorithm 1 `intersection_test`(`edge` $\alpha \in CH(A)$, `edge` $\beta \in CH(B)$).

```
1: if α = null OR β = null then
2:     return No
3: end if
4: s(α, β) = line(α) ∩ line(β)
5: if If s ∈ α and s ∈ β then
6:     return Yes
7: end if
8: if α.slope < β.slope then
9:     if α.first.x > s(α, β).x then
10:        return intersection_test(α.left, β)
11:    else if β.first.x > s(α, β).x then
12:        return intersection_test(α, β.left)
13:    else if α.first.x > β.second.x AND α.first.y > β.second.y then
14:        return intersection_test(α.left, β)
15:    else if α.second.x < β.first.x AND α.second.y < β.first.y then
16:        return intersection_test(α, β.left)
17:    else
18:        return yes
19:    end if
20: end if
21: if α.slope > β.slope then
22:    if α.second.x < s(α, β).x then
23:        return intersection_test(α.right, β)
24:    else if β.second.x < s(α, β).x then
25:        return intersection_test(α, β.right)
26:    else if α.first.x > β.second.x AND α.first.y > β.second.y then
27:        return intersection_test(α, β.right)
28:    else if α.second.x < β.first.x AND α.second.y < β.first.y then
29:        return intersection_test(α.right, β)
30:    else
31:        return yes
32:    end if
33: end if
```

`intersection_test` (Algorithm 1) which determines whether an upper quarter convex hull $CH(A)$ and a lower quarter convex hull $CH(B)$ intersect. We receive these hulls as two trees. Our algorithm computes a few geometric predicates given the edges α and β stored at their respective roots. Given (α, β), we either conclude that $CH(A)$ and $CH(B)$ intersect, or, that all edges succeeding (or preceding) α (or β) cannot intersect the other convex hull. Based on the Boolean output, our algorithm then branches into a subtree of α (or β). This way, we verify whether $CH(A)$ and $CH(B)$ intersect in logarithmic time. Rounding causes these predicates to output a wrong conclusion, and our algorithm may branch into a subtree containing edges of $CH(A)$ that are guaranteed to not intersect $CH(B)$. Our algorithm then wrongfully concludes that there exists a line ℓ separating $CH(A)$ and $CH(B)$. Subsequent algorithms then exhibit undefined behaviour when they attempt to compute this line.

Geometric predicates. Our algorithms use on three predicates for their decision making:

- `slope`. Given positive segments (α, β), output whether $\text{slope}(\alpha) < \text{slope}(\beta)$.
- `lies_right`. Given two positive segments α and β with different slopes, output whether the first vertex of β lies right of $line(\alpha) \cap line(\beta)$.
- `wedge`. Consider a pair of positive segments (α, γ) that share a vertex and define W as the cone formed by their supporting halflines containing $(\infty, -\infty)$. Given a positive segment β outside of W, output whether $line(\beta)$ intersects W.

The segments are given by points with integer coordinates. The slopes of these segments (and thereby any representation of their supporting line) are often not integer. A naive way to compute these predicates is to represent slopes using *doubles*. However, this is both computationally slow and prone to rounding errors (and thus, robustness errors).

If we insist on correct output, one can use an *algebraic type* instead. This type represents values using algebraic expressions. E.g., the slope of a positive segment (a, b) is the quotient: $\frac{b.y-a.y}{b.x-a.x}$ and so, in our case, it can be represented as a pair of integers. Algebraic types can subsequently be accurately compared to each other. Indeed, if we want to verify whether $\frac{s}{t} < \frac{q}{r}$ we may robustly verify whether $sr < qt$ using only integers. Exact (algebraic type) comparisons are frequently implemented, and present in the CGAL CORE library [12].

However, exact comparisons are expensive. Our implementation of `slope` requires two integer multiplications, which is still relatively efficient. Evaluating more complex expressions requires too much time. As a rule of thumb, we want to avoid compounding algebraic types to maintain efficiency. Naïvely, `lies_right` compounds two quotients and `wedge` compounds three. We give robust implementations of these functions by invoking three subfunctions. These compare slopes, or whether a point lies above or below a supporting halfplane:

$$\begin{aligned}
&\texttt{slope}((a,b),(c,d)) &&:= (b.y - a.y) \cdot (d.x - c.x) < (d.y - c.y) \cdot (b.x - a.x)\\
&\texttt{above_line}((a,b),c) &&:= (b.x - a.x)(c.y - b.y) - (c.x - b.x)(b.y - a.y) \geq 0\\
&\texttt{below_line}((a,b),c) &&:= (b.x - a.x)(c.y - b.y) - (c.x - b.x)(b.y - a.y) \leq 0
\end{aligned}$$

We can create `lies_right` from our robust predicates (see Figure 3 (a)):

▶ **Lemma 9.** *Let $\alpha = (a, b)$ and $\beta = (c, d)$ be two positive segments of different slope. Then:*

$$\begin{aligned}
\texttt{lies_right}(\alpha,\beta) = &\big(\texttt{slope}((a,b),(c,d)) == \texttt{above_line}((a,b),c)\big)\\
&\vee \big(\texttt{slope}((c,d),(b,c)) == \texttt{below_line}((a,b),c)\big)
\end{aligned}$$

Proof. Suppose that $\text{slope}(\alpha) < \text{slope}(\beta)$. Then c lies right of $line(\alpha) \cap line(\beta)$ if and only if c lies above the halfplane bounded from above by $line((a, b))$. That happens if and only if (a, b, c) are collinear or make a counter-clockwise turn. This in turn occurs if and only if the determinant if the matrix $\begin{vmatrix}(b.x - a.x) & (c.x - b.x)\\ (b.y - a.y) & (c.y - b.y)\end{vmatrix}$ is zero or more. If $slope(\alpha) > slope(\beta)$ the determinant must be negative instead. ◀

Similarly, we can create `wedge` from our robust predicates. We note for the reader that explain our equations in words in the proof of the lemma:

▶ **Lemma 10** (Figure 3 (b))**.** *If $\alpha = (a, b)$, $\gamma = (b, c)$ and $\beta = (d, e)$ be three segments of positive slope where $W = \overleftarrow{\alpha} \cup \overrightarrow{\gamma}$ bounds a convex area containing $(\infty, -\infty)$. Then*

$$\begin{aligned}
&\texttt{wedge}(\alpha,\gamma,\beta) :=\\
&\big(\texttt{below_line}((b,c),d) \wedge \big(\texttt{above_line}((d,e),b) \vee \texttt{slope}((a,b),(d,e))\big)\big) \vee\\
&\big(\texttt{below_line}((a,b),e) \wedge \big(\texttt{above_line}((d,e),b) \vee \texttt{slope}((d,e),(b,c))\big)\big) \vee\\
&\big(\neg\texttt{below_line}((a,b),e) \wedge \neg\texttt{below_line}((b,c),d) \wedge \big(\texttt{slope}((a,b),(d,e)) \vee \texttt{slope}((b,c),(d,e))\big)\big)
\end{aligned}$$

Proof. The predicate is a case distinction of three mutually exclusive cases.

If the first vertex of β lies below the supporting line of γ then $line(\beta)$ intersects W if and only if it intersects $\overleftarrow{\alpha}$. This happens if and only if one of two conditions hold: either b lies below the supporting line of b, or, $\text{slope}(\alpha) < \text{slope}(\beta)$.

If the second vertex of β lies below $line(\alpha)$ then the argument is symmetric.

If neither of those cases apply then both endpoints of β must lie in the open green area. In this case, whenever $\text{slope}(\alpha) < \text{slope}(\beta)$, the supporting line of β always intersects W. Whenever $\text{slope}(\beta) < \text{slope}(\gamma)$, the supporting line of β always intersects W. Whenever $\text{slope}(\alpha) \geq \text{slope}(\beta) \geq \text{slope}(\gamma)$, the supporting line of β cannot intersect W. ◀

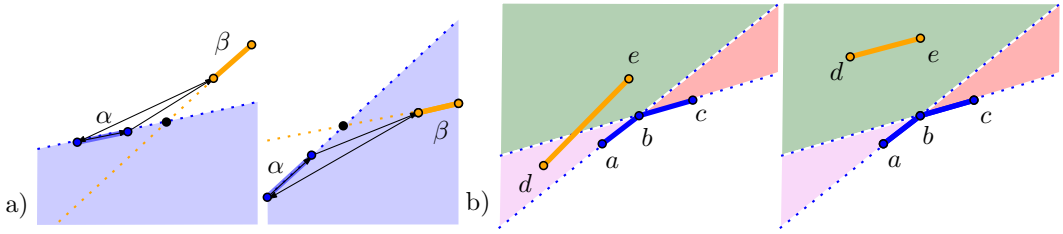


Figure 3 (a) We reduce testing whether the first vertex of β lies right of the intersection point to comparing slopes and the orientation of a triangle. (b) If d lies below the halfplane of $line(b, c)$ then $line((d, e))$ intersects the wedge if and only if b lies below $line((d, e))$.

5 Dynamically maintaining a learned index

We dynamically maintain a learned index h_ε of S by maintaining an ε-cover f of S. We guarantee that there exists no ε-cover f' of S with $|f| > 2|f'|$. By Observation 3, we obtain a learned index h_ε. By Observation 4, there exists no PGM index h_ε where $|f| > 2|h_\varepsilon|$.

To maintain f, we maintain a balanced binary tree $B(f)$ over $\Lambda(f)$. Additionally, for each $[a, b] \in \Lambda(f)$, we maintain a *rank-based convex hull* $T(S[a, b])$ of $S[a, b]$ as described in [17]. We note that we store all segments in f using *relative* x-coordinates. That is, we assume for all $[a, b] \in \Lambda(f)$ that the rank of the first element in $S[a, b]$ is zero. We may then use $B(f)$ to "offset" each line to compute the actual coordinates in rank-space.

▶ **Theorem 11.** *We can dynamically maintain an ε-cover f of S in $O(\log^2 n)$ worst-case time. We guarantee that there exists no ε-cover f' of S where $|f| > 2|f'|$.*

Proof. The proof is illustrated by Figure 4. For any $s, t \in \mathbb{Z}$ with $s \leq t$, we say that $S[s, t]$ is *blocked* if there exists no ε-cover of $S[s, t]$ of size 1. We maintain an ε-cover f where for all consecutive intervals $[a, b], [c, d] \in \Lambda(f)$, $S[a, d]$ is blocked. Thereby, $|f| \leq 2|f'|$ for any ε-cover f' of S (we give a proof of this fact in the full version).

We consider inserting a value s into S; deletions are handled analogously. We query $B(f)$ in $O(\log n)$ time for an interval $[a, b]$ that contains s. If no such interval exists, set $[a, b] = [s, s]$. We search $T(S[a, b])$ and test whether $s \in S$. If so, we reject the update.

Otherwise, we remove $[a, b]$ from $\Lambda(f)$ and insert the intervals $([a, s], [s, s], [s, b])$. We obtain $T(S[a, s])$, $T(S[s, s])$ and $T(S[s, b])$ through the split operation.

Let $([w, x], [y, z], [a, s], [s, s], [s, b], [c, d], [e, f])$ be consecutive intervals in $\Lambda(f)$ and denote $I = ([y, z], [a, s], [s, s], [s, b], [c, d])$ (see Figure 4 (c)). For each $(s, t) \in I$, we have access to $T(S[s, t])$. For any consecutive pair $([s, t], [q, r])$ in I, we may join the trees $T(S[s, t])$ and $T(S[q, r])$ in $O(\log^2 n)$ time to obtain $T([s, r])$. We invoke $T([s, r])$.`get_hull()` and apply

Theorem 8 to test in $O(\log^2 n)$ total time whether $S[s,r]$ is *blocked*. If it is not, we replace $[s,t]$ and $[q,r]$ by $[s,r]$. Otherwise, we keep $T(S[s,r])$ and a complexity-1 ε-cover of $S[s,r]$.

By recursively merging pairs in I, we obtain in $O(\log^2 n)$ time a sequence I' of intervals $([y,\beta],\ldots,[\gamma,d])$ where consecutive intervals are blocked. Since $[y,z] \subseteq [y,\beta]$, $([w,x],[y,\beta])$ is blocked. Similarly, $([\gamma,d],[e,f])$ must be blocked. We remove the line segments corresponding to I from f and replace them with line segments derived from I' in constant time. As a result, we maintain our ε-cover f and our data structure in $O(\log^2 n)$ total time. ◀

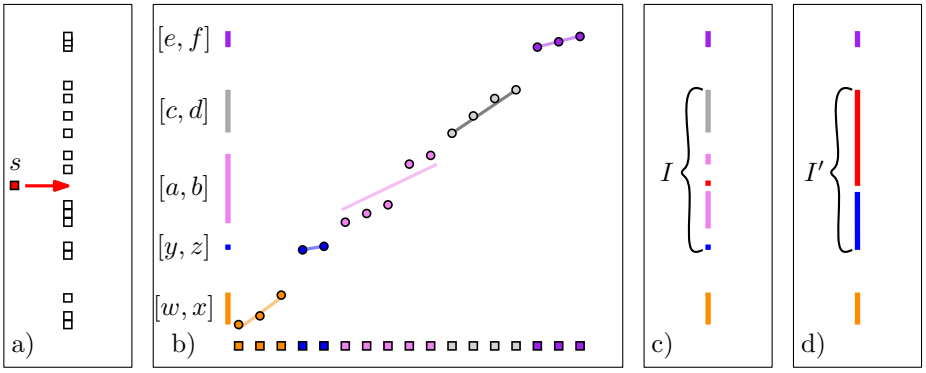


Figure 4 (a) Let S be a set of values and let us insert s. (b) We consider our ε-cover f and five consecutive intervals in $\Lambda(f)$. (c) We create seven intervals by splitting $[a,b]$ on s. (d) By recursively merging intervals in I, we obtain a set of intervals I' where consecutive intervals are blocked.

6 From an ε-cover to an indexing structure

A learned index h_ε does not immediately support indexing and range queries. We obtain an indexing structure by combining h_ε with a hash map H. Combining learned models with hash maps is not new [25, 34, 36] and this technique has even been applied to learned indexing [24]. The core idea is to store S in an unordered vector A and maintain a Hash map $H : \mathbb{Z} \mapsto [n]$. Given some $q \in \mathcal{U}$, the learned function then produces a value v such that $A[h(v)]$ is "close" to q. It is compelling to create H such that $A[H(h(q))]$ is (approximately) the predecessor of q. However, dynamically, this approach fails for the same reason that storing each $s \in S$ at $A[\texttt{rank}(s)]$ fails. Since the ranks of elements in S are constantly changing, we build a hash map using the parts of S that remain constant: the values.

Our data structure. In the full version, we define a data structure independent of the learned index h_ε (we illustrate our approach in Figure 5 (a) + (b)). A *page* p is an integer with a vector that stores all $s \in S$ where $\lfloor \frac{s}{\varepsilon} \rfloor = p$, in order. We store all non-empty pages P in an unordered vector A. We maintain a hash map $H : P \to [|A|]$, where $A[H(p)]$ contains the page p. We additionally maintain a doubly linked list over all pages in P, arranged in sorted order.

Our queries. We restrict our learned index h_ε to a *vertical ε-cover*. I.e., h_ε is a y-monotone collection of line segments such that for all points $p \in F_S$, a vertical line segment of height 2ε centred at p intersects a segment in h_ε. We compute h_ε oblivious of our paging structure.

Given $q \in \mathcal{U}$, we project q onto h_ε (Figure 5 (d)). We project to the x-axis, floor the value, and project back to h_ε. We prove that the resulting y-value corresponds to the page p containing `predecessor`(q). This way, we answer `predecessor` using $O(\varepsilon + \log |h_\varepsilon|)$ time.

As a result, we dynamically maintain a learned index h_ε and a data structure that updates in $O(\varepsilon + \log^2 n)$ and supports indexing queries in $O(\varepsilon + \log |h_\varepsilon|)$ expected time (Theorem 12).

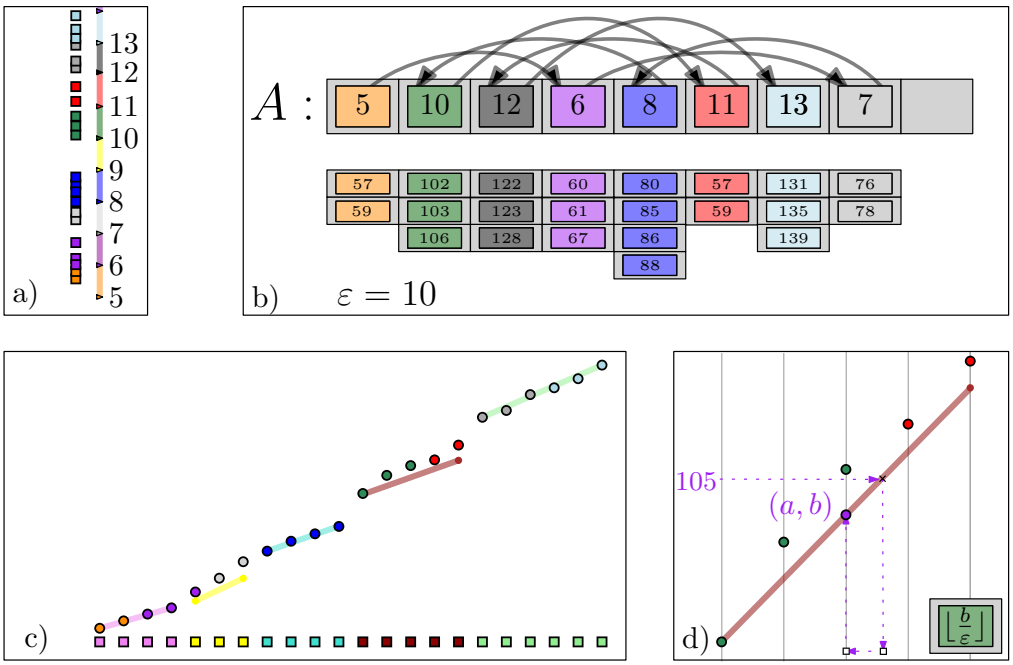


Figure 5 An illustration of our approach in the full version.

▶ **Theorem 12.** *For any ε, there exists a data structure to dynamically maintain a vertical ε-cover F of a dynamic set of distinct integers in $O(\varepsilon + \log^2 n)$ time. We guarantee that there exists no vertical ε-cover F' with $|F| > 2|F'|$. The data structure supports indexing queries in $O(\varepsilon + \log |F|)$ expected time and range queries in additional $O(k)$ time where k is the output size.*

7 Experiments

Our implementation is written in `C++` and made publicly available [18]. We compare to the `C++` implementation in [16], which uses a PGM index under the logarithmic method. The experiments were conducted on a machine with a 4.2GHz AMD Ryzen 7 7800X3D and 128GB memory. Our test bench is available [19], and can replicate experiments, generate synthetic data, and produce plots. As input we consider two synthetic data sets and two real world data sets. Three contain data of geometric nature, with one of random nature to align with precedent. Each set consists of unique 8 byte integers in randomly shuffled order. In the full version, we showcase additional experiments on other datasets.

- **LINES** is a synthetic data set of 5M integers that, in rank space, produces 5 lines of exponentially increasing slope. This set models the ideal scenario for a PGM index.
- **LONGITUDE** is a real world data set that contains the longitudes of roughly 246M points of interest from OpenStreetMap, over the region of Italy. This data is thereby inherently of geometric nature. This data set was used in both [16] and [21]. We follow [16] and convert the data to integers by removing the decimal point from the raw longitudes.

- **UNIF** originates from [16]. It is a synthetic data set, containing a uniform random sample of 50M integers from $(0, 10^{11})$. We adapt this data set to our dynamic setting.
- **DRIFTER** is a real world data set, containing roughly 1.7M steps of accumulated distance travelled by ocean drifters tracked through GPS [7].

Measurements. We compare the quality of the learned indices based the complexity of h_ε, in a dynamic setting. We use the same choice of $\varepsilon = 64$ as in [16] across our experiments. For performance of the indexing structures, we measure their time per operation in a dynamic scenarios with a range of query to update ratios. We note that logarithmic PGM is by default equipped with an optimisation that avoids building a PGM for data below a certain size. In this case, it instead only uses an underlying sorted array without additional search structure. In order to properly compare the performances, this optimisation has been disabled.

7.1 The learned index complexity

Figure 6 presents the complexity of the learned indices, measured by the number of line segments maintained during random-order insertions. The behaviour differs between datasets. For the geometric Lines and Longitude datasets, the dynamic and logarithmic PGMs initially perform similarly. As the data grows, the performance of the logarithmic PGM degrades and its jagged progression reflects its logarithmic partitioning.

On the highly structured synthetic Lines data, the logarithmic method consistently retains more segments than necessary. It misses the optimal line count by a wide margin due to its fragmentation across $O(\log n)$ buckets. A similar pattern appears in the Longitude dataset, where the logarithmic PGM maintains roughly 50 percent more segments than our solution. We note that precisely on these structured data sets, the complexity of the learned indices is $o(n)$. I.e., precisely here one also expects improvements in query time.

For unstructured data such as Unif, both learned indices display similar asymptotic trends, with the logarithmic PGM using approximately 30 percent fewer segments. Here, the complexity appears to scale as $\Omega(n)$ for both methods, suggesting that learned indices offer few to no improvements in the absence of exploitable structure. Surprisingly, the Drifter data – despite its geometric origin – shows similar results to that of Unif. Again, complexity scales linearly, and the logarithmic method outperforms the dynamic one by around 30 percent. This implies that whatever latent geometric structure exists is insufficiently captured by the learned index under either strategy.

An interesting observation is that, on unstructured data, the imposed bucketing of the logarithmic PGM index can introduce a form of regularity that the model benefits from, essentially imposing artificial structure where none exists.

7.2 Running time comparisons

Recall that our indexing structure is composed of a learned index over a vertical ε-cover and a paging structure. We first examine the performance of both the learned index in its own, and then the performance of the full indexing structure.

For the dynamic scenarios, we first follow the precedence set by prior papers [16, 21]. These first construct, insertion-only, the indexing structure. They then perform a batch of 10M operations These batches consist of insertions, deletions, and range queries over ranges such that the output contains approximately $\frac{\sqrt{n}}{10}$ elements. We deviate from the precedent by also deleting from the index before performing a batch of operations, to simulate a scenario in which the structure has existed and transformed prior to processing.

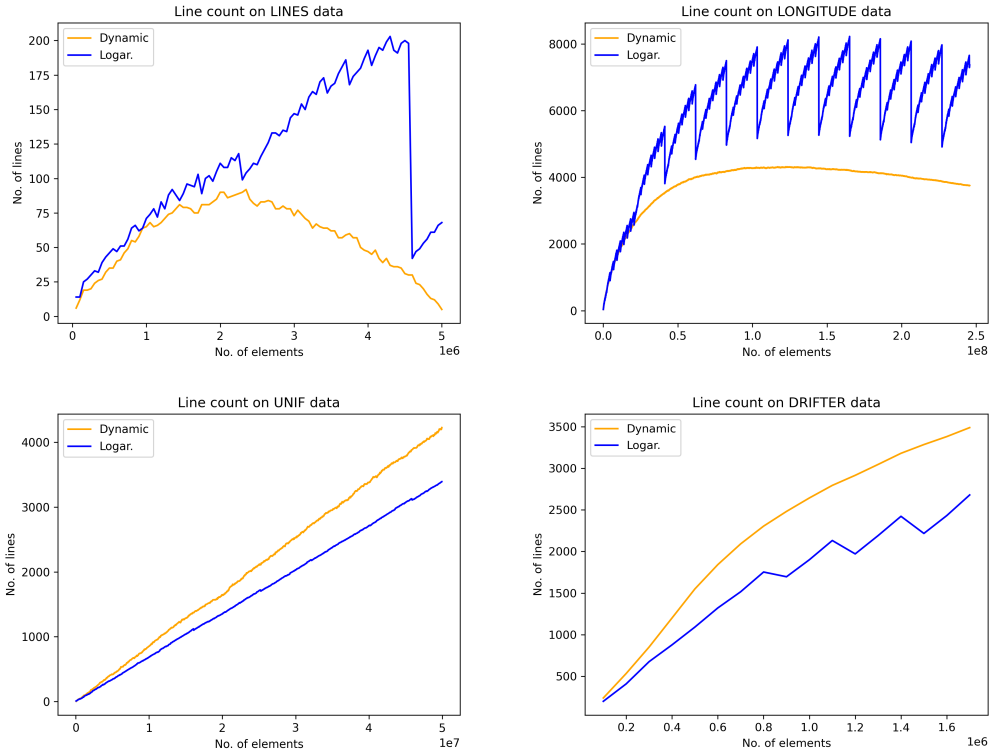


Figure 6 The complexity of the learned indices throughout insertion-only construction. The top graphs represent geometrically structured data. The left graphs represent synthetic data.

7.2.1 Maintaining a learned index

Figure 7 shows the cost of maintaining each learned index under dynamic operations. Our update procedure, which ensures worst-case $O(\log^2 n)$ bounds, performs significantly worse than the amortised $O(\log n)$ updates offered by the logarithmic method – especially on larger datasets such as LONGITUDE and UNIF. On smaller datasets like DRIFTER, the gap narrows, but the logarithmic method generally remains preferable in these update-only scenarios.

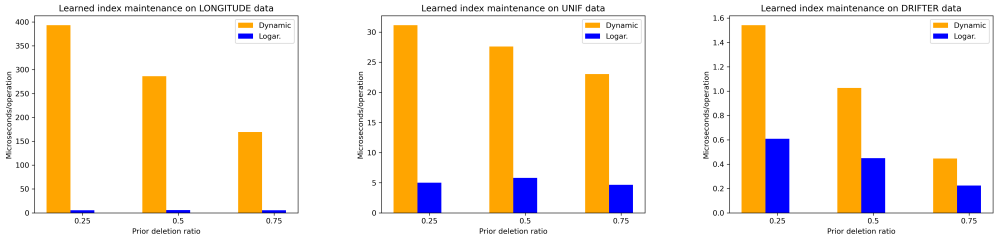


Figure 7 Update times for maintaining the learned index dynamically.

7.2.2 Indexing data structures

We next assess the complete indexing structures under mixed workloads. Batches contain a tunable ratio of queries and updates, where updates are evenly split between insertions and deletions, and operations are randomly ordered. More experiments can be found in the full version. For the LINES dataset, updates are constrained to a single line segment to preserve its idealised structure. For smaller datasets, batch size is limited to 25 percent.

Adversarial workload (ADV). In the above scenario by [16, 21], an update batch of $10M$ operations affects less than ten percent of the data. Therefore, we do not encounter the worst case scenario where range queries take $O(N + \varepsilon + \sum_i^{\lceil \log n \rceil} \log |f_i|)$ time. So, the worst case difference in performance does not come to light. Therefore, we additionally construct an *adversarial scenario* consisting of 10M range queries after deleting all but 1.000 values.

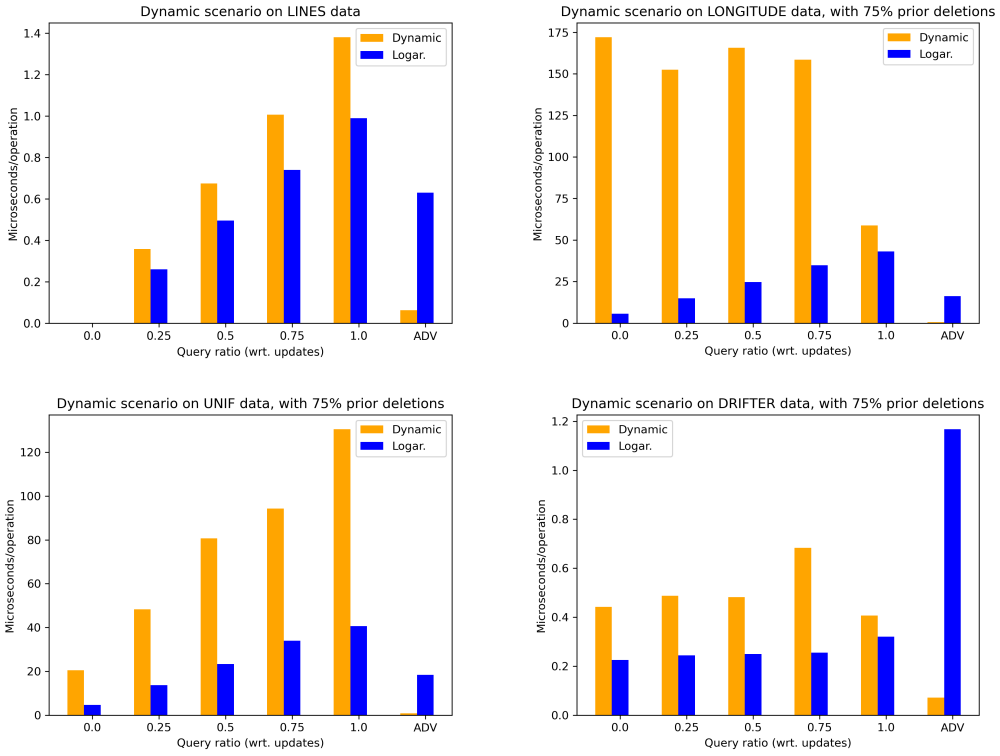


Figure 8 Time per operation in dynamic scenario with varying query ratios. For the LINES data set updates are restricted to points on a single line. The ADV ratio denotes the adversarial case where all but a constant number of elements are deleted before the structure is queried.

Operational performance. Figure 8 summarises the total processing time under varying query-to-update ratios. Structured datasets, particularly LONGITUDE, reveal a trade-off between the fact that we have a lower-complexity learned index and our update time. On one hand, our updates are costly due to our line-merging tests which suffer from the poor cache behaviour from pointer-based trees, and complex rebalancing that is present in the state-of-the-art practical dynamic convex hull data structures. This behaviour is also reflected on our previous analysis of maintaining the learned index itself. On the other hand we see

on geometrically structured data that, as the query ratio increases, performance improves. This trend is absent in the LINES data, where updates are constrained and the structure is small, mitigating cache penalties. Here, running time is largely driven by query processing, which increases as expected with query ratio.

In contrast, the random data from the UNIF dataset shows improved performance with higher query ratios. In our data structure, having smaller segments reduce restructuring costs. Our dynamic implementation suffers greatly from cache inefficiency due to random access during range reporting. The logarithmic PGM, with its lower memory overhead and sequential layout, delivers significantly better performance – particularly as sizes increase.

For the DRIFTER data, one would expect a trend similar to that of the UNIF data, based on the complexity of the ε-cover shown in Figure 6. However, there is little difference in performance for either PGM as the query ratio is varied. This is likely due to the small size of the data set, with both structures performing updates in fractions of microseconds on average. The logarithmic PGM does come out slightly on top, likely due to the machine friendly memory access pattern.

Adversarial impact. Across all datasets, the logarithmic PGM's tombstoning strategy becomes a bottleneck in the adversarial scenario. Its range queries must scan and subtract deleted values before outputting results. Our structure, by contrast, is output-sensitive and avoids such overhead. This illustrates that our data structure, whilst being generally less efficient that the logarithmic PGM, offers worst-case guarantees.

8 Conclusion

We studied dynamic learned indices through a geometric perspective. Following the work of Ferragina and Vinciguerra [16], we maintained a learned index h_ε of a dynamic set S as a piecewise-linear approximation – an ε-cover – of the rank-space point set F_S. We used techniques from computational geometry to answer the following question:

"Can a learned index be dynamically maintained with worst-case guarantees?"

We proposed a new approach to maintain a learned index based on dynamic convex hull data structures. We presented an $O(\log^2 n)$ time algorithm to dynamically maintain a learned index h_ε. To obtain this algorithm, we showed an algorithm to compute a separating line between two non-intersecting convex hulls – an operation previously missing from the literature. The existing logarithmic PGM index has an amortised $O(\log n)$ time update algorithm which is more efficient in practice. Indeed, although close in theory, the memory access pattern associated with dynamically maintaining convex hulls incurs heavy penalties for large datasets. At the same time, the resulting learned index of the logarithmic PGM, h'_ε, can be considerably more complex on geometrically structured data.

From learned indices to indexing. Finally, we considered the following question:

"Can a dynamic learned index be converted into a dynamic indexing data structure?"

To this end, we designed a hybrid technique combining hashing-based fast-access data structures with a doubly linked list to support indexing queries. Our method offers output-sensitive worst-case guarantees, even in the presence of deletions. As it is known that traditional indexing structures currently outperform learned indices in the general dynamic setting [33], we focused our comparisons on improving the theoretical and practical performance within the class of learned approaches.

In practice, the contiguous memory access of the logarithmic PGM index offsets its overhead from querying multiple structures, making it faster in all scenarios, except for an adversarial one. This means that the lower complexity of our learned index does not immediately translate to improved efficiency in the indexing data structure. This raises an interesting open question of whether memory-access efficient fully dynamic approaches to convert a learned index into a dynamic indexing structure can exist.

While we acknowledge that our update-times are slow in comparison with state-of-the art, our approach does illustrate that it brings worst-case guarantees: as it has an advantage when the query-to-update ratio is large and the index has undergone sufficiently many deletions. In adversarial workloads with frequent deletions followed by range queries, we have seen our structure outperform the logarithmic approach – highlighting the value of worst-case guarantees even in specialised settings.

Closing thoughts. We showed what we believe is an interesting connection between the geometric learned index by Ferragina and Vinciguerra, and dynamic convex hulls from computational geometry. We subsequently provided an implementation of a dynamic learned index that relies on the state-of-the-art dynamic convex hull maintenance algorithms. Our empirical analysis shows that the complex tree rebalancing that is used to dynamically maintain a convex hull currently brings considerable operational overhead compared to low-memory techniques under the logarithmic method. Our experiments, though not uniformly favourable, offer interesting insights into the current barriers and adversarial tradeoffs between worst-case dynamic algorithms and memory-efficient amortised rebuilding schemes.

References

1 Manos Athanassoulis and Anastasia Ailamaki. Bf-tree: approximate tree indexing. In *International Conference on Very Large Databases (VLDB)*, 2014.

2 Luis Barba and Stefan Langerman. Optimal detection of intersections between convex polyhedra. *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015.

3 Michael A Bender, Erik D Demaine, and Martin Farach-Colton. Cache-oblivious b-trees. In *Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2000.

4 Chee-Yong Chan and Yannis E Ioannidis. Bitmap index design and evaluation. In *ACM International Conference on Management of Data (SIGMOD)*, 1998.

5 B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *Journal of the ACM*, 1987. doi:10.1145/7531.24036.

6 Bernard Chazelle and David P Dobkin. Detection is easier than computation. In *ACM Symposium on Theory Of Computing (STOC)*, 1980.

7 Jacobus Conradi and Anne Driemel. Finding complex patterns in trajectory data via geometric set cover. *CoRR*, abs/2308.14865, 2023. doi:10.48550/arXiv.2308.14865.

8 Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, et al. Alex: an updatable adaptive learned index. In *ACM International Conference on Management of Data (SIGMOD)*, 2020.

9 David Dobkin and Diane Souvaine. Detecting the intersection of convex objects in the plane. *Computer Aided Geometric Design*, 1991. doi:10.1016/0167-8396(91)90001-R.

10 David P. Dobkin and David G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science (TSC)*, 1983. doi:10.1016/0304-3975(82)90120-7.

11 David P. Dobkin and David G. Kirkpatrick. Determining the separation of preprocessed polyhedra - a unified approach. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 1990.

12 Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schönherr. On the design of cgal a computational geometry algorithms library. *Software: Practice and Experience*, 30(11):1167–1202, 2000. doi:10.1002/1097-024X(200009)30:11\%3C1167::AID-SPE337\%3E3.0.CO;2-B.

13 Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. Why are learned indexes so effective? In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3123–3132. PMLR, 2020. URL: http://proceedings.mlr.press/v119/ferragina20a.html.

14 Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. On the performance of learned data structures. *Theor. Comput. Sci.*, 871:107–120, 2021. doi:10.1016/J.TCS.2021.04.015.

15 Paolo Ferragina and Giorgio Vinciguerra. Learned data structures. In *Recent Trends in Learning From Data: Tutorials from the INNS Big Data and Deep Learning (INNSBDDL)*. Springer, 2020.

16 Paolo Ferragina and Giorgio Vinciguerra. The pgm-index: a fully-dynamic compressed learned index with provable worst-case bounds. *International Conference on Very Large Databases (VLDB)*, 2020.

17 Emil Gæde, Inge Li Gørtz, Ivor Van Der Hoog, Christoffer Krogh, and Eva Rotenberg. Simple and robust dynamic two-dimensional convex hull. *ACM Symposium on Algorithm Engineering and Experiments (ALENEX)*, 2024.

18 Emil Toftegaard Gæde, Ivor van der Hoog, Eva Rotenberg, and Tord Stordalen. A Dynamic Piecewise-Linear Geometric Index with Worst-Case Guarantees. Software, Carlsberg Fonden CF21-0302, Villum Fonden VIL37507, Marie Skłodowska-Curie 899987, swhId: swh:1:dir:b8763eb0504d33beb81ee89d230a30dca8ab0b66 (visited on 2025-09-03). URL: https://github.com/Sgelet/DynamicLearnedIndex, doi:10.4230/artifacts.24667.

19 Emil Toftegaard Gæde, Ivor van der Hoog, Eva Rotenberg, and Tord Stordalen. Testbed for our learned index repository. Software, Carlsberg Fonden CF21-0302, Villum Fonden VIL37507, Marie Skłodowska-Curie 899987, swhId: swh:1:dir:07ea25cfc176438933c1a5507bfdad3ba9461ab6 (visited on 2025-09-03). URL: https://github.com/Sgelet/LearnedIndexBench, doi:10.4230/artifacts.24668.

20 Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. Fiting-tree: A data-aware index structure. In *ACM International Conference on Management of Data (SIGMOD)*, pages 1189–1206, 2019. doi:10.1145/3299869.3319860.

21 Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. Sosd: A benchmark for learned indexes. *Conference on Neural Information Processing Systems (NEURIPS)*, 2019.

22 Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. Radixspline: a single-pass learned index. In *International workshop on exploiting artificial intelligence techniques for data management*, 2020.

23 Nick Koudas. Space efficient bitmap indexing. In *ACM international conference on Information and knowledge management (SIGMOD)*, 2000.

24 Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *ACM International Conference on Management of Data (SIGMOD)*, 2018.

25 Yuming Lin, Zhengguo Huang, and You Li. Learning hash index based on a shallow autoencoder. *Applied Intelligence*, 2023.

26 Ryan Marcus, Andreas Kipf, Alexander van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. Benchmarking learned indexes. *Proc. VLDB Endow.*, 14(1):1–13, 2020. doi:10.14778/3421424.3421425.

27 Joseph O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Communications of the ACM*, 1981.

28 Joseph O'Rourke. *Computational geometry in C (second edition)*. Cambridge University Press, USA, 1998.

29 Mark H Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1983. doi:10.1007/BFB0014927.

30 Mark H Overmars and Jan Van Leeuwen. Maintenance of configurations in the plane. *Journal of computer and System Sciences*, 1981.

31 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 2004.

32 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *ACM Symposium on Theory of Computing (STOC)*, 2006.

33 Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. Learned index: A comprehensive experimental evaluation. *Proc. VLDB Endow.*, 16(8):1992–2004, 2023. doi:10.14778/3594512.3594528.

34 Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2008.

35 Lukas Walther, Gerth Brodal, and Peyman Afshani. Intersection of convex objects in the plane. Master's thesis, Aarhus University, 2015. Available at https://cs.au.dk/~gerth/advising/thesis/lukas-walther.pdf.

36 Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data—a survey. *Proceedings of the IEEE*, 2015.

37 Ziqi Wang, Andrew Pavlo, Hyeontaek Lim, Viktor Leis, Huanchen Zhang, Michael Kaminsky, and David G Andersen. Building a bw-tree takes more than just buzz words. In *ACM International Conference on Management of Data (SIGMOD)*, 2018.

38 Chaichon Wongkham, Baotong Lu, Chris Liu, Zhicong Zhong, Eric Lo, and Tianzheng Wang. Are updatable learned indexes ready? *International Conference on Very Large Databases (VLDB)*, 2022.

From Theory to Practice: Engineering Approximation Algorithms for Dynamic Orientation

Ernestine Grossmann
Heidelberg University, Germany

Henrik Reinstädtler
Heidelberg University, Germany

Eva Rotenberg
IT University of Copenhagen, Denmark

Christian Schulz
Heidelberg University, Germany

Ivor van der Hoog
IT University of Copenhagen, Denmark

Juliette Vlieghe
IT University of Copenhagen, Denmark

Abstract

Dynamic graph algorithms have seen significant theoretical advancements, but practical evaluations often lag behind. This work bridges the gap between theory and practice by engineering and empirically evaluating recently developed approximation algorithms for dynamically maintaining graph orientations. We comprehensively describe the underlying data structures, including efficient bucketing techniques and round-robin updates. Our implementation has a natural parameter λ, which allows for a trade-off between algorithmic efficiency and the quality of the solution. In the extensive experimental evaluation, we demonstrate that our implementation offers a considerable speedup. Using different quality metrics, we show that our implementations are very competitive and can outperform previous methods. Overall, our approach solves more instances than other methods while being up to 112 times faster on instances that are solvable by all methods compared.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic graphs, out-orientation

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.65

Related Version *Full Version*: `https://arxiv.org/abs/2504.16720`

Supplementary Material
Software (Source Code): `https://github.com/DynGraphLab/DynDeltaApprox` [26]
archived at `swh:1:dir:0ea52feafca9f6d8ce3ee893686fde2510513e9e`
Software (Dataset): `https://zenodo.org/records/15785840`

Funding *Ivor van der Hoog*, *Eva Rotenberg*, and *Juliette Vlieghe* are grateful to the VILLUM Foundation for supporting this research via Eva Rotenberg's Young Investigator grant (VIL37507) "Efficient Recomputations for Changeful Problems". Part of this work took place while these authors were affiliated with the Technical University of Denmark.
Ernestine Grossmann, *Henrik Renstädtler*, and *Christian Schulz* acknowledge support by DFG grant SCHU 2567/8-1.
Ivor van der Hoog: This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987.



© Ernestine Grossmann, Henrik Reinstädtler, Eva Rotenberg, Christian Schulz, Ivor van der Hoog, and Juliette Vlieghe;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

A (fully) dynamic graph algorithm is a data structure designed to support edge insertions, edge deletions, and answer specific queries. While there has been extensive research on dynamic graph problems, many of these theoretically efficient algorithms have not been implemented or empirically evaluated. In this work, we dynamically maintain an orientation of a graph. Let $G = (V, E)$ be an undirected graph with n vertices and m edges. A *fractional orientation* $\overrightarrow{G}$ of G assigns to each edge $\{u, v\} \in E$ two non-negative weights $d(u \to v)$ and $d(v \to u)$ such that $d(u \to v) + d(v \to u) = 1$. The *fractional out-degree* $d^+(u)$ of a vertex u is the sum over all edges $\{u, v\} \in E$ of $d(u \to v)$. An *integral orientation* is a fractional orientation in which all weights are either 0 or 1. We will denote by $d^*(u)$ the out-degree of a vertex u if the orientation is integral. Graph orientations have a wide range of applications:

Fractional Orientations and Graph Mining. Given a graph G, a *density decomposition* of G is a nested set of subgraphs of G of decreasing density. Finding dense subgraphs lies at the core of graph mining [25]. Density decompositions were first applied in finding clusters in web-data [23, 24, 30, 31] but quickly embedded themselves in pattern mining techniques of other fields such as social graph mining [2, 11, 18, 34, 41], bioinformatics [3, 21, 28, 39, 44], finance [19] and even drawing algorithms for large graphs [1]. Danisch, Chan, and Sozio [14] propose a state-of-the-art density composition of a graph, which they call the *local density*. Due to its applicability, there exist many *static* implementations to efficiently approximate (variants of) this decomposition [29, 35, 42, 43]. There exists a 1:1 correspondence between this density decomposition and fractional orientations. Indeed, consider a fractional orientation $\overrightarrow{G}$ that minimizes the sum-of-squares metric defined by $\texttt{sum}^2(\overrightarrow{G}) := \sum_{v \in V} \left(d^+(v)\right)^2$. The decomposition by Danish, Chan, and Sozio is the partition of vertices by their out-degree.

Integral Orientations and Dynamic Efficiency. A crucial building block for efficient, fully dynamic algorithms is efficient, dynamic edge-orientations. Given an undirected graph G, the *edge orientation problem* asks for an orientation of G in which the maximum out-degree of the vertices is minimized. Many real-world graphs are relatively sparse and allow for edge orientations where the maximum out-degree is considerably smaller than the maximum degree of the graph. Low out-degree integral orientations allow for efficient and low-memory adjacency queries. This, in turn, has many applications, including dynamic map labeling, graph colouring, maximal matchings, and finding k-cliques [16]. Other example applications include Neiman and Solomon [36] who have shown how to maintain a maximal matching in $O(\log n / \log \log n)$ amortized time using a dynamic edge orientation algorithm, dynamic matrix vector multiplication [32], or Kowalik and Kurowski [33] who use dynamic edge orientations to answer shortest-path queries of length at most k in $O(k)$ time in planar graphs. The aforementioned algorithms use time proportional to the maximum out-degree in the orientation. Thus, it becomes important to minimize $\texttt{max}(\overrightarrow{G}) := \max_{v \in V} d^+(v)$.

Minimax versus Least Square Regression. In graph mining, we are interested in orientations $\overrightarrow{G}$ that minimize $\texttt{sum}^2(\overrightarrow{G})$. Minimising this measure optimises for the "average" out-degree which identifies the density of the neighborhood of each vertex. On the other hand, many dynamic algorithmic application domains are interested in orientations $\overrightarrow{G}$ that minimize $\max(\overrightarrow{G})$. Danisch, Chan, and Sozio prove that, in this case, the least square regression is strictly more stringent than the minimax regression as any fractional orientation $\overrightarrow{G}$ minimizing $\texttt{sum}^2(\overrightarrow{G})$ also minimizes $\max(\overrightarrow{G})$ [14]. The value of $\max(\overrightarrow{G})$ equals the pseudoarboricity of

the graph. The (pseudo-)arboricity is defined as the number of (pseudo-)forests that the graph can be partitioned into. The arboricity is either equal to the pseudoarboricity or plus one [37].

Prior Work in Theory. Theoretical dynamic algorithms typically maintain a fractional orientation $\overrightarrow{G}$ over G. A naive rounding scheme then converts each fractional orientation $\overrightarrow{G}$ to an integral orientation $\overrightarrow{G}^*$ where for all $v \in V$, the out-degree $d^*(v)$ is at most $2d^+(v)$. A simple rounding scheme rounds all fractional edges with a value greater than $\frac{1}{2}$ and ones with equal value by a deterministic scheme, like the assigned vertex id. Christiansen and Rotenberg [12] propose a dynamic polylog-time rounding algorithm where for all $v \in V$, $d^*(v) \leq d^+(v) + 1$. The state-of-the-art dynamic theoretical algorithm to maintain a fractional orientation that minimizes $\texttt{max}(\overrightarrow{G})$ is by Chekuri et al. [10]. They dynamically maintain a fractional orientation $\overrightarrow{G}$ where for all $\overrightarrow{G}'$, $\texttt{max}(\overrightarrow{G}) \leq (1+\varepsilon) \cdot \texttt{max}(\overrightarrow{G}')$. Their algorithm uses linear space and has $O(\varepsilon^{-6} \log^4 n)$ update time. Christiansen, van der Hoog, and Rotenberg [13] show that the algorithm from [10], in actuality, maintains a fractional orientation $\overrightarrow{G}$ such that for all $\overrightarrow{G}'$, $\texttt{sum}^2(\overrightarrow{G}) \leq (1+\varepsilon) \cdot \texttt{sum}^2(\overrightarrow{G}')$. Thus, the state-of-the-art theoretical algorithm maintains a fractional orientation that $(1+\varepsilon)$-approximates the optimal solution for both our metrics of interest in $O(\varepsilon^{-6} \log^4 n)$ time per update.

Prior Work in Practice. In contrast to theoretical algorithms, most practical methods for computing edge-orientations on dynamic graphs maintain an integral orientation. We now discuss some experimentally evaluated approaches. Brodal and Fagerberg [9] were the first to consider the problem in the dynamic case. With a bound c on the arboricity of the graph as input, their approach supports adjacency queries in time $O(c)$, edge insertions in amortized time $O(1)$, and edge deletions in amortized time $O(c + \log n)$ time.

Berglin and Brodal [5] allow a *worst-case* user-specific trade-off between out-degree and running time of the operations. Their algorithm maintains an $O(\alpha + \log n)$ orientation in $O(\log n)$ worst-case time or an $O(\alpha \log^2 n)$ orientation in constant worst-case time. Borowitz et al. [6] experimentally evaluated different heuristics and approximation algorithms. This evaluation includes new approaches presented by the authors, which are based on a BFS strategy as well as the algorithm by Brodal and Fagerberg [9] and the K-Flip algorithm introduced by Berglin and Brodal [5]. In this experimental evaluation, the BFS approach performed best. Recently, Großmann et al. [27] presented a dynamic algorithm to solve the problem exactly. The worst-case complexity of this method for insertions is $O(m)$, while for deletion it is amortized $O(m)$. Their experiments show that their exact algorithm achieves up to 32 % lower running times than the inexact BFS approach by Borowitz et al. [6].

Contribution. We engineer an implementation of the recently proposed approximation algorithms by Chekuri et al. [10]. In Section 2.1, we provide crucial details required for an implementation. A straightforward implementation of the algorithms would be quite slow and memory-intensive. This is because [10] defines a large (constant) number of values and pointers to guarantee that all algorithmic operations take theoretically $O(1)$ time. Real-world graphs contain many edges compared to the number of available bits in memory. Storing up to eight integers and pointers per edge is infeasible and a clear bottleneck.

Thus, in Section 3, we present engineering aspects to adapt the algorithm from [10] to reduce space use and considerably improve its efficiency. As a consequence, we obtain two implementations of [10] to maintain a fractional orientation $\overrightarrow{G}$ of a dynamic graph G that simultaneously approximately minimizes both $\texttt{sum}^2(\overrightarrow{G})$ and $\texttt{max}(\overrightarrow{G})$. We note that

these existing algorithms maintain an integral orientation $\overrightarrow{G}^*$ and are designed to minimize $\texttt{max}(\overrightarrow{G}^*)$. To provide a fair comparison between our implementation and the current state-of-the-art, we convert our fractional orientation $\overrightarrow{G}$ to an integral orientation through the naive rounding scheme. Experiments in Section 4 show that our implementations offer a considerable speedup of up to 112.17 times whilst being very competitive on the measure $\texttt{max}(\overrightarrow{G}^*)$. Our algorithm performs considerably better than previous implementations when using $\texttt{sum}^2(\overrightarrow{G}^*)$ as the quality metric. Our implementation has a natural parameter λ, which allows for a trade-off between algorithmic efficiency and the quality of the solution.

2 Preliminaries

Let $G = (V, E)$ be a dynamic, unweighted, undirected graph subject to edge insertion and edge deletion. We say that $|V| = n$ at the time of an update. A *fractional* orientation $\overrightarrow{G}$ of G assigns for every edge $\{u, v\}$ two non-negative weights: $d(u \to v)$ and $d(v \to u)$ such that $d(u \to v) + d(v \to u) = 1$. The *out-degree* of a vertex u is defined as $d^+(u) = \sum_{v \in V} d(u \to v)$. An *integral orientation* is a fractional orientation where for all $\{u, v\} \in E$, $d(u \to v) \in \{0, 1\}$. To distinguish between integral and fractional orientations, we denote by $d^*(u)$ the out-degree of a vertex u if the orientation is integral. In integral orientations, we reference directed edges as $\overrightarrow{uv}$ and $\overrightarrow{vu}$. A naive rounding scheme converts each fractional orientation $\overrightarrow{G}$ into an integral orientation $\overrightarrow{G}^*$ where for all $v \in V$, the out-degree $d^*(v)$ is at most $2d^+(v)$.

Quality Measures. We denote by $\mathbb{O}(G)$ the infinite-size set of all fractional orientations of G. For any orientation $\overrightarrow{G} \in \mathbb{O}(G)$ we define two quality measures:

$$\texttt{max}(\overrightarrow{G}) := \max_{v \in V} d^+(v), \qquad \text{and,} \qquad \texttt{sum}^2(\overrightarrow{G}) := \sum_{v \in V} \left(d^+(v)\right)^2.$$

These measures induce respectively a linear and a convex optimization problem over $\mathbb{O}(G)$:

$$\Delta^{\min}(G) := \min_{\overrightarrow{G} \in \mathbb{O}(G)} \texttt{max}(\overrightarrow{G}), \qquad \text{and,} \qquad \Delta^{local}(G) := \min_{\overrightarrow{G} \in \mathbb{O}(G)} \texttt{sum}^2(\overrightarrow{G}).$$

Problem Statement. State-of-the-art edge orientation implementations use integral orientations. We therefore maintain for a dynamic unweighted graph G either:

- an integral orientation $\overrightarrow{G}^*$ that approximately minimizes $\texttt{max}(\overrightarrow{G}^*)$, or,
- an integral orientation $\overrightarrow{G}^*$ that approximately minimizes $\texttt{sum}^2(\overrightarrow{G}^*)$

To this end, we dynamically maintain a fractional orientation $\overrightarrow{G}$ and apply naive rounding.

Fairness. Sawlani and Wang [40] say that $\overrightarrow{G} \in \mathbb{O}(G)$ is *locally fair* whenever $d(u \to v) > 0$ implies that $d^+(u) \leq d^+(v)$. They prove that if $\overrightarrow{G}$ is locally fair, then $\texttt{max}(\overrightarrow{G}) = \Delta^{\min}(G)$. Christiansen, van der Hoog, and Rotenberg [13] show an even stronger correlation between fair orientations and the sum-of-squares measure as $\overrightarrow{G}$ is locally fair if and only if $\texttt{sum}^2(\overrightarrow{G}) = \Delta^{local}(G)$. A locally fair orientation cannot be maintained without requiring $\Omega(n)$ update time. Hence, several works relax the notion of fairness to approximate the local fairness and thereby $\Delta^{\min}(G)$ and $\Delta^{local}(G)$ [4, 7, 8, 13, 20, 40, 45]. For a fixed $\theta \in \{0, 1\}$ and $\lambda \in [0, 1)$, Chekuri et al. [10] define $\overrightarrow{G}$ as λ-fair if: $d(u \to v) \Rightarrow d^+(u) \leq (1 + \lambda) \cdot d^+(v) + \theta$.
If we subsequently smartly choose $\lambda \in \Theta\left(\frac{\varepsilon^2}{\log n}\right)$ and $\theta = 0$, then any λ-fair fractional orientation $\overrightarrow{G}$ guarantees that $\texttt{max}(\overrightarrow{G})$ is a $(1 + \varepsilon)$-approximation of $\Delta^{\min}(G)$ [10] and that

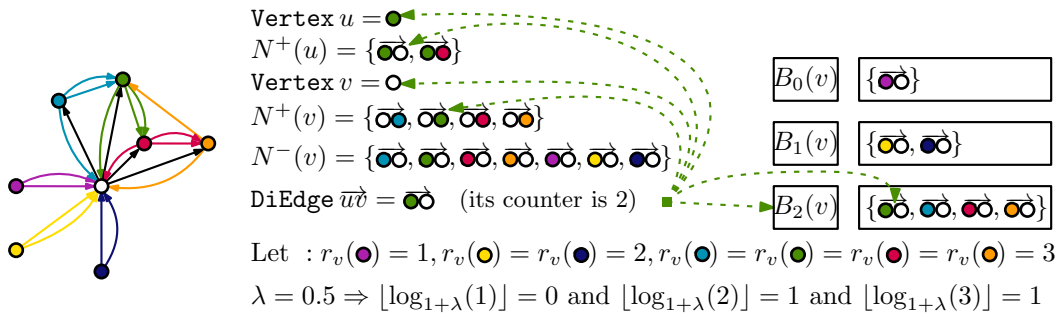


Figure 1 A graph G_b for $b = 2$ and an orientation G_b^*. We illustrate the data structure from [10]. Observe that the `DiEdge` $\overrightarrow{uv}$ stores six pointers in this data structure to allow constant-time updates.

$\texttt{sum}^2(\overrightarrow{G})$ is a $(1+\varepsilon)$-approximation of $\Delta^{local}(G)$ [13]. If $\theta = 1$, the λ-fair orientation may be integral instead, which is considerably easier to maintain. Setting $\theta = 1$ does come at a cost in quality, as the approximation becomes an additive error of $O(\log n)$.

2.1 The Algorithm for Maintaining a λ-fair Orientation by [10]

Let $\lambda \in [0, 1]$ be some quality parameter that the user fixed beforehand. Then, the algorithm by Chekuri et al. [10], on a high level, inserts (or deletes) directed edges in G_b^* (which is an integral orientation over the graph G where each edge is duplicated b times). They orient the edges in such way that they maintain an (integral) λ-fair orientation. After inserting a directed edge $\overrightarrow{uv}$, the out-degree of u increases. So, there may exist some edge $\overrightarrow{ux}$ for which $d^*(u) > (1+\lambda)d^*(x)$. The algorithm then flips this edge, restoring the out-degree of x, and recurses on the edge-insertion $\overrightarrow{xu}$. Instead of spending $O(d^*(u))$ time to inspect all out-edges x of u, the authors show that it suffices to loop over a fraction of the out-edges of u in round-robin fashion. To make all supporting operations in this algorithm run in constant time, the algorithm uses elaborate data structures, data types, and information updates. We explain these first.

For $b \in \mathbb{N}$, denote by G_b the graph G where every edge is duplicated b times. Each edge insertion in G thereby inserts b edges into G_b. We maintain an integral orientation G_b^* over G_b where $d_b^*(v)$ denotes the out-degree. G_b^* induces a fractional orientation over G and we obtain an integral orientation G^* of G from G_b^* by naively rounding each edge.

We denote for all $u \in V$ by $N^+(u)$ all vertices v for which G_b^* has at least one edge $\overrightarrow{uv}$ (see Figure 1). The set $N^-(u)$ are all vertices v where $\overrightarrow{G_b}$ includes at least one edge $\overrightarrow{vu}$. Note that $d_b^*(u)$ is not necessarily equal to $|N^+(u)|$ because G_b is a multigraph.

Information Updates. The update algorithm to G_b^* implements a lazy information update scheme where adding an edge $\overrightarrow{uv}$ to G_b^* increments a counter recording the number of edges $\overrightarrow{uv}$ that G_b^* contains without updating all auxiliary information and data structures.

Formally we define an *information update* (we forward reference Algorithm 3). The algorithm may invoke an information along a directed edge.

▶ **Definition 1.** *Let $d_b^*(w)$ be the out-degree of w at the time of the information update. We define the* log-degree $r_u(w)$ *as the floor of the base $(1+\lambda)$ logarithm of $d_b^*(w)$* at the time of an information update along $\overrightarrow{wu}$. *Or formally: $r_u(w) = \lfloor \log_{1+\lambda} d_b^*(w) \rfloor$.*

This value is recorded such that each directed edge $\overrightarrow{wu}$ in G_b^* has access to the value $r_u(w)$.

Data Structure. Chekuri et al. [10] equip the oriented graph G_b^* with a data structure. They store for each vertex $u \in V$ the following (see also Figure 1):

(a) The *exact value* of $d_b^*(u)$.
(b) The set $N^+(u)$ in a linked list and a pointer some current "position" in the linked list.
(c) The set $N^-(u)$ in a doubly linked list of buckets $B_j(u)$ sorted by j from high to low. Each bucket $B_j(u)$ contains a doubly linked list of all $w \in N^-(u)$ where $r_u(w) = j$.
(d) A pointer to the bucket $B_j(u)$ where j is closest to $\log_{1+\lambda} d^+(u)$.

Data Types. To guarantee that all data structures have $O(1)$ update time, the proofs in [10] reference several pointers stored in their data types. We formalise these here:

`Bucket` $B_i(v)$ stores a doubly linked list with all `DiEdges` $\overrightarrow{uv}$ with $i = r_v(u)$.

`Vertex` u stores:
- a linked list of all `DiEdges` $\overrightarrow{uv}$ (corresponding to vertices $v \in N^+(u)$).
- a doubly linked list of all non-empty `Buckets` $B_i(u)$, ordered by i.
- a pointer to the `DiEdge` *robin* which represents our "current position" in $N^+(u)$.
- a pointer to the *primary bucket* $B_j(u)$ (here j is closest to $\log_{1+\lambda} d^+(u)$).
- the value $d_b^*(u)$.

`DiEdge` $\overrightarrow{uv}$ represents all directed edges $\overrightarrow{uv}$ in G_b^*, simultaneously. It stores:
- two pointers to the `Vertices` u and v, respectively.
- a *counter* counting how many edges $\overrightarrow{uv}$ there are in G_b^*,
- the value $r_v(u)$,
- a pointer to the location of $\overrightarrow{uv}$ in $N^+(u)$.
- a pointer to the bucket $B_i(v)$ of v with $i = r_v(u)$,
- a pointer to the location of $\overrightarrow{uv}$ in the linked list in the bucket $B_i(v)$.
- a pointer to the `DiEdge` $\overrightarrow{vu}$.

2.1.1 Algorithm Definition

The algorithm has three parameters that are used throughout its algorithmic subroutines: $\theta \in \{0, 1\}$, $\lambda \in [0, 1]$ and $b \in \mathbb{N}$. We explain their interplay later in this subsection. Each update operation in G simply invokes b updates to the digraph G_b^* (see Algorithm 1 and 2).

Algorithm 1 Insert(edge $\{u, v\}$ in G).

```
1  for i ∈ [b] do
2    if d*_b(u) ≤ d*_b(v) then
3      Insert(uv→)
4    else
5      Insert(vu→)
```

Algorithm 2 Delete(edge $\{u, v\}$ in G).

```
for i ∈ [b] do
  if u ∈ N⁻(v) then
    Delete(vu→)
  else
    Delete(uv→)
```

Recall that the algorithm does not always immediately update the data structure. Rather, it only exactly maintains the value $d_b^*(u)$ for each vertex u and the set $N^+(u)$. The bucketing data structure is updated only when an *information update* (Algorithm 3) is triggered.

Algorithm 3 InfoUpdate(`DiEdge` $\overrightarrow{xy}$, `Bucket` $B_j(y)$).

```
6  r_y(x) = log_{1+λ} d*_b(x)
7  Starting from B_j(y) find the bucket B_i(y) with i = r_y(x) Inspect at most O(1) buckets
8  Move xy→ into the bucket B_i(y)
```

Suppose that we add a directed edge $\overrightarrow{uv}$ to G_b^*. If there already exists an edge $\overrightarrow{uv}$ in $\overrightarrow{G_b}$ then we implicitly insert the extra copy by identifying the corresponding `DiEdge` $\overrightarrow{uv}$ and incrementing the counter. Otherwise, we need to update $N^+(u)$ and do an information update. The same principle holds when removing a directed edge, where we only do work if we remove the last copy of a directed edge in G_b^* (Algorithm 4 and 5).

Algorithm 4 Add(`DiEdge` $\overrightarrow{uv}$).

```
9   Increment the counter of uv.
10  d*_b(u) = d*_b(u) + 1
11  if uv.counter = 1 then
12    Add uv at the end of N+(u)
13    B_j(v) ← the primary Bucket of v
14    InfoUpdate(uv, B_j(v))
```

Algorithm 5 Remove(`DiEdge` $\overrightarrow{uv}$).

```
    Decrement the counter of uv
    d*_b(u) = d*_b(u) − 1
    if uv.counter = 0 then
      Remove v from N+(u)
      B_i(v) ← the Bucket stored at uv
      InfoUpdate(uv, B_i(v))
```

The true complexity of this algorithm lies in the logic surrounding directed edge insertions and deletions. Intuitively, inserting an edge $\overrightarrow{uv}$ iterates over the out-edges $\overrightarrow{ux}$ of u in a round-robin fashion. If the out-degree of u is considerably larger than the out-degree of x, we flip an edge $\overrightarrow{ux}$ to become an edge $\overrightarrow{xu}$, we terminate the round-robin and recurse on the insertion of $\overrightarrow{xu}$. After this flip, the out-degree of u is unchanged and thus locally, the orientation is still λ-fair. This increased the out-degree of x by one, and we recurse on x (Algorithm 1). Deleting an edge $\overrightarrow{uv}$ only considers a single directed edge, which is an edge $\overrightarrow{xu}$ where the out-degree $d_b^*(x)$ is approximately maximal. If $d_b^*(u)$ is considerably larger than $d_b^*(u)$ then we flip an edge $\overrightarrow{xu}$. After this flip, the out-degree of u is unchanged. Instead, the out-degree of x is decreased by one, and we recurse on x (Algorithm 1).

Finally if, after either an edge insertion or deletion, a vertex u actually changes its out-degree then the algorithm will inform up to $\frac{2}{\lambda}$ out-neighbours of u of its new out-degree.

Algorithm 6 Insert(`DiEdge` $\overrightarrow{uv}$).

```
15  Add(uv)
16  for ⌈2/λ⌉ iterations do: do
17    u.robin = (u.robin + 1) mod |N+(u)|
18    DiEdge ux = *u.robin
19    if d*_b(u) > max{(1 + λ) · d*_b(x) + θ, b/4} then
20      Remove(ux)
21      Insert(xu)
22      Break
23  for all the next ⌈2/λ⌉ uw in N+(u) do
24    InfoUpdate(uw, uw.Bucket)
```

Algorithm 7 Delete(`DiEdge` $\overrightarrow{uv}$).

```
    Remove(uv)
    x ← First( Max( Bucket(N−(u))))
    if d*_b(x) > max{(1 + λ) · d*_b(u) + θ, b/4}
    then
      Add(ux)
      Delete(xu)
    else
      for the next ⌈2/λ⌉ uw in N+(u) do
        InfoUpdate(uw, uw.Bucket)
```

Algorithm Parameters and Asymptotic Complexity. The algorithm has three parameters:

- $\theta \in \{0, 1\}$ is a Boolean that is used in the logic of Algorithms 6 and 7. The Boolean specifies whether we want a slower $(1+\varepsilon)$-approximation (setting $\theta \leftarrow 0$), or, whether we want a faster algorithm that achieves an additive $O(\log n)$-error (setting $\theta \leftarrow 1$).
- $\lambda \in [0, 1]$ is the approximation parameter.
  - If $\theta = 0$, then we may choose $\lambda \in \Theta(\frac{\varepsilon^2}{\log n})$ to obtain a $(1+\varepsilon)$-approximation [10, 13].
  - If $\theta = 1$, then we may choose $\lambda \in \Theta(\frac{1}{\log n})$ to obtain an additive $O(\log n)$ error [10, 13].
- $b \in \mathbb{N}$ records how many edges the multigraph G_b has per edge in G. If $\theta = 1$ then $b = 1$. If $\theta = 0$ then $b > \frac{4}{\lambda}$ in order to make the algorithmic logic work. Indeed, consider an edge $\{u, v\}$ in G. Our algorithm maintains the invariant that if $\overrightarrow{uv}$ is in G_b^* then $d_b^*(u) \leq (1+\lambda)d_b^*$. If both $d_b^*(u)$ and $d_b^*(v)$ are less than $\frac{1}{\lambda}$ then there is no way to orient $\{u, v\}$ in G_b^* such that our invariant is satisfied.

Each update in G triggers at most b updates in G_b. Each update to G_b calls Algorithm 6 or 7. These functions do at most $O(\frac{1}{\lambda})$ constant-time operations before recursing. These functions recurse at most $O(\log_{1+\lambda} n)$ times. It follows that the total runtime is at most:

$$O(b \cdot \frac{1}{\lambda} \cdot \log_{1+\lambda} n) \subset O(b\lambda^{-2} \log n) \Rightarrow \begin{cases} O(\lambda^{-3} \log n) & \text{if } \theta = 0 \\ O(\lambda^{-2} \log n) & \text{if } \theta = 1 \end{cases}$$

In theory, one chooses $\lambda \in \Theta(\varepsilon^2 / \log n)$ where ε is the approximation constant. In practice, we simply choose a smaller λ for a better approximation.

Space Usage. In theory, the data type as we specified in this paper allows each of the operations in Algorithms 1–7 to take constant time. We give three examples: (i) we can flip a directed edge in G_b^* in constant time because the `DiEdge` $\overrightarrow{ux}$ stores a pointer to the `DiEdge` $\overrightarrow{xu}$, (ii) a `DiEdge` $\overrightarrow{uv}$ can remove itself from the linked list $N^+(u)$ in constant time because it stores a pointer to its location, and (iii) a `DiEdge` $\overrightarrow{uv}$ can remove itself from the doubly linked list $B_i(v)$ of $N^-(v)$ in constant time because it stores a pointer to its location.

While theoretically efficient, this solution faces a practical bottleneck due to high space usage. Indeed, it stores six pointers and two integers per edge in the original graph G.

Moreover, the excessive use of pointers makes executing the algorithm less predictable and slowing it down in practice. Additionally, each pointer is not efficient in addressing only a part of the memory, i.e. the out-going edges, since it requires constant (64-bit) memory.

3 Engineering Aspects

We now engineer an efficient implementation of [10]. Our key observation is that there are two data containers where the order of the elements in the container is irrelevant:

- `Vertex` u stores all `DiEdges` $\overrightarrow{uv}$ in arbitrary order, and
- `Bucket` $B_i(v)$ stores all `DiEdges` $\overrightarrow{uv}$ with $i = r_v(u)$ in arbitrary order.

By sorting these sets as arrays, we obtain more control over the size of these objects and the size of pointers to elements in these sets. This in turn allows us to use packing techniques to store the same information in fewer bits.

Arrays. This allows us to use arrays instead of linked list as we give each:

- `Vertex` u an array A_u^+ with all `DiEdges` $\overrightarrow{uv}$ in arbitrary order.
- `Bucket` $B_i(v)$ an array B_i^v with all `DiEdges` $\overrightarrow{uv}$ with $i = r_v(u)$ in arbitrary order.

Adding a `DiEdge` $\overrightarrow{uv}$, given the `Vertex` u and the correct `Bucket` $B_i(v)$, takes amortized constant time. Indeed, we simply increase the sizes of A_u^+ and B_i^v and append $\overrightarrow{uv}$. Removing $\overrightarrow{uv}$ also takes amortized constant time: given its index a in A_u^+ (and b in B_i^v) we may swap $A_u^+[a]$ with the last element of A_u^+ (swap $B_i^v[b]$ with the last element) and decrement the array's size by one. These operations are supported by the `vector` data type in C++.

From Arrays to Packing. Replacing doubly linked lists with vectors naturally reduces algorithmic overhead. It also has a secondary benefit: instead of maintaining, for a `DiEdge` $\overrightarrow{uv}$, pointers to its location in linked lists we store integers maintaining its index in vectors. Recall that we maintain an orientation G_b^* where the maximum out-degree $\texttt{Max}(G_b^*)$ is low. As a consequence, for any vertex v, the array A_u^+ is small. Thus, for a `DiEdge` $\overrightarrow{uv}$ its index in A_u^+ is small. Via the same logic, the integers *robin*, $d_u^*(u)$, $\lfloor \log_{1+\lambda}(r_u(v)) \rfloor$ and the index of the `DiEdge` $\overrightarrow{vu}$ are small. We pack these small values into 32-bit integers.

Indirection. Any `DiEdge` $\overrightarrow{uv}$ contained pointers to: the `Vertex` u, the `Vertex` v, and the `DiEdge` $\overrightarrow{vu}$. If each `DiEdge` only stores a pointer to its target then we can access the `Vertex` u from $\overrightarrow{uv}$ by first accessing the `DiEdge` $\overrightarrow{vu}$ and then reading its target.

Removing Bucket Pointers. Each `Vertex` u maintained a pointer to the primary bucket $B_j(u)$ and each `DiEdge` $\overrightarrow{uv}$ maintained a pointer to the bucket $B_i(v)$ that contains $\overrightarrow{uv}$. These pointers are only used in Algorithm 3. Chekuri et al. [10] prove that to find for a `DiEdge` $\overrightarrow{xy}$ is current bucket, one steps over at most $O(1)$ buckets from the given pointer. We choose to remove these pointers. Instead, we find the correct bucket for $\overrightarrow{xy}$ by looping over all buckets $B_i(y)$. Observe that there are at most $O(\log_{1+\lambda} n) \subseteq O(\lambda^{-1} \log n)$ buckets $B_i(y)$. Moreover, a call to Algorithm 6 or 7 triggers $O(\lambda^{-1})$ information updates *only if* it does not recurse. So, the asymptotic running time remains the same.

Compact Types. By our above analysis, we obtain more compact data types (see Figure 2):

`Bucket` $B_i(u)$ stores a vector B_i^u with `DiEdges` $\overrightarrow{wu}$ with $i = r_u(w)$.

`Vertex` u stores:

- a vector A_u^+ storing all `DiEdges` $\overrightarrow{uv}$ (corresponding to vertices $v \in N^+(u)$).
- a linked list of all non-empty `Buckets` $B_i(u)$, ordered by i.
- a 32-bit integer storing our "current position" *robin* in A_u^+ and the value $d_b^*(u)$.

`DiEdge` $\overrightarrow{uv}$ represents all directed edges $\overrightarrow{uv}$ in G_b^*, simultaneously. It stores:

- A pointer to the `Vertex` v.
- A 32-bit integer storing the *counter*, the value $r_v(u)$, and the index of $\overrightarrow{vu}$ in A_u^+.
- An 32-bit integer storing the index of $\overrightarrow{uv}$ in B_i^v for $i = r_v(u)$.

3.1 A Space-Time Trade-off

Finally, we propose one more modification to the algorithm. The algorithm in [10] and our above implementation store for a vertex v all non-empty buckets $B_i(v)$ in a linked list. The algorithm in [10] accesses these buckets by storing a pointer for each edge in G. In our implementation, we find the bucket $B_i(v)$ where $i = r_v(u)$ in $O(\log_{1+\lambda} n)$ time by iterating over all buckets $B_j(v)$. We showed that that this does not increase the asymptotic running time. However, it does increase the *actual* running time of the program.

Both methods have an inherent downside: [10] uses a lot of memory per edge, and our above implementation is relatively slow. Moreover, doubly linked list inherently bring a lot of overhead and cache misses. We therefore propose a third solution to store the buckets:

We define for a `Vertex` v the *Bucket array* $\mathbb{B}_v$ such that for all i, $\mathbb{B}_v[i] = B_i(v)$. Any `DiEdge` $\overrightarrow{uv}$ can access the bucket $B_i(v)$ where $i = r_v(u)$ by simply indexing $\mathbb{B}_v[r_v(u)]$ in constant time. By storing the array $\mathbb{B}_v$ as a vector, we require only a relatively small amount

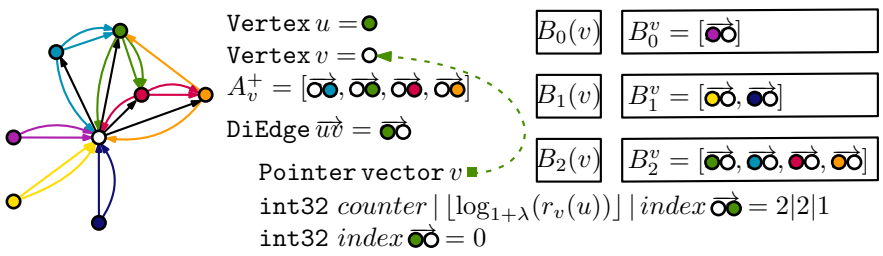


Figure 2 We illustrate our more compact data structure compared to the one from Figure 1.

of memory for every empty bucket. There are $O(\log_{1+\lambda}(n))$ buckets for each vertex v, so this additively increases our space usage by $O(\frac{n}{\lambda}\log n)$. Since the number of edges m is considerably larger than the number of vertices n, this is space-efficient when compared to [10]. This solution does use more space compared to our above implementation, with the benefit of having a significantly faster implementation of Algorithm 3. Thus, we offer two implementations – one using singly linked lists, and one using this array data structure – that together provide a trade-off between time and space used.

4 Experimental Evaluation

In this section we evaluate our algorithm and compare its performance with previous work by Großmann et al. [27] and Borowitz et al. [6]. We compare them on the sparse data set introduced in [6] and present a new dense data set featuring also deletions. The analysis is done in separate sections (sparse, dense) since the performance of the various algorithms highly depend on the density of used inputs.

Methodology. We carried out the implementation of the algorithms in C++ and compiled the programs with g++-11.4. We tested each algorithm on a machine equipped with 768 GB of RAM and an AMD EPYC 9754 128-Core Processor running at 2.25 GHz having a cache of 256 MB. The time loading the graphs is not measured, but instead we measure every operation and sum up the timings. We repeated every experiment 3 times, computations taking longer than a day in total were cancelled and are marked as out of time (OOT) in the result tables. For comparing the results we use performance profiles as introduced by Dolan and Moré [17]. For a factor of $\tau > 1$ we plot the fraction of instances that could be solved within that factor of the best result per instance. Algorithms in the top left corner of the plot (e.g. Figure 3) are considered the best algorithm.

Instances. We use two sets of graphs for the experiments. First, the *sparse set* containing 83 instances collected by Borowitz et al. [6]. Only four of the instances feature deletions. The size of the graphs is small to medium and they have low-density and out-degree. Second, the *dense set*, that contains three, large, denser graphs sourced from the Suite Sparse Matrix collection [15] and four ultra-dense graphs generated with KaGen [22]. See also our supplemental details. The ultra-dense graphs were generated in the Erdos-Renyi model having 8192 vertices and between $\frac{1}{2}$ and $\frac{7}{8}$ of all possible edges. The graphs and their features are listed in the full version. For these instance we generated three different update sequences. The first, building up the graph in lexicographical ordering (Lex). The other two also include deletion of edges afterwards. The edges to be deleted are selected based on an optimal orientation of the final graph, computed with HeiOrient [38]. In the Lex+Top50 configuration all edges outgoing from the upper half of vertices ordered by out-degree are deleted.

The Lex+Out50 configuration is built by maintaining a heap of the vertices by out-degree. Outgoing edges are deleted and the out-degree is updated in the heap until the maximum out-degree is half of the optimal orientation. We select these sequences in order to see how different deletion strategies impact the quality, memory and runtime metrics.

Comparisons and Implementations. We refer to our basic implementation of the algorithm by Chekuri et al. [10] as Fractional and with our engineering improvements as PackedFractional. We provide two implementations of PackedFractional. One with PackedFractional list organizes

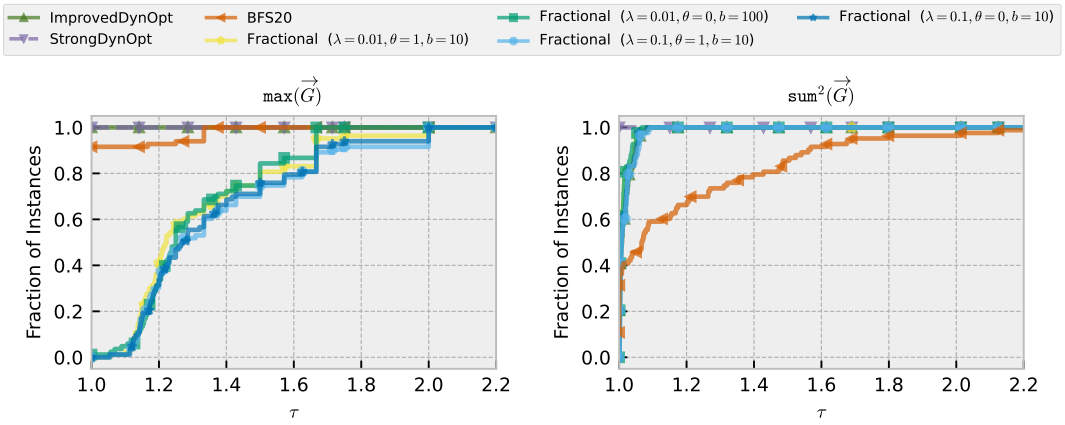


Figure 3 State-of-the-art comparison with different quality measures on the sparse instance set. The PackedF* is omitted due to its similar results as Fractional with a maximum difference of 0.3%.

the in-buckets in a singly linked list, and the one without suffix works with directly addressable, possibly empty, buckets. We compare with BFS20 by Borowitz et al. [6] and StrongDynOpt and ImprovedDynOpt by Großmann et al. [27]. These algorithms are the best performing in the extensive studies in [6] and [27] with regards to the max out-degree metric $\max(\overrightarrow{G})$. The $\text{sum}^2(\overrightarrow{G})$ metric has not been studied in practice. The BFS20 algorithm performs a limited breadth-first-search of length up to 20 to find improving paths upon insertion and removes only the edge on deletion, not optimizing the orientation. The StrongDynOpt runs a breadth-first-search to find improving paths on every insertion and deletion, returning the optimal orientation concerning out-degree and squared-sum of out-degree. The ImprovedDynOpt only searches for improving paths if it is necessary to maintain the optimal out-degree.

Parameters. The algorithms Fractional and PackedFractional derived in Section 2.1 and Section 3 have the three parameters: λ, θ, b. First, $\lambda > 0$ and $\theta \in \{0, 1\}$ control the quality of the approximation by defining its λ-fairness. If $\theta = 0$, b must be larger than $\frac{1}{\lambda}$ so that the fractional orientation inequality can be satisfied. Otherwise ($\theta = 1$), b can be set to 1 making the orientation integral. This comes at costs of quality in the form of an additive error of $O(\log n)$ in the approximation.

4.1 Experiments on Sparse Instances

The following experiments are conducted on the sparse instance set collected by Borowitz et al. [6] and analyse the running times and quality of the orientations. Since some of graphs in this set were too small to consistently measure the maximum resident memory by repeatedly querying the linux internal process stats, we examine the memory consumption only for the dense instances. The maximum peak memory is 23 GB for the Fractional and PackedFractional with $\lambda = 0.01$ and $\theta = 0$ implementations on the LJOURNAL-2008 instance. The PackedFractional list algorithm with $\lambda = 0.1$, $\theta = 1$ only requires 4.4 GB on this instance - only 2.29 times the amount of StrongDynOpt.

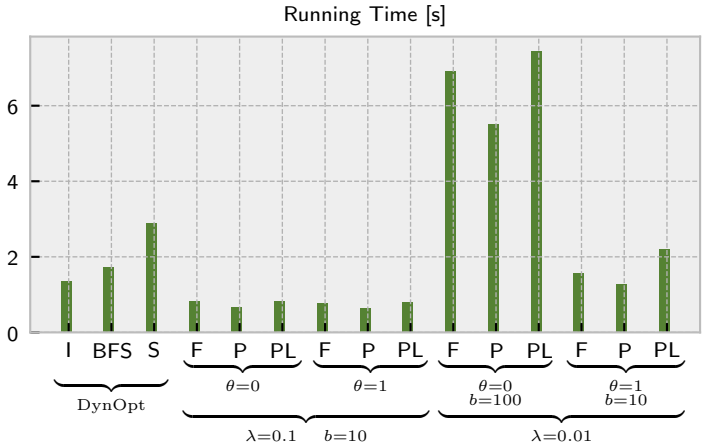


Figure 4 Running times on the sparse set comparing Fractional (F), PackedFractional (P), and PackedFractional, list (PL) with BFS20 (BFS), ImprovedDynOpt (I), and StrongDynOpt (S). Parameters: λ overall fairness of the orientation maintained. θ is a binary variable deciding an additional $O(\log n)$ error, b is the number of fractional edges.

Quality Metrics. In Figure 3 we compare different configurations of Fractional with the state-of-the-art for integral orientations. We evaluate the performance using the two quality measures discussed in Section 2. For better readability, the results for the packed implementations are omitted. They achieve similar to those results of the Fractional configurations with a maximum difference in quality of less than 0.3%. We can observe that smaller values of λ lead to improved solution quality for both metrics.

Maximum Out-Degree. When comparing the algorithms using the metric $\max(\overrightarrow{G})$, we can observe that both exact approaches ImprovedDynOpt and StrongDynOpt always find the optimum solution. Compared to these, Fractional performs worse overall, but maintains a factor of less than two for all parameter choices.

Squared Sum of Out-Degrees. When evaluating with $\text{sum}^2(\overrightarrow{G})$ of the naively rounded fractional orientations, Fractional performs significantly better than the BFS20 algorithm, matching the results of ImprovedDynOpt. The StrongDynOpt algorithm returns the best values on every instance. This is the expected behaviour, because improving all possible paths minimizes both objective functions. Our newly engineered algorithms perform very well in the $\text{sum}^2(\overrightarrow{G})$ metric as they were designed for it. The variation in $\lambda \in \{0.01, 0.1\}$ is not visible in this metric. In exchange, they allow themselves a higher maximum out-degree on some of the vertices which is not accounted for in the definition of λ-fair.

Running Time. Figure 4 shows the geometric mean running time for the approaches on the sparse data set. The normalized running times can be found in the full version. Our fastest algorithm PackedFractional is twice as fast as ImprovedDynOpt, which is the fastest state-of-the-art method on this data set. The running time of our approach is highly dependent on the parameter choices. Setting $\theta = 1$ is faster than $\theta = 0$. Decreasing λ results in longer running times. The smallest value $\lambda = 0.01$ we tested is 8.58 times slower than the fastest configuration with $\lambda = 0.1$. The packed list implementation is up to 35 % slower. The measured difference between ImprovedDynOpt and BFS20 matches the result in [27].

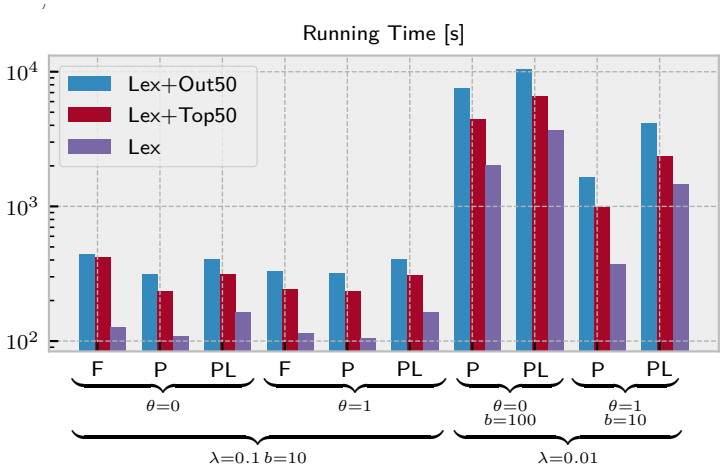


Figure 5 Geometric mean running times on the dense set comparing Fractional (F), PackedFractional (P), and PackedFractional, list (PL). We evaluate them for the different deletion sequences.

4.2 Experiments on Dense Instances

We conduct the following experiments on the new dense instance set, which includes three real-world graphs and four ultra-dense graphs. As in previous sections, we analyze running time and solution quality, and additionally provide an in-depth examination of memory usage.

Quality Metrics. Only three out of the 21 instances were solved optimally by StrongDynOpt on the set of dense instances in the given timeout of 24 hours. Therefore, we show performance profiles for our parameter choices and PackedFractional in the full version on all instances and limited the following discussion about quality on those three instances.

Maximum Out-Degree. Evaluating the performance using $\max(\overrightarrow{G})$ the solutions of Fractional are between 24 % ($\lambda = 0.01$) up to 30 % ($\lambda = 0.1$) worse than the optimal orientations. The method BFS20 also performs worse on these instances with solutions being 26 % worse. Note that this approach does not handle deletions explicitly by running a improving path search, which can explain the observed suboptimal results on our instances featuring deletions. These results match the observations of previous work.

Squared Sum of Out-Degrees. Using $\text{sum}^2(\overrightarrow{G})$, we observe a solution quality for PackedFractional which is less than 0.8 % higher for $\lambda = 0.1$ and only 0.3 % higher when setting $\lambda = 0.01$ than the optimum computed by StrongDynOpt. With this, we achieve better results on these instances compared to ImprovedDynOpt, which in the geometric mean has a solution quality that is 15 % worse. The ImprovedDynOpt is specialised in maintaining the minimum maximum out-degree while requiring fewer operations than StrongDynOpt. Therefore, its $\text{sum}^2(\overrightarrow{G})$ metric is worse than our results. The impact of the λ factor, in the range we tested, on the $\text{sum}^2(\overrightarrow{G})$ metric is small for our new algorithms.

Running Time. The running times on the dense set for our implementations are displayed in Figure 5. We detail these results in the full version. Detailed results per instance are shown in the full version. The choice of lambda greatly impacts the running time. Reducing the value of λ by a factor of 10 from 0.1 to 0.01 (keeping $\theta = 0$) can result in running time that is 20 times slower in the geometric mean. Setting $\theta = 1$ speeds up the algorithm by up to a factor of 5 for small $\lambda = 0.01$ compared to $\theta = 0$. The methods from Borowitz et al. [6]

and Großmann et al. [27] find solutions for considerably fewer number of instances in this data set while requiring more running time. More precisely, BFS20, ImprovedDynOpt, and StrongDynOpt find solutions on 15, 5, and 3 instances while having a geometric mean running time which is 112.17, 29.15, and 25.98 times slower than our fastest algorithm (PackedFractional) on the respective instances.

Memory. As a smaller λ value results in more buckets, we expect the memory consumption to increase with decreasing λ. This behaviour can be observed in Figure 6. Here, we compare the memory consumption of our implementations, which is evaluated separately for the different deletion sequences. Already for $\lambda = 0.1$, we can observe that the PackedFractional is improving over Fractional, where we achieve a geometric mean reduction in memory consumption by a factor of 1.94. In the case of $\lambda = 0.01$, we omit Fractional due to its already high memory consumption for $\lambda = 0.1$ and focus on comparing the implementations PackedFractional and PackedFractional, list. With this configuration, the List implementation can save up to a factor of 2.72 memory ($\lambda = 0.01, \theta = 0, b = 100$). Overall, we can see that the choice for λ has the highest impact regarding memory consumption. However, the parameters θ and b also influence the memory needed. The memory consumption for the competitor algorithms is generally smaller. Compared to our best approach, the competitor methods perform up to a factor of 3.32 better. However, this result is expected since they do not maintain a fractional orientation but an integral one. They store each edge's target on both sides in combination with the position of the reverse edge in the other array. Lastly, we observe that the deletion of edges (Top50, Out50) requires more memory than just the Lex ordering, with the Out50 requiring the most. This is caused by the spreading of indices in the in-bucket, as seen by the smaller difference in the list implementations.

Practitioner Advice: When optimizing the sum of squared out-degrees we recommend using the packed PackedFractional implementation since it is the fastest implementation overall and the difference in quality to the exact method (StrongDynOpt) is minimal. The list implementation should only be used if memory consumption is of utmost importance. The choice of parameters is paramount. Choosing higher values for λ speeds up the computation and reduces the memory needed. Setting $\theta = 1$ can be used to relax the running time further, but is penalized by a quality trade-off. The b parameter should be chosen small to speed up computation, but has to stay above $\frac{1}{\lambda}$ if $\theta = 0$.

5 Conclusion

We have bridged the gap between theoretical results and practical experiments for adaptive edge orientations. We engineered a space efficient implementation of a λ-fair fractional orientation algorithm by Chekuri et al. [10] and presented several effective methods to reduce the memory footprint of the algorithm. In extensive experiments we show the feasibility of our approach in comparison with several competitors from previous work. Our results for the squared sum degree metric are close to optimal, while the running time is considerably lower than competing algorithms. Our algorithm PackedFractional solves more instances than the competitors within one day of compute time per instance while it is up to 112 times faster on the instances that are solvable by all methods compared. Our list-based bucket implementations can decrease the memory consumption by a factor of 2.72. Future avenues of work include reducing the memory costs further and looking at batch updates. These aspects could be combined with efforts to parallelize the algorithms.

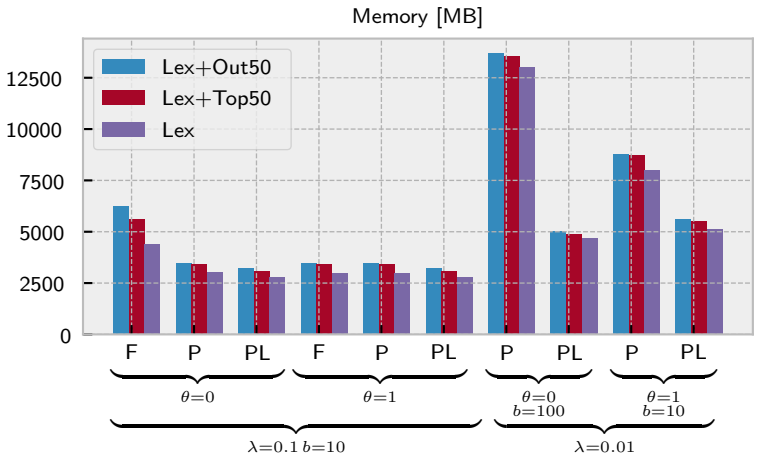


Figure 6 Memory consumption of Fractional (F), PackedFractional (P), and PackedFractional, list (PL) for different configurations evaluated separately for the different deletions sequences.

References

1 J. Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, volume 18, pages 41–50, 2005. URL: `https://proceedings.neurips.cc/paper/2005/hash/b19aa25ff58940d974234b48391b9549-Abstract.html`.

2 Albert Angel, Nick Koudas, Nikos Sarkas, Divesh Srivastava, Michael Svendsen, and Srikanta Tirthapura. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *The VLDB journal*, 23:175–199, 2014. `doi:10.1007/s00778-013-0340-z`.

3 Gary D. Bader and Christopher W. V. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinform.*, 4:2, 2003. `doi:10.1186/1471-2105-4-2`.

4 Suman K. Bera, Sayan Bhattacharya, Jayesh Choudhari, and Prantar Ghosh. A new dynamic algorithm for densest subhypergraphs. In Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 1093–1103. ACM, 2022. `doi:10.1145/3485447.3512158`.

5 Edvin Berglin and Gerth Stølting Brodal. A Simple Greedy Algorithm for Dynamic Graph Orientation. *Algorithmica*, 82(2):245–259, 2020. `doi:10.1007/s00453-018-0528-0`.

6 Jannick Borowitz, Ernestine Großmann, and Christian Schulz. Engineering Fully Dynamic Δ-orientation Algorithms. In *ACDA*, pages 25–37, 2023. `doi:10.1137/1.9781611977714.3`.

7 Glencora Borradaile, Jennifer Iglesias, Theresa Migler, Antonio Ochoa, Gordon Wilfong, and Lisa Zhang. Egalitarian graph orientations. *Journal of Graph Algorithms and Applications*, 2017. `doi:10.7155/jgaa.00435`.

8 Glencora Borradaile, Theresa Migler, and Gordon Wilfong. Density decompositions of networks. In *Conference on Complex Networks (CompleNet)*. Springer, 2018. `doi:10.7155/jgaa.00505`.

9 Gerth Stølting Brodal and Rolf Fagerberg. Dynamic Representation of Sparse Graphs. In Frank K. H. A. Dehne, Arvind Gupta, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures, 6th International Workshop, WADS '99, Vancouver, British Columbia, Canada, August 11-14, 1999, Proceedings*, volume 1663 of *Lecture Notes in Computer Science*, pages 342–351, 1999. `doi:10.1007/3-540-48447-7_34`.

10 Chandra Chekuri, Aleksander Bjørn Christiansen, Jacob Holm, Ivor van der Hoog, Kent Quanrud, Eva Rotenberg, and Chris Schwiegelshohn. Adaptive out-orientations with applications. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3062–3088. SIAM, 2024. doi:10.1137/1.9781611977912.110.

11 Jie Chen and Yousef Saad. Dense subgraph extraction with application to community detection. *IEEE Trans. Knowl. Data Eng.*, 24(7):1216–1230, 2012. doi:10.1109/TKDE.2010.271.

12 Aleksander B. G. Christiansen and Eva Rotenberg. Fully-Dynamic $\alpha + 2$ Arboricity Decompositions and Implicit Colouring. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.42.

13 Aleksander Bjørn Christiansen, Ivor van der Hoog, and Eva Rotenberg. Local Density and Its Distributed Approximation. In Olaf Beyersdorff, Michał Pilipczuk, Elaine Pimentel, and Nguyễn Kim Thang, editors, *Theoretical Aspects of Computer Science (STACS)*, volume 327 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:20, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2025.25.

14 Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. Large scale density-friendly graph decomposition via convex programming. In Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, WWW '17, pages 233–242, Republic and Canton of Geneva, CHE, 2017. ACM. doi:10.1145/3038912.3052619.

15 Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, December 2011. doi:10.1145/2049662.2049663.

16 Laxman Dhulipala, Quanquan C Liu, Julian Shun, and Shangdi Yu. Parallel batch-dynamic k-clique counting. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 129–143. SIAM, 2021. doi:10.1137/1.9781611976489.10.

17 Elizabeth D. Dolan and Jorge J. Moré. Benchmarking Optimization Software With Performance Profiles. *Math. Program.*, 91(2):201–213, 2002. doi:10.1007/s101070100263.

18 Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense communities in the web. In *Proceedings of the 16th international conference on World Wide Web*, pages 461–470, 2007. doi:10.1145/1242572.1242635.

19 Xiaoxi Du, Ruoming Jin, Liang Ding, Victor E Lee, and John H Thornton Jr. Migration motif: a spatial-temporal pattern mining approach for financial markets. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2009. doi:10.1145/1557019.1557142.

20 András Frank and Kazuo Murota. Fair integral network flows. *Mathematics of Operations Research*, 2023. doi:10.1287/moor.2022.1303.

21 Eugene Fratkin, Brian T. Naughton, Douglas L. Brutlag, and Serafim Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. In *Proceedings 14th International Conference on Intelligent Systems for Molecular Biology 2006, Fortaleza, Brazil, August 6-10, 2006*, volume 22, pages 156–157. Oxford University Press, 2006. doi:10.1093/bioinformatics/btl243.

22 Daniel Funke, Sebastian Lamm, Ulrich Meyer, Manuel Penschuck, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. Communication-free massively distributed graph generation. *J. Parallel Distributed Comput.*, 131:200–217, 2019. doi:10.1016/J.JPDC.2019.03.011.

23 David Gibson, Jon Kleinberg, and Prabhakar Raghavan. Inferring web communities from link topology. In *Proceedings of the ninth ACM conference on Hypertext and hypermedia: links, objects, time and space-structure in hypermedia systems: links, objects, time and space-structure in hypermedia systems*, pages 225–234, 1998. doi:10.1145/276627.276652.

24 David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on Very large data bases*, pages 721–732, 2005. URL: http://www.vldb.org/archives/website/2005/program/paper/thu/p721-gibson.pdf.

25 Aristides Gionis and Charalampos E Tsourakakis. Dense subgraph discovery: Kdd 2015 tutorial. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2313–2314, 2015. doi:10.1145/2783258.2789987.

26 Ernestine Grossmann, Henrik Reinstädtler, Eva Rotenberg, Christian Schulz, Ivor van der Hoog, and Juliette Vlieghe. Implementation of our dynamic algorithm for minimum orientation. Software, Villum Fonden VIL37507, DFG SCHU 2567/8-1, Marie Skłodowska-Curie 899987, swhId: swh:1:dir:0ea52feafca9f6d8ce3ee893686fde2510513e9e (visited on 2025-09-04). URL: https://github.com/DynGraphLab/DynDeltaApprox, doi:10.4230/artifacts.24670.

27 Ernestine Großmann, Henrik Reinstädtler, Christian Schulz, and Fabian Walliser. Engineering fully dynamic exact-orientation algorithms. In *2025 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 15–28. SIAM, 2025. doi:10.1137/1.9781611978339.2.

28 Patric Hagmann, Leila Cammoun, Xavier Gigandet, Reto Meuli, Christopher J Honey, Van J Wedeen, and Olaf Sporns. Mapping the structural core of human cerebral cortex. *PLoS biology*, 6(7):e159, 2008.

29 Elfarouk Harb, Kent Quanrud, and Chandra Chekuri. Faster and scalable algorithms for densest subgraph and decomposition. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, volume 35, pages 26966–26979, 2022. URL: http://papers.nips.cc/paper_files/paper/2022/hash/ac8fbba029dadca99d6b8c3f913d3ed6-Abstract-Conference.html.

30 Jon M Kleinberg et al. Authoritative sources in a hyperlinked environment. In Howard J. Karloff, editor, *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California, USA*, volume 98, pages 668–677. Citeseer, ACM/SIAM, 1998. URL: http://dl.acm.org/citation.cfm?id=314613.315045.

31 Jon M Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S Tomkins. The web as a graph: Measurements, models, and methods. In *International Computing and Combinatorics Conference*, pages 1–17. Springer, 1999. doi:10.1007/3-540-48686-0_1.

32 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting Fully Dynamic Graphs with Worst-case Time Bounds. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 532–543, 2014. doi:10.1007/978-3-662-43951-7_45.

33 Lukasz Kowalik and Maciej Kurowski. Oracles for Bounded-length Shortest Paths in Planar Graphs. *ACM Trans. Algorithms*, 2(3):335–363, 2006. doi:10.1145/1159892.1159895.

34 Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 611–617, 2006. doi:10.1145/1150402.1150476.

35 Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V. S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. Efficient algorithms for densest subgraph discovery on large directed graphs. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1051–1066. ACM, 2020. doi:10.1145/3318464.3389697.

36 Ofer Neiman and Shay Solomon. Simple Deterministic Algorithms for Fully Dynamic Maximal Matching. *ACM Trans. Algorithms*, 12(1):7:1–7:15, 2016. doi:10.1145/2700206.

37 Jean-Claude Picard and Maurice Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12(2):141–159, 1982. doi:10.1002/NET.3230120206.

38 Henrik Reinstädtler, Christian Schulz, and Bora Uçar. HeiOrient. Software, DFG-SCHU 2567/3-1, swhId: swh:1:dir:be7317d125554a54dfd1c9d17521c500c4e1ebc3 (visited on 2024-11-28). doi:10.4230/artifacts.22501.

39 Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *Annual International Conference on Research in Computational Molecular Biology*, pages 456–472. Springer, 2010. doi:10.1007/978-3-642-12683-3_30.

40 Saurabh Sawlani and Junxing Wang. Near-optimal fully dynamic densest subgraph. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 181–193, 2020. doi:10.1145/3357713.3384327.

41 Mauro Sozio and Aristides Gionis. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 939–948, 2010. doi:10.1145/1835804.1835923.

42 Bintao Sun, Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. Kclist++: A simple algorithm for finding k-clique densest subgraphs in large graphs. *Proc. VLDB Endow.*, 13(10):1628–1640, 2020. doi:10.14778/3401960.3401962.

43 Yichen Xu, Chenhao Ma, Yixiang Fang, and Zhifeng Bao. Efficient and effective algorithms for generalized densest subgraph discovery. *Proceedings of the ACM on Management of Data*, 1(2):1–27, 2023. doi:10.1145/3589314.

44 Bin Zhang and Steve Horvath. A general framework for weighted gene co-expression network analysis. *Statistical applications in genetics and molecular biology*, 4(1), 2005.

45 Yalong Zhang, Ronghua Li, Qi Zhang, Hongchao Qin, and Guoren Wang. Efficient algorithms for density decomposition on large static and dynamic graphs. *Proc. VLDB Endow.*, 17(11):2933–2945, 2024. doi:10.14778/3681954.3681974.

Bicriteria Approximation for k-Edge-Connectivity

Zeev Nutov
The Open University of Israel, Ra'anana, Israel

Reut Cohen
The Open University of Israel, Ra'anana, Israel

Abstract

In the k-Edge Connected Spanning Subgraph (k-ECSS) problem we are given a (multi-)graph $G = (V, E)$ with edge costs and an integer k, and seek a min-cost k-edge-connected spanning subgraph of G. The problem admits a 2-approximation algorithm and no better approximation ratio is known. Recently, Hershkowitz, Klein, and Zenklusen [STOC 24] gave a bicriteria $(1, k-10)$-approximation algorithm that computes a $(k-10)$-edge-connected spanning subgraph of cost at most the optimal value of a standard Cut-LP for k-ECSS. We improve the bicriteria approximation to $(1, k-4)$ and also give another non-trivial bicriteria approximation $(3/2, k-2)$.

The k-Edge-Connected Spanning Multi-subgraph (k-ECSM) problem is almost the same as k-ECSS, except that any edge can be selected multiple times at the same cost. A $(1, k-p)$ bicriteria approximation for k-ECSS w.r.t. Cut-LP implies approximation ratio $1+p/k$ for k-ECSM, hence our result also improves the approximation ratio for k-ECSM.

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

Keywords and phrases k-edge-connected subgraph, bicriteria approximation, iterative LP-rounding

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.66

1 Introduction

A graph is **k-edge-connected** if it contains k edge-disjoint paths between any two nodes. We consider the following problem.

k-Edge-Connected Spanning Subgraph (k-ECSS)
Input: A (multi-)graph $G = (V, E)$ with edge costs and an integer k.
Output: A min-cost k-edge-connected spanning subgraph J of G.

The problems admits approximation ratio 2 by a reduction to a bidirected graph (c.f. [17]), and no better approximation ratio is known. Recently, Hershkowitz, Klein, and Zenklusen [14] (henceforth HKZ) gave a bicriteria $(1, k-10)$-approximation algorithm that computes a $(k-10)$-edge-connected spanning subgraph of cost at most the optimal value of a standard Cut-LP for k-ECSS. Our first result improves the connectivity approximation to $k-4$.

▶ **Theorem 1.** *k-ECSS admits a bicriteria approximation ratio $(1, k-4)$ w.r.t. the Cut-LP.*

The k-Edge-Connected Spanning Multi-subgraph (k-ECSM) problem is almost the same as k-ECSS, except that any edge can be selected multiple times at the same cost. For k-ECSM, approximation ratios $3/2$ for k even and $3/2 + O(1/k)$ for k odd were known long time ago [10, 11], and it was conjectured by Pritchard [29] that k-ECSM admits approximation ratio $1 + O(1/k)$. HKZ [14] showed that k-ECSM has an $(1 + \Omega(1/k))$-approximation threshold and that a $(1, k-p)$ bicriteria approximation for k-ECSS w.r.t. the standard Cut-LP implies approximation ratio $1 + p/k$ for k-ECSM. Thus the novel result of HKZ [14] implies an approximation ratio $1 + 10/k$ for k-ECSM (improving the previous ratio $1 + O(1/\sqrt{k})$ of [16]), and we reduce the approximation ratio to $1 + 4/k$.

▶ **Corollary 2.** *k-ECSM admits approximation ratio $1 + 4/k$.*


© Zeev Nutov and Reut Cohen;
licensed under Creative Commons License CC-BY 4.0



In addition, we also study bicriteria approximation ratios when the cost approximation is bellow 2, and prove the following.

▶ **Theorem 3.** k-ECSS *admits a bicriteria approximation ratio* $(3/2, k-2)$.

We note that while HKZ presented a novel original approach for attacking the problem and gave the first additive bicriteria approximation, our contribution here is more modest, as follows.

- We present a simpler algorithm that uses the HKZ [14] idea and a much simpler proof, based on new properties of k-ECSS Cut-LP extreme point solutions; see Lemmas 4.
- This enabled us to improve the connectivity approximation from $k-10$ to $k-4$.
- We use the same idea to present another nontrivial bicriteria approximation $(3/2, k-2)$.

We now survey some related work. k-ECSS is an old fundamental problem in combinatorial optimization and network design. The case $k=1$ is the MST problem, while for $k=2$ the problem is MAX-SNP hard even for unit costs [7]. Pritchard [29] showed that there is a constant $\epsilon > 0$ such that for any $k \geq 2$, k-ECSS admits no $(1+\epsilon)$-approximation algorithm, unless P = NP. No such hardness of approximation was known for k-ECSM, and Pritchard [29] conjectured that k-ECSM admits approximation ratio $1 + O(1/k)$. This conjecture was resolved by HKZ [14], who also proved a matching hardness of approximation result.

The k-ECSS problem and its special cases have been extensively studied, c.f [22, 8, 10, 11, 18, 1, 4, 17, 13, 12, 29, 30, 32, 14] for only a small sample of papers in the field. Nevertheless, the current best known approximation ratio for k-ECSS is 2, the same as was four decades ago. The 2-approximation is obtained by bidirecting the edges of G, computing a min-cost directed spanning subgraph that contains k edge-disjoin dipaths to all nodes from some root node (this can be done in polynomial time [5]), and returning its underlying graph. A 2-approximation can also be achieved with the iterative rounding method [15], that gives a 2-approximation also for the more general Steiner Network problem, where we seek a min-cost subgraph that contains r_{uv} edge-disjoint paths for every pair of nodes $u, v \in V$. The directed version of k-ECSS was also studied, c.f. [5, 13, 1, 17], and for this case also no better than 2-approximation is known, except for special cases.

Bicriteria approximation algorithms were widely studied for degree constrained network design problems such as Bounded Degree MST and Bounded Degree Steiner Network, c.f. [31, 21, 20] and the references therein. A particular case of a bicriteria approximation is when only one parameter is relaxed while the cost/size of the solution is bounded by a budget B. For example, the $(1-1/e)$-approximation for the Budgeted Max-Coverage problem can be viewed as a bicriteria approximation $(1, 1-1/e)$ for Set Cover, where only a fraction of $1-1/e$ of the elements is covered by opt sets. Similarly, in the budgeted version Budgeted ECSS of k-ECSS, instead of k we are given a budget B and seek a spanning subgraph of cost at most B that has maximum edge-connectivity k^*.

One can view Theorem 1 as a $k^* - 4$ additive approximation for Budgeted ECSS, where k^* is the maximum edge-connectivity under the budget B. We note that budgeted connectivity problems were studied before for unit costs. Specifically, in the Min-Size k-ECSS and Min-Size k-CSS problems we seek a k-edge-connected and k-node-connected, respectively, spanning subgraph with a minimal number of edges. Nutov [24] showed that the following simple heuristic (previously analyzed by Cheriyan and Thurimella [1]) is a $(1, k-1)$ bicriteria approximation for Min-Size k-CSS: compute a min-size $(k-2)$-edge-cover I_{k-2} (a spanning subgraph of minimal degree $\geq k-2$) and then augment it by an inclusion minimal edge set $F \subseteq E$ such that $I_{k-2} \cup F$ is $(k-1)$-connected. This $(1, k-1)$ bicriteria

approximation implies a $k^* - 1$ approximation for Budgeted Min-Size k-CSS, which is tight, since the problem is NP-hard. A similar result for k-ECSS, for large enough k, can be deduced from the work of Gabow and Gallagher [12].

One can obtain a bicriteria approximation $(2/\theta, \lfloor k/\theta \rfloor)$ for k-ECSS for any $\theta \geq 1$, by just solving the $\lfloor k/\theta \rfloor$-ECSS problem. This is since the 2 approximation is w.r.t. the Cut-LP. Specifically, if x is a k-ECSS Cut-LP solution, then x/θ is a $\lfloor k/\theta \rfloor$-ECSS Cut-LP solution, thus a 2-approximation w.r.t. Cut-LP for $\lfloor k/\theta \rfloor$-ECSS computes a $\lfloor k/\theta \rfloor$-connected subgraph of cost $\leq 2/\theta \cdot \mathsf{opt}_k$. Similar "LP-scaling" gives a bicriteria approximation $(2/\theta, \lfloor r_{uv}/\theta \rfloor)$ for any $\theta \geq 1$ for the Steiner Network problem.

The node connectivity version k-CSS of k-ECSS was also studied extensively [19, 6, 25, 1, 24, 2, 28, 26, 27]; it admits approximation ratio $4 + \epsilon$ when k is bounded by a constant, but for general k only a polylogarithmic approximation is known [6, 25]. The Survivable Network Design Problem (SNDP) is the node-connectivity version of Steiner Network when there should be r_{uv} internally node disjoint path between every $u, v \in V$; k-CSS is a particular case when $r_{uv} = k$ for all $u, v \in V$. The Rooted SNDP is another particular case of SNDP where we require k disjoint paths from a root s to every node in a given set T of terminals, and the k-Out-Connected Spanning Subgraph (k-OCSS) problem is a particular case of Rooted SNDP when $T = V \setminus \{s\}$. Since the best approximation ratios for these problems are w.r.t. to the Cut-LP (a.k.a. "biset LP") they can be used to obtain bicretiria approximations using the method described above. We summarize the best approximation ratios for these problems and the bicriteria approximation ratios that can be derived for them by a simple "LP-scaling" in the following table.

Table 1 LP and bicriteria approximation ratios for node connectivity problems. The third row gives the approximation cost that can achieve connectivity $\lfloor k/\theta \rfloor$ ($\lfloor r_{uv}/\theta \rfloor$ for SNDP) for any $\theta \geq 1$.

problem	SNDP	Rooted SNDP	k-CSS	k-OCSS
approximation	$O(k^3 \log n)$ [3]	$O(k \log k)$ [23]	$4 + \frac{4 \lg k + 1}{\lg n - \lg k}$ [28]	2 [9]
bicriteria appr.	$\frac{1}{\theta} O((k/\theta)^3 \log n)$	$\frac{1}{\theta} O((k/\theta) \log(k/\theta))$	$\frac{1}{\theta}\left(4 + \frac{4 \lg(k/\theta) + 1}{\lg n - \lg(k/\theta)}\right)$	$2/\theta$

This paper is organized as follows. In Sect. 2 we describe our iterative rounding algorithm. In Sect. 3 we prove the connectivity guarantee, and in Sect. 4 we prove a certain property of extreme point solutions that is used in the algorithm. Sect. 5 gives a bicriteria approximation $(3/2, k-2)$.

2 The Algorithm

For an edge set F and disjoint node sets S, T let $\delta_F(S,T)$ denote the set of edges in F with one end in S and the other end in T, and let $d_F(S,T) = |\delta_F(S,T)|$. Let $\delta_F(S) = \delta_F(S, \bar{S})$ and $d_F(S) = |\delta_F(S)|$, where $\bar{S} = V \setminus S$ is the node complement of S. For $x \in \mathbb{R}^E$ let $x(F) = \sum_{e \in F} x(e)$. We will often refer to a proper subset S of V (so $\emptyset \neq S \subset V$) as a **cut**. A standard Cut-LP for k-ECSS is:

$$\begin{array}{lll} \min & c^T \cdot x & \\ \text{s.t.} & x(\delta_E(S)) \geq k & \forall \emptyset \neq S \subset V \\ & 0 \leq x_e \leq 1 & \forall e \in E \end{array}$$

As in HKZ [14], we will use the iterative rounding relaxation method. This means that while $E \neq \emptyset$, we repeatedly compute an extreme point optimal solution x to the Cut-LP above, and do at least one of the following steps:

1. Remove from E an edge e with $x_e = 0$.
2. Add to a partial solution I an edge e with $x_e = 1$, and remove e from E.
3. Relax some cut constraints $x(\delta(S)) \geq k$ to $x(\delta(S)) \geq k - q$ for some integer q.

The relaxation of the cut constraints will be implemented in two different ways.

(i) Relaxing the cut constraint of a single cut S (and also of $\bar{S} = V \setminus S$) and then contracting S into a single node v_S; this also removes the constraint of every cut A such that both $A \cap S$ and $A \cap \bar{S}$ are non-empty. We will store the nodes that correspond to such relaxed contracted cuts S in a set U.

(ii) Relaxing the constraints of all cuts S that separate two nodes $u_1, u_2 \in U$ (that correspond to contracted sets C_1, C_2) to $x(\delta(S)) \geq k - 2$. This is achieved by adding to the partial solution a "ghost edge" u_1u_2 of capacity 2, and considering the "residual" demands of sets w.r.t the ghost edges. We will store the added ghost edges in a set H.

Instead of working with the cut constraints $x(\delta_E(S)) \geq k$ it would be convenient to consider the residual constraints $x(\delta_E(S)) \geq f(S)$ for an appropriately defined set function f. To define this f, suppose that we are already given:

- A partial solution I of already chosen "integral" edges, that were removed from E.
- A set H of "ghost-edges" – these are new virtual edges that are added to the solution.
- A set U of nodes that correspond to contracted cuts whose constraints are relaxed.

Let $f(S)$ be a set function defined on proper node subsets S by

$$f(S) = \begin{cases} k - d_I(S) - 2d_H(S) - 2 & \text{if } S = \{u\} \text{ or } \bar{S} = \{u\} \text{ for some } u \in U \\ k - d_I(S) - 2d_H(S) & \text{otherwise} \end{cases} \tag{1}$$

This function $f(S)$ is a relaxation of the usual residual function $k - d_I(S)$ of k-ECSS. Consider the following LP-relaxation with the function f above:

$$\begin{array}{lll} \min & c^T \cdot x & \\ \text{s.t.} & x(\delta_E(S)) \geq f(S) & \forall \emptyset \neq S \subset V \\ & 0 \leq x_e \leq 1 & \forall e \in E \end{array} \tag{2}$$

A set S is **f-positive** if $f(S) > 0$. Given an LP solution x we say that S is **x-tight** if the LP-inequality of S holds with equality, namely, if $x(\delta(S)) = f(S)$. An **x-core**, or simply a **core** if x is clear from the context, is an inclusion-minimal f-positive x-tight set (note that an x-core must be f-positive and not only x-tight).

To **contract a node subset** C of a graph $G = (V, E)$ means to identify all nodes in C into one node v_C; edges that have an end in C now have v_C as their end and the arising loops (if any) are deleted. During the algorithm, we iteratively contract certain node subsets of G, so we denote the initial graph by $G_0 = (V_0, E_0)$.

Our algorithm is given in Algorithm 1. The main difference between Algorithm 1 and that of HKZ is that we never add parallel ghost edges, namely we must have $d_H(u, v) = 0$ whenever we add a ghost edge uv. This has a chain reaction of simplifying some other parts of the algorithm. For example, we add a ghost edge when $d_I(u, v) \geq \lceil (k-3)/2 \rceil$ (and $d_H(u, v) = 0$) while the analogous condition in HKZ is $d_{I \cup H}(uv) \in [k/2 - 2, k/2)$. The upper bound $d_{I \cup H}(u, v) < k/2$ in HKZ is needed to control the number of parallel ghost edges added, which also complicates the analysis.

In Section 4 we will prove the following lemma about extreme point solutions of LP (2).

Algorithm 1 A bicriteria $(1, k-4)$-approximation.

```
1  initialization: I ← ∅, H ← ∅, U ← ∅, μ ← ⌈(k − 3)/2⌉
2  while E ≠ ∅ do
3      compute an extreme point solution x to LP (2) with f in (1)
4      remove from E every edge e with x_e = 0
5      if there is e ∈ E with x_e = 1 then
6          move from E to I every edge e with x_e = 1 and goto line 2
7      if there is an x-core C with d_E(C) ∈ {2, 3} then
8          contract C into a single node v_C and update U ← (U \ C) ∪ {v_C}
9      else if there are u, v ∈ U with d_I(u, v) ≥ μ and d_H(u, v) = 0 then
10         add to H a ghost-edge uv and update U ← U \ {u, v}
11 return J = I
```

▶ **Lemma 4.** *Let f be defined by (1) and suppose that $d_H(u) \leq 1$ and $d_I(u) + 2d_H(u) \geq k-2$ for all $u \in U$. Let x be an extreme point of the polytope $P = \{x \in [0,1]^E : x(\delta_E(S)) \geq f(S)\}$ such that $0 < x_e < 1$ for all $e \in E$, where $E \neq \emptyset$. Then at least one of the following holds.*

(i) *There is an x-core C with $d_E(C) \in \{2,3\}$; moreover, no edge in E has both ends in C.*

(ii) *There are $u, v \in U$ with $d_H(u,v) = 0$ and $d_I(u,v) \geq \mu$, where $\mu = \lceil (k-3)/2 \rceil$.*

Furthermore, such u, v or such C can be found in polynomial time.

Interestingly, the two cases in Lemma 4 depend on whether there exist a laminar set family $\mathcal{L}$ such that x is the unique solution to the equation system $\{x(\delta_E(S) = f(S) : S \in \mathcal{L}\}$. Specifically, if such $\mathcal{L}$ exists then case (i) in Lemma 4 holds, otherwise case (ii) must hold.

In the next Section 3 we will show that the algorithm terminates after a polynomial number of iterations and prove the connectivity guarantee $k-4$. Since in each iteration the cut constraints are only relaxed (by contractions, or by including nodes in U, or by adding ghost edges), the cost of the produced solution is at most the initial LP-value. In this context note that the -2 relaxation of the nodes $u, v \in U$ is not canceled when u, v are removed from U, it is just replaced by the -2 relaxation caused by the added ghost edge uv.

3 Connectivity guarantee

For every ghost edge $h = uv$ added, let $I(h) = I(uv)$ be the set of (at least $\mu = \lceil (k-3)/2 \rceil$) edges in J that appear in $\delta_I(u, v)$. A key observation is the following.

▶ **Lemma 5.** *For any distinct ghost edges h, h' added during the algorithm, $I(h) \cap I(h') = \emptyset$.*

Proof. This follows from the condition that when a ghost edge $h = uv$ added, there is no other ghost edge between u and v. Specifically, assume w.l.o.g. that $h = uv$ was added after h'. Then all edges in $I(h)$ are uv-edges – go between the sets C_u, C_v that correspond to u, v, while no edge in $I(h')$ is a uv-edge. ◀

▶ **Lemma 6.** *When a core C is found in the algorithm but not yet contracted, $d_H(C) \leq 1$, $x(\delta_E(C)) \in \{1,2\}$, and $d_I(C) = k - 2d_H(C) - x(\delta_E(C))$. Consequently,*

- *$d_I(C) \in \{k-1, k-2\}$ if $d_H(C) = 0$.*
- *$d_I(C) \in \{k-3, k-4\}$ if $d_H(C) = 1$.*

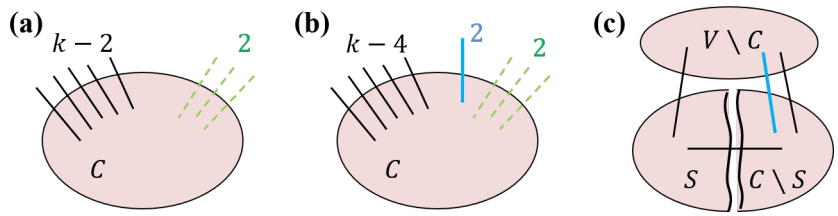


Figure 1 Illustration to the proofs of Lemmas 6 and 8. Edges in I are shown by black lines, edges in E (the fractional edges) by green dashed lines, and the ghost edge is blue.

Proof. If $d_H(C) \geq 2$ then $d_I(C) + 2d_H(C) \geq 2\mu + 4 \geq k$, hence $f(C) \leq 0$, contradicting that C is a core (recall that a core must be an f-positive set). We have $d_E(C) \in \{2, 3\}$, so there are 2 or 3 fractional edges in $\delta_E(C)$. Since C is x-tight, $x(\delta_E(C))$ is a positive integer that is either 1 or 2, so there are $k - 1 - 2d_H(C)$ or $k - 2 - 2d_H(C)$ (integral) edges in $\delta_I(C)$. If $d_H(C) = 0$ then $d_I(C) \in \{k - 1, k - 2\}$ and if $d_H(C) = 1$ then $d_I(C) \in \{k - 3, k - 4\}$; see Fig. 1(a,b) where $\delta_E(x(C)) = 2$ and $d_H(C) = 0$ or $d_H(C) = 1$, respectively. ◀

The following lemma shows that the conditions of Lemma 4 hold during the algorithm, and thus the algorithm terminates.

▶ **Lemma 7.** *During the algorithm $d_H(u) \leq 1$ and $d_I(u) + 2d_H(u) \geq k - 2$ holds for all $u \in U$. Furthermore, $d_H(v) \leq 2$ for all $v \in V \setminus U$.*

Proof. By Lemma 6, when u enters U we have $d_H(u) \leq 1$ and $d_I(u) \geq k - 2d_H(u) - 2$. This implies $f(\{u\}) = k - d_I(u) - 2d_H(u) - 2 \leq 0$. The addition of a ghost edge incident to u excludes u from U but does not increase $f(\{u\})$. Hence $\{u\}$ will never be chosen as a core C and will never enter U again. If we had $d_H(u) = 1$ when u entered U, then u will leave U with $d_H(u) = 2$ and will never enter U again. Thus $d_H(v) \leq 2$ for all $v \in V \setminus U$. ◀

A set family $\mathcal{R}$ is **laminar** if any two sets in the family are either disjoint or one contains the other. Let $\mathcal{R}$ be the family of subsets of V_0 that correspond to the contracted sets during the algorithm (recall that $G_0 = (V_0, E_0)$ denotes the initial graph). It is known and not hard to see that $\mathcal{R}$ is laminar. We will say that a cut S of G_0 is **compatible** with a graph G during the algorithm if $R \subseteq S$ or $R \cap S = \emptyset$ for every $R \in \mathcal{R}$ that was contracted so far to obtain G. As long as S is compatible with G, we can look at the cut of G that corresponds to S, which for simplicity of notation we will also denote by S, and lower bound $d_I(S)$ (the degree of the set that corresponds to S w.r.t to the partial solution I) instead of $d_J(S)$ (the degree of S w.r.t to the final solution J), since $I \subseteq J$ and hence $d_J(S) \geq d_I(S)$.

We would like to establish the bound $d_J(S) \geq k - 4$ for a cut S of G_0 such that $\mathcal{R} \cup \{S\}$ is laminar. For such S, it is easy to determine the last graph G that S is compatible with. For a cut S of G_0 let R_S be the inclusion minimal set in $\mathcal{R} \cup \{V\}$ that properly contains S. If $R_S = V$ then S compatible with G till the end of the algorithm. If $R_S \neq V$ then S (such that $\mathcal{R} \cup \{S\}$ is laminar) is compatible with G as long as R_S is not contracted, hence we will analyze such S as a cut of G right before the core $C = R_S$ is contracted. Due to the fact that at this moment no edge in E has both ends in C and since $d_E(C) \leq 3$ and $x(\delta_E(C)) \leq 2$ (by Lemma 4(i)), the difference between $d_I(S)$ and $d_J(S)$ is small.

▶ **Lemma 8.** *When a core C in the algorithm is found but not yet contracted, the following holds for any proper subset S of C.*

(i) *$d_I(S) \geq d_H(S) \cdot \mu$ and $d_I(S) \geq k - 2d_H(S) - 2$.*

(ii) *$d_I(S, C \setminus S) \geq \mu$.*

Proof. We prove (i). $d_I(S) \geq d_H(S) \cdot \mu$ by Lemma 5. If $S = \{u\}$ for some $u \in U$ then $d_I(u) \geq k - 2d_H(u) - 2$ by Lemma 6. Otherwise, the constraint of S is relaxed by ghost edges only, and since $\delta_E(S) \subseteq \delta_E(C)$ (Lemma 4(i)) and $x(\delta_E(C)) \leq 2$ (Lemma 6), we get

$$d_I(S) \geq k - 2d_H(S) - x(\delta_E(S)) \geq k - 2d_H(S) - 2 \ .$$

We prove (ii). If $d_H(S, C \setminus S) \geq 1$ then $d_I(S, C \setminus S) \geq \mu$ since $d_I(u, v) \geq \mu$ when the ghost edge uv is added. Otherwise, $d_H(S) + d_H(C \setminus S) = d_H(C) \leq 1$ and since $d_J(R) \leq k - 2d_H(C) - 1$ (by Lemma 6) and by part (i) we get

$$\begin{aligned} 2d_I(S, C \setminus S) &= d_I(S) + d_I(C \setminus S) - d_I(C) \\ &\geq [k - 2d_H(S) - 2] + [k - 2d_H(C \setminus S) - 2] - [k - 2d_H(C) - 1] \\ &= k - 3 - 2[d_H(S) + d_H(C \setminus S) - d_H(C)] = k - 3 \ , \end{aligned}$$

concluding the proof. ◀

The notation $\delta_H(S)$ and $d_H(S)$ are well defined for any node subset S at any iteration of the algorithm. We now extend it to any subset S of V. We can view a ghost edge v_1v_2 as a pair $\{C_1, C_2\}$ of sets in $\mathcal{R}$ that correspond to v_1, v_2. We say that such a **ghost-edge covers** S if one of C_1, C_2 is contained in S and the other in $\bar{S}$. Let $\delta_H(S)$ denote the set of ghost edges in H that cover S and let $d_H(S)$ be their number.

▶ **Corollary 9.** *Let S be a cut such that $\mathcal{R} \cup \{S\}$ is laminar. Then $d_J(S) \geq k - 2d_H(S) - 2$ and $d_J(S) \geq d_H(S) \cdot \mu$. Furthermore, if $R_S = V$ and $S \notin R$ then $d_J(S) \geq k - 2d_H(S)$. Consequently, $d_J(S) \geq \max\{k - 2d_H(S) - 2, d_H(S) \cdot \mu\} \geq k - 4$.*

Proof. The bound $d_J(S) \geq d_H(S) \cdot \mu$ follows from Lemma 5, so we prove only the bound $d_J(S) \geq k - 2d_H(S) - 2$. For $S \in \mathcal{R}$ it follows from Lemma 6, so assume that $S \notin \mathcal{R}$. If S is contained in some set in $\mathcal{R}$, then let C be the inclusion minimal set in $\mathcal{R}$ that contains S. The bound then follows from Lemma 8(i). Otherwise, the constraint of S was relaxed by ghost edges only and then $d_J(S) \geq k - 2d_H(S)$. ◀

Before establishing the bound $d_J(S) \geq k - 4$ for sets S such that $\mathcal{R} \cup \{S\}$ is not laminar, we need three additional lemmas. The next lemma considers sets $R \in \mathcal{R}$ and $T \subset \bar{R}$ such that both $\mathcal{R} \cup T, \mathcal{R} \cup (\bar{R} \setminus T)$ are laminar; one can verify that this is equivalent to $\mathcal{R} \cup \{T, \bar{R} \setminus T\}$ being laminar. Moreover, this means that R is an inclusion maximal set in $\mathcal{R}$, and any other set in $\mathcal{R}$, except for maybe $\bar{R}$ (if $\bar{R} \in \mathcal{R}$), is contained in one of $R, T, \bar{R} \setminus T$.

▶ **Lemma 10.** *Let $R \in \mathcal{R}$ and let T be a proper subset of $\bar{R}$ such that the set family $\mathcal{R} \cup \{T, \bar{R} \setminus T\}$ is laminar. Then $d_J(T, \bar{R} \setminus T) \geq \lceil (k-6)/2 \rceil$.*

Proof. If $d_H(T, \bar{R} \setminus T) \geq 1$ then $d_J(T, R \setminus T) \geq \mu > \lceil (k-6)/2 \rceil$ by Lemma 5. Otherwise, $d_H(R) = d_H(T) + d_H(\bar{R} \setminus T)$. Since $d_J(T) \geq k - 2d_H(T) - 2$ and $d_J(\bar{R} \setminus T) \geq k - 2d_H(\bar{R} \setminus T) - 2$ by Corollary 9, and $d_J(R) \leq k - 2d_H(R) + 2$ by Lemma 6, we get (see Fig. 2(a))

$$\begin{aligned} 2d_J(T, \bar{R} \setminus T) &= d_J(T) + d_J(\bar{R} \setminus T) - d_J(R) \\ &\geq [k - 2d_H(T) - 2] + [k - 2d_H(\bar{R} \setminus T) - 2] - [k - 2d_H(R) + 2] \\ &= k - 6 - 2[d_H(T) + d_H(\bar{R} \setminus T) - d_H(R)] = k - 6 \ , \end{aligned}$$

concluding the proof. ◀

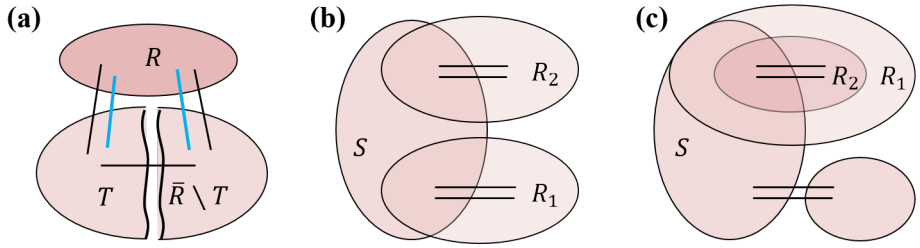


Figure 2 Illustration to the proofs of Lemmas 10 and 13.

We say that a set S **overlaps** a set R, or that R, S **overlap**, if the pair $\{R, S\}$ is not laminar, namely if the sets $R \cap S, R \setminus S, S \setminus R$ are all non-empty. Now we consider sets S that overlap some $R \in \mathcal{R}$. Note however, that it may be that S overlaps some $R \in \mathcal{R}$, but $\bar{S}$ does not, so we can still deduce from Corollary 9 that $d_J(S) = d_J(\bar{S}) \geq k - 4$. To avoid this confusion we will examine among the sets $S, \bar{S}$ one that is more "compatible" with $\mathcal{R}$. More formally, let $\mathcal{R}(S) = \{R \in \mathcal{R} : S \text{ overlaps } R\}$ be the family of sets in $\mathcal{R}$ that S overlaps, and note that so far we examined the case $|\mathcal{R}(S)| = 0$. So now w.l.o.g. we will assume that $1 \leq |\mathcal{R}(S)| \leq |\mathcal{R}(\bar{S})|$. Under this assumption, we have the following.

▶ **Lemma 11.** *If $1 \leq |\mathcal{R}(S)| \leq |\mathcal{R}(\bar{S})|$ then $V \setminus (R \cup S) \neq \emptyset$ for any $R \in \mathcal{R}(S)$.*

Proof. Let $R \in \mathcal{R}(S)$ and suppose that $|\mathcal{R}(S)| \leq |\mathcal{R}(\bar{S})|$. It is known that $\mathcal{R}(R \setminus S) \subset \mathcal{R}(S)$; this is so since every set in $\mathcal{R}$ that overlaps $R \setminus S$ also overlaps S, but R overlaps S but not $R \setminus S$, c.f. [33, Lemma 23.15]. Thus $|\mathcal{R}(R \setminus S)| < |\mathcal{R}(S)|$. If $V \setminus (R \cup S) = \emptyset$ then $R \setminus S = \bar{S}$ and we get $|\mathcal{R}(\bar{S})| < |\mathcal{R}(S)|$ contradicting the assumption $|\mathcal{R}(S)| \leq |\mathcal{R}(\bar{S})|$. ◀

▶ **Lemma 12.** *Suppose that $1 \leq |\mathcal{R}(S)| \leq |\mathcal{R}(\bar{S})|$.*
(i) *If R is a minimal set in $\mathcal{R}(S)$ then $d_J(R \cap S, R \setminus S) \geq \mu$.*
(ii) *If R is a unique maximal set in $\mathcal{R}(S)$ then $d_J(\bar{R} \cap S, \bar{R} \setminus S) \geq \lceil (k-6)/2 \rceil$.*

Proof. We prove (i). The minimality of R implies that if $T \in \{R \cap S, R \setminus S\}$ then $\mathcal{R} \cup \{T\}$ is laminar. Therefore by Lemma 8(ii) $d_J(R \cap S, R \setminus S) \geq \mu$.

We prove (ii). The maximality of R implies that if $T \in \{\bar{R} \cap S, \bar{R} \setminus S\}$ then $\mathcal{R} \cup \{T\}$ is laminar. Therefore by Lemma 10 $d_J(\bar{R} \cap S, \bar{R} \setminus S) \geq \lceil (k-6)/2 \rceil$. ◀

▶ **Lemma 13.** *If $1 \leq |\mathcal{R}(S)| \leq |\mathcal{R}(\bar{S})|$ then $d_J(S) \geq k - 4$.*

Proof. Suppose that there are two disjoint sets in $\mathcal{R}(S)$. Then there are two minimal sets in $\mathcal{R}(S)$ that are disjoint, say R_1, R_2; see Fig. 2(b). Then by Lemma 12(i)

$$d_J(S) \geq d_J(R_1 \cap S, R_1 \setminus S) + d_J(R_2 \cap S, R_2 \setminus S) \geq \mu + \mu > k - 4 \ .$$

Otherwise, $\mathcal{R}(S)$ is a nested family with a unique maximal set R_1 and a unique minimal set R_2; see Fig. 2(c) and note that $R_2 \subseteq R_1$ and possibly $R_1 = R_2$. Then by Lemma 12 we get

$$d_J(S) \geq d_J(\bar{R}_1 \cap S, \bar{R}_1 \setminus S) + d_J(R_2 \cap S, R_2 \setminus S) \geq \lceil (k-6)/2 \rceil + \mu = k - 4 \ ,$$

concluding the proof. ◀

This concludes the proof of the connectivity guarantee of Algorithm 1. Now we show that the algorithm terminates after a polynomial number of iterations.

▶ **Lemma 14.** *The algorithm terminates after $O(n)$ iterations, where $n = |V|$.*

Proof. Let x be an extreme point computed at the first iteration of the algorithm. It is known (c.f. [13]) that $|\{0 < x_e < 1 : e \in E\}| \leq 2n-1$, namely, at most $2n-1$ variables x_e are fractional. This follows from two facts. The first is that after the integral entries are substituted, the fractional entries are determined by a full rank system of constraints $\{x(\delta_E(S)) = k : S \in \mathcal{L}\}$ where $\mathcal{L}$ is laminar, c.f. [15, 13]. The second is that a laminar family on a set of n elements has size at most $2n-1$. The latter fact also implies that $|\mathcal{R}| \leq 2n-1$. By Lemma 7, the ghost edges form a 2-matching on $\mathcal{R}$, thus $|H| \leq |\mathcal{R}| \leq 2n-1$. At each iteration we remove an edge from E, or add a set to $\mathcal{R}$, or add a ghost edge. Hence the number of iterations is bounded by $|\mathcal{L}|+|\mathcal{R}|+|H| \leq (2n-1)+(2n-1)+(2n-1) = 3(2n-1) = O(n)$. ◀

4 Properties of extreme point solutions (Lemma 4)

Here we will prove Lemma 4. Fix an extreme point solution x as in the Lemma. Then there exists a family $\mathcal{L}$ of f-positive x-tight sets such that x is the unique solution to the equation system $\{x(\delta_E(S)) = f(S) : S \in \mathcal{L}\}$; namely, $|\mathcal{L}| = |E|$ and the characteristic vectors of the sets $\{\delta_E(S) : S \in \mathcal{L}\}$ are linearly independent. We will call such a family $\mathcal{L}$ **x-defining**. The following was essentially proved in [14].

▶ **Lemma 15** ([14]). *If there exists a laminar x-defining family $\mathcal{L}$ then there is an x-core C such that $d_E(C) \in \{2,3\}$ and $f(C) \in \{1,2\}$; moreover, no edge in E has both ends in C.*

Proof. Let $\mathcal{C}_{\mathcal{L}}$ be the family of minimal sets in $\mathcal{L}$. It was proved by HKZ [14] that there is $S \in \mathcal{C}_{\mathcal{L}}$ such that $d_E(S) \leq 3$ and no edge in E has both ends in S; we provide a proof sketch for completeness of exposition. Suppose to the contrary that $d_E(S) \geq 4$ for every $S \in \mathcal{C}_{\mathcal{L}}$. Assign 2 tokens to every edge uv, and then reassign these tokens – one to the smallest set in $\mathcal{L}$ that contains u and one to the smallest set in $\mathcal{L}$ that contains v (if such sets exist). The paper of Jain [15] shows that then for any $S \in \mathcal{L}$, the tokens assigned to sets in $\mathcal{L}(S) = \{T \in \mathcal{L} : T \subseteq S\}$ can be redistributed such that S gets at least 4 tokens and every $T \in \mathcal{L}(S)$ gets at least 2 tokens, giving the contradiction $|E| > |\mathcal{L}|$. The proof in [15] is by induction on $|\mathcal{L}(S)|$. If $S \in \mathcal{C}_{\mathcal{L}}$ is a leaf then S already has 4 tokens, since $d_E(S) \geq 4$ for every $S \in \mathcal{C}_{\mathcal{L}}$. Else, S can collect 2 tokens from each child in $\mathcal{L}(S)$, so if S has at least two children then we are done. Else, S has exactly one child, and then [15] shows that by linear independence, there are 2 tokens assigned to S, which together with the extra 2 tokens of the child of S gives the required 4 tokens. This shows that there is $S \in \mathcal{C}_{\mathcal{L}}$ such that $d_E(S) \leq 3$.

Note that for any $T \in \mathcal{C}_{\mathcal{L}}$, no edge in E has both ends in T; this is since for every $e \in E$ there is $S_e \in \mathcal{L}$ such that $e \in \delta_E(S_e)$, since the variable x_e must appear in at least one equation among $\{x(\delta_E(S)) = f(S) : S \in \mathcal{L}\}$. Let $S \in \mathcal{C}_{\mathcal{L}}$ be such that $d_E(S) \leq 3$. If S is an x-core then we are done. Otherwise, let C be an x-core properly contained in S. We claim that $\delta_E(C) \subseteq \delta_E(S)$, and that $d_E(C) \geq 2$. This follows from the following observations.

1. $d_E(C) \geq 2$, since C is f-positive and since we assume that $0 < x_e < 1$ for all $e \in E$.
2. For every $e \in \delta_E(C)$ there is $S_e \in \mathcal{L}$ such that $e \in \delta_E(S_e)$, since the variable x_e must appear in at least one equation among $\{x(\delta(S)) = f(S) : S \in \mathcal{L}\}$.

This gives $d_E(C) \in \{2,3\}$, as required. ◀

In the rest of this section we will prove the following.

▶ **Lemma 16.** *Under the assumptions of Lemma 4, if no laminar x-defining family exists then there are $u, v \in U$ with $d_I(u,v) \geq \mu$ and $d_H(u,v) = 0$.*

In the rest of this section assume that the assumptions of Lemma 4 hold ($d_H(u) \leq 1$ and $d_I(u) + 2d_H(u) \geq k-2$ for all $u \in U$), and that no laminar x-defining family exists.

▶ **Lemma 17.** *There are f-positive x-tight sets A, B that cross (namely, all the four sets $A \cap B, A \setminus B, B \setminus A, \bar{A} \cap \bar{B}$ are non-empty) such that the following holds:*
(i) *$A \cap B = \{u\}$ or $\bar{A} \cap \bar{B} = \{u\}$ for some $u \in U$.*
(ii) *$A \setminus B = \{v\}$ or $B \setminus A = \{v\}$ for some $v \in U$.*

Proof. For a cut S let $\chi_S \in \{0,1\}^E$ be the incidence vector of $\delta_E(S)$. Let $\mathcal{T}$ be the family of f-positive x-tight sets (if S is tight but not f-positive, then $\chi_S \equiv 0$, since $x_e > 0$ for all $e \in E$). Let us say that $A, B \in \mathcal{T}$ are **x-uncrossable** if at least one of the following holds:
(a) $A \cap B, A \cup B$ are both x-tight and $\chi_A + \chi_B = \chi_{A \cap B} + \chi_{A \cup B}$.
(b) $A \setminus B, B \setminus A$ are both x-tight and $\chi_A + \chi_B = \chi_{A \setminus B} + \chi_{B \setminus A}$.
We will prove two claims that will imply the lemma.

▷ Claim. There are $A, B \in \mathcal{T}$ that are not x-uncrossable.

Proof. Let $\mathcal{L}$ be a maximal laminar subfamily of $\mathcal{T}$. Let $\mathsf{span}(\mathcal{T})$ denote the linear space spanned by the incidence vectors of the sets in $\mathcal{T}$ and similarly $\mathsf{span}(\mathcal{L})$ is defined. Since no laminar x-defining family exists and by the maximality of $\mathcal{L}$, there is $A \in \mathcal{T} \setminus \mathcal{L}$ such that $\chi_A \notin \mathsf{span}(\mathcal{L})$; any such A overlaps some set in $\mathcal{L}$. Choose such A that overlaps the minimal number of sets in $\mathcal{L}$, and let $B \in \mathcal{L}$ be a set that A overlaps. We show that (a) cannot hold; the proof that (b) cannot hold is similar. Suppose to the contrary that (a) holds, namely that $\chi_A + \chi_B = \chi_{A \cap B} + \chi_{A \cup B}$ and $A \cap B, A \cup B$ are both x-tight. It is known that each of the sets $A \cap B, A \cup B$ overlaps strictly less sets in $\mathcal{L}$ than A, c.f. [33, Lemma 23.15]. Thus by our choice of A, each of these sets has its incidence vector in $\mathsf{span}(\mathcal{L})$, namely $\chi_{A \cap B}, \chi_{A \cup B} \in \mathsf{span}(\mathcal{L})$. But $\chi_A = \chi_{A \cap B} + \chi_{A \cup B} - \chi_B$, contradicting that $\chi_A \notin \mathsf{span}(\mathcal{L})$. ◁

▷ Claim. If $A, B \in \mathcal{T}$ are not x-uncrossable then A, B cross and both (i) and (ii) hold.

Proof. We show that A, B cross. If A, B do not overlap then $\{A, B\} = \{A \cap B, A \cup B\}$ or $\{A, B\} = \{A \setminus B, B \setminus A\}$ and we are done. We claim that also $\bar{A} \cap \bar{B} \neq \emptyset$. Otherwise, $A \setminus B = \bar{B}$, $B \setminus A = \bar{A}$, and thus $\chi_{A \setminus B} = \chi_{\bar{B}} = \chi_B$ and $\chi_{B \setminus A} = \chi_{\bar{A}} = \chi_A$, implying that $A \setminus B, B \setminus A$ are both x-tight and $\chi_A + \chi_B = \chi_{A \setminus B} + \chi_{B \setminus A}$. Now we prove that (i) holds; a proof that (ii) holds is similar, and in fact can be deduced from (i) by the symmetry of f. Suppose to the contrary that (i) does not hold. Denoting $\Delta = x(\delta_E(A \setminus B, B \setminus A))$ we have:

$$\begin{aligned} f(A) + f(B) &= x(\delta_E(A)) + x(\delta_E(B)) \\ &= x(\delta_E(A \cap B)) + x(\delta_E(A \cup B)) + 2\Delta \\ &\geq f(A \cap B) + f(A \cup B) + 2\Delta \\ &\geq f(A) + f(B) + 2\Delta \ . \end{aligned}$$

The first equality is since A, B are tight, and the second can be verified by counting the contribution of each edge to both sides. The first inequality is since x is a feasible LP-solution. The second inequality is since none of $A, B, A \cap B, A \cup B$ is a singleton from U or its complement, hence f coincides on these sets with the symmetric supermodular function $g(S) = k - d_I(S) - 2d_H(S)$, that satisfies the inequality $g(A \cap B) + g(A \cup B) \geq g(A) + g(B)$. Consequently, equality holds everywhere, hence $A \cap B, A \cup B$ are both tight. Moreover, $\Delta = 0$, and this implies $\chi_A + \chi_B = \chi_{A \cap B} + \chi_{A \cup B}$, contradicting that A, B are not x-uncrossable. ◁

By the first claim there exists $A, B \in \mathcal{T}$ that are not x-uncrossable, while by the second claim such A, B cross and satisfy both (i) and (ii), concluding the proof of the lemma. ◀

By the symmetry of f, assume w.l.o.g. that $A \cap B = \{u\}$ and $A \setminus B = \{v\}$ for $u, v \in U$.

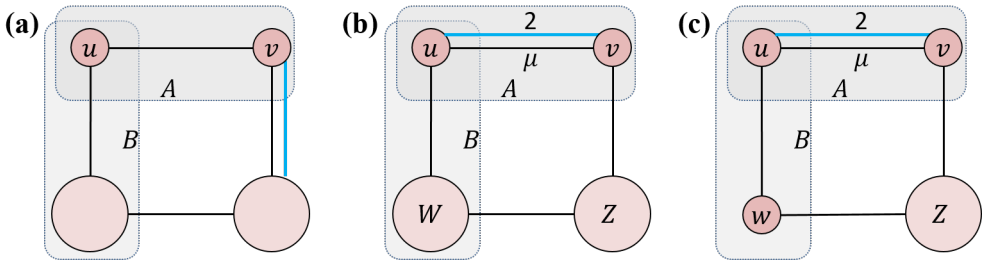


Figure 3 Illustration to the proofs of Lemmas 18 and 19.

▶ **Lemma 18.** *If $d_H(u,v) = 0$ then $d_I(u,v) \geq \mu$.*

Proof. By the assumption of Lemma 4, $d_I(u) + 2d_H(u) \geq k-2$ and $d_I(v) + 2d_H(v) \geq k-2$. Since A is f-positive, $d_I(A) + 2d_H(A) \leq k-1$. Thus we get (see Fig. 3(a))

$$k-1 \geq d_I(A)+2d_H(A) = [d_I(u)+2d_H(u)]+[d_I(v)+2d_H(v)]-2d_I(u,v) \geq 2(k-2)-2d_I(u,v)\ .$$

Consequently, $2d_I(u,v) \geq k-3$, as claimed. ◀

The next lemma shows that if $d_H(u,v) = 1$ then there is an "alternative" node $w \in U$ such that we can add a ghost edge uw or vw.

▶ **Lemma 19.** *If $d_H(u,v) = 1$ then there is $w \in U$ such that one of the following holds:*
(i) *$\bar{A} \cap B = \{w\}$, $d_H(u,w) = 0$, and $d_I(u,w) \geq \mu$.*
(ii) *$\bar{A} \setminus B = \{w\}$, $d_H(v,w) = 0$, and $d_I(v,w) \geq \mu$.*

Proof. For disjoint sets S, T let $\psi(S,T) = d_I(S,T) + 2d_H(S,T) + x(\delta_E(S,T))$, and let $\psi(S) = \psi(S,\bar{S})$ be the "coverage" of S by I, H, and the fractional edges. Note that

$$\psi(S) \geq \begin{cases} k-2 & \text{if } S = \{u\} \text{ or } \bar{S} = \{u\} \text{ for some } u \in U \\ k & \text{otherwise} \end{cases}$$

In particular, $\psi(S) < k$ implies that $S = \{w\}$ or $\bar{S} = \{w\}$ for some $w \in U$.

Let $W = \bar{A} \cap B$ and $Z = \bar{A} \setminus B$, see Fig. 3(b). Note that $\psi(A) = \psi(B) = k$ since A, B are x-tight and that $\psi(u,v) \geq 2d_H(u,v) + \mu = 2 + \mu$. This implies (see Fig. 3(b))

$$\psi(W,Z) \leq \psi(B) - \psi(u,v) \leq k - (2+\mu)\ .$$

Thus we get

$$2k - 2(2+\mu) \geq 2\psi(W,Z) = \psi(W) + \psi(Z) - \psi(A) = \psi(W) + \psi(Z) - k$$

Consequently, $\psi(W) + \psi(Z) \leq 3k - 2(2+\mu) < 2k$, since $2(2+\mu) > k$. Thus $\psi(W) < k$ or $\psi(Z) < k$, and w.l.o.g. assume that $\psi(W) < k$. This implies that $W = \{w\}$ for some $w \in U$; (see Fig. 3(c)). Note that $d_H(u,w) = 0$, since $d_H(u,v) = 1$ and since $d_H(u) \leq 1$ by the assumption of Lemma 4. Also, $d_I(B) + 2d_H(B) \leq k-1$, since B is f-positive. Thus we have

$$\begin{aligned} 2d_I(u,w) &= 2[d_I(u,w) + 2d_H(u,w)] \\ &= [d_I(u) + 2d_H(u)] + [d_I(w) + 2d_H(w)] - [d_I(B) + 2d_H(B)] \\ &\geq (k-2) + (k-2) - (k-1) = k-3\ , \end{aligned}$$

concluding the proof. ◀

To show a polynomial time implementation it is sufficient to prove the following lemma, that was essentially proved in [14]; we provide a proof for completeness of exposition.

▶ **Lemma 20** ([14]). *At any step of the algorithm, an extreme point solution x to LP (2) with f in (1), as well as the x-cores can be computed in polynomial time.*

Proof. An extreme point solution can be found using the Ellipsoid Algorithm. For that, we need the show an existence of a polynomial time separation oracle: given x, determine whether x is a feasible LP solution or find a violated inequality. Checking the inequalities $0 \leq x_e \leq 1$ is trivial. For cut inequalities, assign capacities w_e to the edges in $E \cup I \cup H$ as follows: $w_e = x_e$ if $e \in E$, $w_e = 1$ if $e \in I$, and $w_e = 2$ if $e \in H$. The inequality of a cut S is violated if and only if $w(\delta(S)) < k$ and none of $S, \bar{S}$ is a singleton in U. Thus to find an st-cut with violated inequality do the following. If $s, t \notin U$ then we find a minimum st-cut S; if $w(\delta(S)) < k$ then we found a violated inequality, else no st-cut with violated inequality exists. If $s \in U$ and $t \notin U$, then for every $u \in V \setminus \{s, t\}$ we contract u into s and find a minimum st-cut S_v; if $w(\delta(S_v)) < k$ for some v, then we found a violated inequality, else no st-cut with violated inequality exists. Similarly, If $s, t \in U$ then for every pair $u, v \in V \setminus \{s, t\}$ we contract u into s and v into t and find a minimum st-cut $S_{u,v}$; if $w(\delta(S_{u,v})) < k$ for some u, v then we found a violated inequality, else no st-cut with violated inequality exists.

We show how to find all x-cores. Assume w.l.o.g. that $0 < x_e < 1$ for all $e \in E$. One can see that a cut C is an x-core if and only if C is an inclusion minimal set that has w-capacity k and $\delta_E(C) \neq \emptyset$. For every ordered pair (s, t) with $st \in E$ we check if there exists an st-cut C (so $s \in C$ and $t \notin C$) of capacity exactly k, and if so, then we find the inclusion minimal such cut $C(s, t)$. The x-cores are the inclusion minimal sets among the sets $C(s, t)$. ◀

This concludes the proof of Lemma 4 and thus also of Theorem 1.

5 Proof of Theorem 3

In the algorithm of Theorem 3 we use the following function f:

$$f(S) = \begin{cases} k - d_I(S) - d_H(S) - 1 & \text{if } S = \{u\} \text{ or } \bar{S} = \{u\} \text{ for some } u \in U \\ k - d_I(S) - d_H(S) & \text{otherwise} \end{cases} \tag{3}$$

The main difference between Algorithm 1 and the following Algorithm 2 are:

1. We use a different function f – the one defined in (3).
2. We round to 1 edges e with $x_e \geq 2/3$.
3. Adding a ghost edge uv requires $\mu = \lceil (k-1)/2 \rceil$ integral uv-edges.

Algorithm 2 A bicriteria $(3/2, k-2)$-approximation.

```
1  initialization: I ← ∅, H ← ∅, U ← ∅, μ ← ⌈(k − 1)/2⌉
2  while E ≠ ∅ do
3      compute an extreme point solution x to LP (2) with f in (3)
4      remove from E every edge e with x_e = 0
5      if there is e ∈ E with x_e ≥ 2/3 then
6          move from E to I every edge e with x_e ≥ 2/3 and goto line 2
7      else if there is an x-core C with d_I(C) = k − 1 and d_E(C) ∈ {2, 3} then
8          contract C into a single node v_C and update U ← (U \ C) ∪ {v_C}
9      else if there are u, v ∈ U with d_I(u, v) ≥ μ and d_H(u, v) = 0 then
10         add to H a ghost-edge uv and update U ← U \ {u, v}
11 return J = I
```

A counterpart of Lemma 4 is as follows.

▶ **Lemma 21.** *Let f be defined by (3) and suppose that $d_H(u) \leq 1$ and $d_I(u)+d_H(u) \geq k-1$ for all $u \in U$. Let x be an extreme point of the polytope $P = \{x \in [0,1]^E : x(\delta_E(S)) \geq f(S)\}$ with $0 < x_e < 2/3$ for all $e \in E$, where $E \neq \emptyset$. Then at least one of the following holds.*

- **(i)** *There is an x-core C with $d_E(C) \in \{2,3\}$ and $f(C) = 1$; moreover, no edge in E has both ends in C.*
- **(ii)** *There are $u, v \in U$ with $d_H(u,v) = 0$ and $d_I(u,v) \geq \mu$, where $\mu = \lceil (k-1)/2 \rceil$.*

Furthermore, such u, v or such C can be found in polynomial time.

Proof. By Lemma 15, if there exists an x-defining laminar family, then there is an x-core C such that $d_E(C) \in \{2,3\}$, $f(C) \in \{1,2\}$, and no edge in E has both ends in C. But $f(C) = 2$ is not possible, as otherwise there is $e \in \delta_E(C)$ with $x_e \geq 2/3$. Hence in this case (i) holds.

We now show that if no laminar x-defining family exists then (ii) holds. Similarly to Lemma 17 we conclude that there is a pair A, B of x-tight f-positive sets that cross such that $A \cap B = \{u\}$ and $A \setminus B = \{v\}$ for some $u, v \in U$.

We claim that if $d_H(u,v) = 0$ then $d_I(u,v) \geq \mu$. By the assumption of the lemma, $d_I(u)+d_H(u) \geq k-1$ and $d_I(v)+d_H(v) \geq k-1$. Since A is f-positive, $d_I(A)+d_H(A) \leq k-1$. Thus we get (see Fig. 3(a))

$$k-1 \geq d_I(A)+d_H(A) = [d_I(u)+d_H(u)]+[d_I(v)+d_H(v)]-2d_I(u,v) \geq 2(k-1)-2d_I(u,v) \; .$$

Consequently, $2d_I(u,v) \geq k-1$, as claimed.

We prove that if $d_H(u,v) = 1$ then there is $w \in U$ such that one of the following holds:

- $\bar{A} \cap B = \{w\}$, $d_H(u,w) = 0$, and $d_I(u,w) \geq \mu$.
- $\bar{A} \setminus B = \{w\}$, $d_H(v,w) = 0$, and $d_I(v,w) \geq \mu$.

Let $\psi(S,T) = d_I(S,T) + d_H(S,T) + x(\delta_E(S,T))$ and $\psi(S) = \psi(S,\bar{S})$. Note that

$$\psi(S) \geq \begin{cases} k-1 & \text{if } S = \{u\} \text{ or } \bar{S} = \{u\} \text{ for some } u \in U \\ k & \text{otherwise} \end{cases}$$

In particular, $\psi(S) < k$ implies that $S = \{w\}$ or $\bar{S} = \{w\}$ for some $w \in U$.

Let $W = \bar{A} \cap B$ and $Z = \bar{A} \setminus B$. Note that $\psi(A) = \psi(B) = k$ since A, B are x-tight and that $\psi(u,v) \geq d_H(u,v) + \mu = 1 + \mu$. This implies $\psi(W,Z) \leq \psi(B) - \psi(u,v) \leq k - (1+\mu)$, see Fig. 3(b). Thus we get

$$2k - 2(1+\mu) \geq 2\psi(W,Z) = \psi(W) + \psi(Z) - \psi(A) = \psi(W) + \psi(Z) - k$$

Consequently, $\psi(W) + \psi(Z) \leq 3k - 2(1+\mu) < 2k$, since $2(1+\mu) > k$. Thus $\psi(W) < k$ or $\psi(Z) < k$, and w.l.o.g. $\psi(W) < k$. This implies that $W = \{w\}$ for some $w \in U$, see Fig. 3(c). Note that $d_H(u,w) = 0$, since $d_H(u,v) = 1$ and since $d_H(u) \leq 1$ by the assumption of the lemma. Also, $d_I(B) + d_H(B) \leq k-1$, since B is f-positive. Thus we have

$$\begin{aligned} 2d_I(u,w) &= 2[d_I(u,w) + d_H(u,w)] \\ &= [d_I(u) + d_H(u)] + [d_I(w) + d_H(w)] - [d_I(B) + d_H(B)] \\ &\geq (k-1) + (k-1) - (k-1) = k-1 \; . \end{aligned}$$

The polynomial time implementation details are identical to those in Lemma 20. ◀

We now prove the connectivity guarantee. As before, let $\mathcal{R}$ be the family of subsets of V_0 that correspond to the contracted sets during the algorithm. Similarly to the previous case, we have the following sequence of lemmas.

▶ **Lemma 22.** *When a core C is found in the algorithm but not yet contracted, $d_H(C) \leq 1$, $x(\delta_E(C)) = 1$, and $d_I(C) = k - d_H(C) - x(\delta_E(C)) = k - d_H(C) - 1$. Consequently, $d_I(C) = k - 1$ if $d_H(C) = 0$ and $d_I(C) = k - 2$ if $d_H(C) = 1$.*

Proof. If $d_H(C) \geq 2$ then $d_I(C) + 2d_H(C) \geq 2\mu + 2 \geq k$, contradicting that C is a core. Since $d_E(C) \in \{2, 3\}$ and $x(\delta_E(C))$ is a positive integer, we must have $x(\delta_E(C)) = 1$, as otherwise there is $e \in \delta_E(C)$ with $x_e \geq 2/3$. ◀

▶ **Lemma 23.** *During the algorithm $d_H(u) \leq 1$ and $d_I(u) + d_H(u) = k - 1$ holds for all $u \in U$. Furthermore, $d_H(v) \leq 2$ for all $v \in V \setminus U$. Thus the algorithm terminates after $O(m)$ iterations.*

Proof. By Lemma 22, when u enters U we have $d_H(u) \leq 1$ and $d_I(u) = k - d_H(u) - 1$. This implies $f(\{u\}) = k - d_I(u) - d_H(u) - 1 = 0$. The addition of a ghost edge incident to u excludes u from U but does not increase $f(\{u\})$, hence $\{u\}$ will never enter U again. If we had $d_H(u) = 1$ when u entered U, then u will leave U with $d_H(u) = 2$ and will never enter U again. Thus $d_H(v) \leq 2$ for all $v \in V \setminus U$. The proof that the algorithm terminates after $O(m)$ iterations is identical to the proof of Lemma 14. ◀

▶ **Lemma 24.** *When a core C is found in the algorithm but not yet contracted, the following holds for any proper subset S of C.*

(i) *$d_I(S) \geq d_H(S) \cdot \mu$ and $d_I(S) \geq k - d_H(S) - 1$.*

(ii) *$d_I(S, C \setminus S) \geq \mu$.*

Proof. We prove (i). $d_I(S) \geq d_H(S) \cdot \mu$ by Lemma 5. If $S = \{u\}$ for some $u \in U$ then $d_I(u) = k - d_H(u) - 1$ by Lemma 22. Otherwise, the constraint of S is relaxed by ghost edges only, and since $\delta_E(S) \subseteq \delta_E(C)$ (by Lemma 21(i)) and $x(\delta_E(C)) \leq 1$ (by Lemma 22), we get $d_I(S) \geq k - d_H(S) - x(\delta_E(S)) = k - d_H(S) - 1$.

We prove (ii). If $d_H(S, C \setminus S) \geq 1$ then $d_I(S, C \setminus S) \geq \mu$ since $d_I(u, v) \geq \mu$ when the ghost edge uv is added. Otherwise, $d_H(S) + d_H(C \setminus S) = d_H(C) \leq 1$ and since $d_J(C) \leq k - d_H(C) - 1$ (by Lemma 22) and by part (i) we get

$$\begin{aligned} 2d_I(S, C \setminus S) &= d_I(S) + d_I(C \setminus S) - d_I(C) \\ &\geq [k - d_H(S) - 1] + [k - d_H(C \setminus S) - 1] - [k - 2d_H(C) - 1] \\ &= k - 1 - [d_H(S) + d_H(C \setminus S) - d_H(C)] = k - 1 \ , \end{aligned}$$

concluding the proof. ◀

▶ **Corollary 25.** *Let S be a cut such that $\mathcal{R} \cup \{S\}$ is laminar. Then $d_J(S) \geq k - d_H(S) - 1$ and $d_J(S) \geq d_H(S) \cdot \mu$. Furthermore, if $R_S = V$ and $S \notin R$ then $d_J(S) \geq k - d_H(S)$. Consequently, $d_J(S) \geq \max\{k - d_H(S) - 1, d_H(S) \cdot \mu\} \geq k - 2$.*

Proof. The bound $d_J(S) \geq d_H(S) \cdot \mu$ follows from Lemma 5, so we prove only the bound $d_J(S) \geq k - d_H(S) - 1$. For $S \in \mathcal{R}$ it follows from Lemmas 22, so assume that $S \notin \mathcal{R}$. If S is contained in some set in $\mathcal{R}$, then let C be the inclusion minimal set in $\mathcal{R}$ that contains S. The bound then follows from Lemma 24(i). Otherwise, the constraint of S was relaxed by ghost edges only and then $d_J(S) \geq k - d_H(S)$. ◀

▶ **Lemma 26.** *Let $R \in \mathcal{R}$ and let T be a proper subset of $\bar{R}$ such that $\mathcal{R} \cup \{T, \bar{R} \setminus T\}$ is laminar. Then $d_J(T, \bar{R} \setminus T) \geq \lceil (k-4)/2 \rceil$.*

Proof. If $d_H(T, \bar{R} \setminus T) \geq 1$ then by Lemma 5 $d_J(T, R \setminus T) \geq \mu > \lceil (k-4)/2 \rceil$. Otherwise, $d_H(R) = d_H(T) + d_H(\bar{R} \setminus T)$, and since $d_J(R) \leq k - d_H(C) + 2$ (by Lemma 22) we get

$$\begin{aligned} 2d_J(T, \bar{R} \setminus T) &= d_J(T) + d_J(\bar{R} \setminus T) - d_J(R) \\ &\geq [k - d_H(T) - 1] + [k - d_H(\bar{R} \setminus T) - 1] - [k - d_H(R) + 2] \\ &= k - 4 - [d_H(T) + d_H(\bar{R} \setminus T) - d_H(R)] = k - 4 \ , \end{aligned}$$

concluding the proof. ◀

▶ **Lemma 27.** *Suppose that $1 \leq |\mathcal{R}(S)| \leq |\mathcal{R}(\bar{S})|$.*

(i) *If R is a minimal set in $\mathcal{R}(S)$ then $d_J(R \cap S, R \setminus S) \geq \mu$.*

(ii) *If R is a unique maximal set in $\mathcal{R}(S)$ then $d_J(\bar{R} \cap S, \bar{R} \setminus S) \geq \lceil (k-4)/2 \rceil$.*

Proof. We prove (i). The minimality of R implies that if $T \in \{R \cap S, R \setminus S\}$ then $\mathcal{R} \cup \{T\}$ is laminar. Therefore by Lemma 24(ii) $d_J(R \cap S, R \setminus S) \geq \mu$.

We prove (ii). The maximality of R implies that if $T \in \{\bar{R} \cap S, \bar{R} \setminus S\}$ then $\mathcal{R} \cup \{T\}$ is laminar. Therefore by Lemma 26 $d_J(\bar{R} \cap S, \bar{R} \setminus S) \geq \lceil (k-4)/2 \rceil$. ◀

▶ **Lemma 28.** *If $1 \leq |\mathcal{R}(S)| \leq |\mathcal{R}(\bar{S})|$ then $d_J(S) \geq k - 2$.*

Proof. Suppose that there are two disjoint sets in $\mathcal{R}(S)$. Then there are two minimal sets in $\mathcal{R}(S)$ that are disjoint, say R_1, R_2; see Fig. 2(b). Then by Lemma 27(i)

$$d_J(S) \geq d_J(R_1 \cap S, R_1 \setminus S) + d_J(R_2 \cap S, R_2 \setminus S) \geq \mu + \mu > k - 2 \ .$$

Otherwise, $\mathcal{R}(S)$ is a nested family with a unique maximal set R_1 and a unique minimal set R_2; see Fig. 2(c) and note that $R_2 \subseteq R_1$ and possibly $R_1 = R_2$. Then by Lemma 27

$$d_J(S) \geq d_J(\bar{R}_1 \cap S, \bar{R}_1 \setminus S) + d_J(R_2 \cap S, R_2 \setminus S) \geq \lceil (k-4)/2 \rceil + \mu = k - 2 \ ,$$

concluding the proof. ◀

This concludes the proof of the connectivity guarantee of Algorithm 2 and thus also of Theorem 3.

References

1. J. Cheriyan and R. Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM J. Computing*, 30:528–560, 2000. doi:10.1137/S009753979833920X.
2. J. Cheriyan and L. A. Végh. Approximating minimum-cost k-node connected subgraphs via independence-free graphs. *SIAM Journal on Comput.*, 43(4):1342–1362, 2014. doi:10.1137/120902847.
3. J. Chuzhoy and S. Khanna. An $O(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design. *Theory Comput.*, 8(1):401–413, 2012. doi:10.4086/TOC.2012.V008A018.
4. A. Czumaj and A. Lingas. On approximability of the minimum-cost k-connected spanning subgraph problem. In *SODA*, pages 281–290, 1999.
5. J. Edmonds. Edge-disjoint branchings. In B. Rustin, editor, *Combinatorial Algorithms*, pages 91–96. Academic, New York, 1973.
6. J. Fackharoenphol and B. Laekhanukit. An $O(\log^2 k)$-approximation algorithm for the k-vertex connected subgraph problem. In *STOC*, pages 153–158, 2008.

7 C. G. Fernandes. A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem. *J. Algorithms*, 28(1):105–124, 1998. doi:10.1006/JAGM.1998.0931.

8 A. Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM J. Discrete Math.*, 5(1):25–53, 1992. doi:10.1137/0405003.

9 A. Frank and É. Tardos. An application of submodular flows. *Linear Algebra and its Applications*, 114-115:329–348, 1989.

10 G. N. Fredrickson and J. F. JáJá. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.*, 10(2):270–283, 1981. doi:10.1137/0210019.

11 G. N. Fredrickson and J. F. JáJá. On the relationship between the biconnectivity augmentation and traveling salesman problem. *Theoretical Computer Science*, 19:189–201, 1982. doi:10.1016/0304-3975(82)90059-7.

12 H. N. Gabow and S. Gallagher. Iterated rounding algorithms for the smallest k-edge connected spanning subgraph. *SIAM J. Computing*, 41(1):61–103, 2012. doi:10.1137/080732572.

13 H. N. Gabow, M. X. Goemans, É. Tardos, and D. P. Williamson. Approximating the smallest k-edge connected spanning subgraph by LP-rounding. *Networks*, 53(4):345–357, 2009. doi:10.1002/NET.20289.

14 D. E. Hershkowitz, N. Klein, and R. Zenklusen. Ghost value augmentation for k-edge-connectivity. In *STOC*, pages 1853–1864, 2024. Full version available at arXiv:2311.09941.

15 K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001. doi:10.1007/S004930170004.

16 A. R. Karlin, N. Klein, S. Oveis Gharan, and X. Zhang. An improved approximation algorithm for the minimum k-edge connected multi-subgraph problem. In *STOC*, pages 1612–1620, 2022.

17 S. Khuller. Approximation algorithms for for finding highly connected subgraphs. In D. S. Hochbaum, editor, Chapter 6 in *Approximation Algorithms for NP-hard problems*, pages 236–265. PWS, 1995.

18 S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *J. ACM*, 41(2):214–235, 1994. doi:10.1145/174652.174654.

19 G. Kortsarz and Z. Nutov. Approximating node connectivity problems via set covers. *Algorithmica*, 37(2):75–92, 2003. doi:10.1007/S00453-003-1027-4.

20 L. C. Lau, R. Ravi, and M. Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.

21 L. C. Lau and H. Zhou. A unified algorithm for degree bounded survivable network design. *Math. Program.*, 154(1-2):515–532, 2015. doi:10.1007/S10107-015-0858-5.

22 L. Lovász. *Combinatorial Problems and Exercises*. North-Holland Publishing Co., Amsterdam, 1979.

23 Z. Nutov. Approximating minimum-cost connectivity problems via uncrossable bifamilies. *ACM Trans. Algorithms*, 9(1):1:1–1:16, 2012. doi:10.1145/2390176.2390177.

24 Z. Nutov. Small ℓ-edge-covers in k-connected graphs. *Discret. Appl. Math.*, 161(13-14):2101–2106, 2013.

25 Z. Nutov. Approximating minimum-cost edge-covers of crossing biset-families. *Combinatorica*, 34(1):95–114, 2014. doi:10.1007/S00493-014-2773-4.

26 Z. Nutov. Improved approximation algorithms for minimum cost node-connectivity augmentation problems. *Theory Comput. Syst.*, 62(3):510–532, 2018. doi:10.1007/S00224-017-9786-5.

27 Z. Nutov. The k-connected subgraph problem. In T. Gonzalez, editor, *Approximation Algorithms and Metaheuristics*, chapter 12, pages 213–232. Chapman & Hall, 2018.

28 Z. Nutov. A $(4+\epsilon)$-approximation for k-connected subgraphs. *J. Comput. Syst. Sci.*, 123:64–75, 2022.

29 D. Pritchard. k-edge-connectivity: Approximation and LP relaxation. In *WAOA*, pages 225–236, 2010.

30 A. Sebö and J. Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014. doi:10.1007/S00493-014-2960-3.

31 M. Singh and L. C. Lau. Approximating minimum bounded degree spanning trees to within one of optimal. *J.of the ACM (JACM)*, page 62.1, 2015.

32 V. Traub and R. Zenklusen. A $(1.5+\epsilon)$-approximation algorithm for weighted connectivity augmentation. In *STOC*, pages 1820–1833, 2023.

33 V. V. Vazirani. *Approximation Algorithms*. Springer, 2010.

Compact Representation of Semilinear and Terrain-Like Graphs

Jean Cardinal
Université libre de Bruxelles (ULB), Belgium

Yelena Yuditsky
Université libre de Bruxelles (ULB), Belgium

Abstract

We consider the existence and construction of *biclique covers* of graphs, consisting of coverings of their edge sets by complete bipartite graphs. The *size* of such a cover is the sum of the sizes of the bicliques. Small-size biclique covers of graphs are ubiquitous in computational geometry, and have been shown to be useful compact representations of graphs. We give a brief survey of classical and recent results on biclique covers and their applications, and give new families of graphs having biclique covers of near-linear size.

In particular, we show that semilinear graphs, whose edges are defined by linear relations in bounded dimensional space, always have biclique covers of size $O(n \operatorname{polylog} n)$. This generalizes many previously known results on special classes of graphs including interval graphs, permutation graphs, and graphs of bounded boxicity, but also new classes such as intersection graphs of L-shapes in the plane. It also directly implies the bounds for Zarankiewicz's problem derived by Basit, Chernikov, Starchenko, Tao, and Tran (*Forum Math. Sigma*, 2021).

We also consider capped graphs, also known as terrain-like graphs, defined as ordered graphs forbidding a certain ordered pattern on four vertices. Terrain-like graphs contain the induced subgraphs of terrain visibility graphs. We give an elementary proof that these graphs admit biclique partitions of size $O(n \log^3 n)$. This provides a simple combinatorial analogue of a classical result from Agarwal, Alon, Aronov, and Suri on polygon visibility graphs (*Discrete Comput. Geom.* 1994).

Finally, we prove that there exists families of unit disk graphs on n vertices that do not admit biclique coverings of size $o(n^{4/3})$, showing that we are unlikely to improve on Szemerédi-Trotter type incidence bounds for higher-degree semialgebraic graphs.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Graph algorithms analysis

Keywords and phrases Biclique covers, intersection graphs, visibility graphs, Zarankiewicz's problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.67

Related Version *Full Version*: `https://arxiv.org/abs/2507.00252` [20]

Funding *Jean Cardinal*: Jean Cardinal was supported by the Fonds de la Recherche Scientifique - FNRS under Grant n° T003325F.
Yelena Yuditsky: Yelena Yuditsky was supported by the Fonds de la Recherche Scientifique - FNRS as a Postdoctoral Researcher.

Acknowledgements This work was inspired by discussions at the Tenth Annual Workshop on Geometry and Graphs (WoGaG'23), held at the Bellairs Research Institute, (McGill), Barbados, in February 2023.

1 Introduction

Graph covering and partitioning is a well-studied topic in graph theory with numerous connections to real-world problems, see for example the survey of Schwartz [68]. Given a graph G, a *cover* of G is a collection $\mathcal{H}$ of subgraphs of G such that each edge of G is contained in at least one of the subgraphs in $\mathcal{H}$, that is, $E(G) \subseteq \bigcup_{H \in \mathcal{H}} E(H)$. A *partition*


© Jean Cardinal and Yelena Yuditsky;
licensed under Creative Commons License CC-BY 4.0

of G is a collection $\mathcal{H}$ of subgraphs of G such that each edge of G is contained in *exactly* one of the subgraphs in $\mathcal{H}$. Clearly, any partition of a graph is also a cover. The subgraphs in the cover (or partition) can be chosen in different ways, for example, they can be chosen to be cliques, complete bipartite graphs (to which we refer as *bicliques*), cycles, paths, and other graphs (see [68] for a list of results for each of those graph classes).

A natural problem is, for a given graph, to determine the smallest possible number of subgraphs in a cover. We are interested in a related parameter of the cover. Let G be a graph and let $\mathcal{H}$ be a cover of G with any type of subgraphs. The *size* of a cover $\mathcal{H}$ is $\sum_{H\in\mathcal{H}} |V(H)|$. Clearly, those parameters are related. If G has a cover with ℓ subgraphs, then there is also a cover of G of size ℓn where $n = |V(G)|$.

In this work, we focus on the size of covers with bicliques, to which we refer as *biclique covers*. See Figure 1 for an example of biclique cover of a graph.

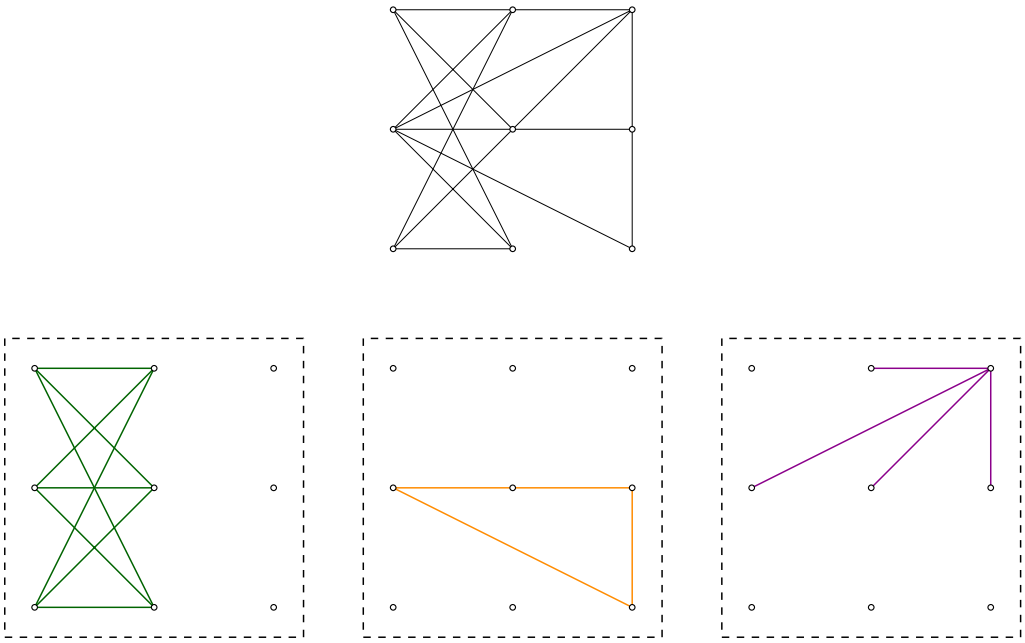


Figure 1 An example of a graph and a biclique cover with three bicliques of total size 15.

Biclique covers have applications to the multicommodity flow problem [47], quantified Boolean formulas [55], and communication complexity of boolean functions [51]. In a seminal paper, Feder and Motwani [41] showed that biclique covers of small sizes can be used as compact representations of graphs, on which many computational problems can be solved more efficiently than on standard representations. The minimum size of a biclique cover of a graph is therefore sometimes referred to as its *representation complexity* [37].

1.1 Our results

We study biclique covers for various classes of graphs defined in geometric terms. We focus on graph classes for which there exist biclique covers of size $O(n \operatorname{polylog} n)$, where n is the number of vertices. Some of the terms used below are defined in the next section 1.2.

Our contribution is threefold. We first consider *semilinear graphs*, defined as semialgebraic graphs for which the defining functions are linear. This class of graphs has first been defined by Basit, Chernikov, Starchenko, Tao, and Tran [13], and further studied by Tomon [73].

We generalize a number of results above by showing that semilinear graphs have biclique partitions of size $O(n \operatorname{polylog} n)$. This in particular gives a simple proof of the bound on Zarankiewicz's problem for semilinear graphs shown by Basit et al. [13]. Semilinear graphs include many previously studied graph classes such as interval graphs, threshold graphs, permutation graphs, bounded-dimension comparability graphs, bounded-boxicity graphs, and intersection graphs of L-shapes and other orthogonal shapes in the plane. For certain restricted classes of semilinear graphs, we establish improved bounds on the size of a biclique cover compared to those obtainable from the general bounds.

We then show that *terrain-like graphs*, also known as *capped graphs* [34] have biclique partitions of size $O(n \log^3 n)$. These graphs contain in particular induced subgraphs of *terrain visibility graphs*. This result can be seen as a combinatorial variant of the influential result of Agarwal, Alon, Aronov, and Sharir [3] on compact representation of visibility graphs of polygons, since terrain-like graphs include terrain visibility graphs. The two results, however, are incomparable, since there exist terrain-like graphs that are not visibility graphs of terrains [11].

Finally, we answer a question asked by Csaba Tóth [74]: Can unit disk graphs have near-linear biclique covers? Note that the existence of such covers is not ruled out by a counting argument, since there are $2^{O(n \log n)}$ unit disk graphs on n vertices [67]. Yet, we answer the question in the negative: There exist unit disk graphs on n vertices, every biclique cover of which has size $\Omega(n^{4/3})$, which is essentially tight. This is proved by considering the standard examples of configurations of points and lines that are tight for the Szemerédi-Trotter Theorem [63], and applying a charging argument. Together with our upper bound for semilinear graphs, this result provides a delineation of what classes of graphs have near-linear biclique covers; as soon as we allow the functions defining a semialgebraic graph to be of degree two, we may obtain strongly superlinear lower bounds on the size of biclique covers.

1.2 Background

Before detailing our results, we first give a brief survey of known results and applications of biclique covers and partitions. We take the opportunity here to gather results that are sometimes considered as folklore, but were never thoroughly compiled.

1.2.1 Covers and partitions in general graphs

The minimal number of bicliques needed to cover a graph has been studied by Tuza [75], Rödl and Ruciński [66], and Jukna and Kulikov [51], among others. As mentioned earlier, we are interested in the size of a biclique cover, which, to recall, is defined as the sum of the number of vertices across all the bicliques in the cover. Let us first note that if G is an n-vertex graph with $O(n)$ edges, then G has a trivial biclique cover of size $O(n)$. Hence the question of bounding the size of a biclique cover is not relevant in classes of graphs with linearly many edges, such as bounded-treewidth graphs or planar graphs.

A cover of size $O(n \log n)$ of the complete graph K_n can be obtained recursively as follows: first split the set of vertices into two disjoint subsets of equal size, and add the corresponding biclique to the cover, then recurse on both subsets. The cover we obtain in this way is also a biclique partition with n bicliques.

A bound on the size of a biclique cover of an arbitrary graph was proved by Chung, Erdős, and Spencer [30], and later by Tuza [75].

▶ **Theorem 1** ([30, 75]). *Let G be a graph on n vertices. Then there exists a biclique cover of G of size $O(n^2/\log n)$, and this bound is tight.*

We observe a connection between the size of a biclique cover and constructions of graphs with many edges and without a complete bipartite graph $K_{t,t}$ as a subgraph, also known as the Zarankiewicz problem. We refer, for example, to Conlon [31] for a construction of such graphs and to Smorodinsky [70] for a survey on the Zarankiewicz problem in geometric graphs. Let G be a graph without a $K_{t,t}$ subgraph and let $\mathcal{H}$ be a biclique cover of G. Let $H \in \mathcal{H}$, then $|E(H)| \leq t|V(H)|$ and therefore $\sum_{H\in\mathcal{H}} |V(H)| \geq \sum_{H\in\mathcal{H}} |E(H)|/t \geq |E(G)|/t$. Hence we can deduce the following observation.

▶ **Observation 2.** *Let G be a graph without a $K_{t,t}$ subgraph, for some $t \in \mathbb{N}$, and with a biclique cover of size s. Then G has at most $t \cdot s$ edges.*

For classes of graphs that have biclique partition of size $O(n \operatorname{polylog} n)$, like the ones studied in this paper, this directly gives an upper bound of $O(n \operatorname{polylog} n)$ on the number of edges of these graphs when $K_{t,t}$ is forbidden, for some constant t.

Let $\mathcal{F}$ be a class of graphs and let $\mathcal{F}_n$ be the graphs in $\mathcal{F}$ with exactly n vertices. If any graph $G \in \mathcal{F}_n$ has a biclique cover of size $s(n)$ then $|\mathcal{F}_n| \leq 2^{s(n)\lceil \log n\rceil + 2s(n)}$. Indeed, every graph can be encoded by a binary string of length at most $s(n)\lceil \log n\rceil + 2s(n)$ where every vertex is encoded by a binary string of length $\lceil \log n\rceil$ and at most $2s(n)$ additional bits separating the bicliques and defining the bipartition inside each biclique. Based on the above, we make the following observation.

▶ **Observation 3.** *Let $\mathcal{F}$ be a family of graphs and let $s(n)$ be the maximum size of a biclique cover of a graph in $\mathcal{F}_n$, then $|\mathcal{F}_n| \leq 2^{O(s(n)\log n)}$.*

Note that for many classes of geometric graphs, the upper bound in the above observation is significantly worse than the best known upper bounds [67].

1.2.2 Algorithms

Feder and Motwani [41] observed that biclique covers can be used to construct compressed representations of graphs, on which several computational problems can be solved efficiently. In a compressed representation, each biclique in the cover is replaced by a subgraph whose number of edges is proportional to the size of the biclique. This can be interpreted as a *sparsification* procedure, that preserves features of the initial graph while reducing the number of edges. A simple application of this idea is to the all-pairs shortest paths problem.

▶ **Lemma 4** ([41])**.** *Given a biclique cover of size s of a graph G on n vertices, then the breadth-first search tree rooted at any vertex of G can be computed in time $O(s)$, hence the (unweighted) all-pairs shortest paths problem can be solved in time $O(n \cdot s)$.*

We refer to Chan and Skrepetos [25] for a discussion on the application of this result to geometric intersection graphs.

Similarly, Feder and Motwani proved that a maximum matching of a bipartite graph given in compressed form as a biclique cover of size s can be found in time $O(\sqrt{n} \cdot s)$ [41, Theorem 4.2]. Using a state-of-the-art maximum flow algorithm [29, 76], Cabello, Cheng, Cheong, and Knauer [18] gave the following improvement.

▶ **Lemma 5** ([18])**.** *Given a biclique cover of size s of a bipartite graph G, the maximum matching problem on G can be solved in time $O(s^{1+\varepsilon})$ for any $\varepsilon > 0$.*

Note that in these results, we assume that a biclique cover is given, and do not take into account the time required to compute it. Ideally, if the graph is given in some kind of implicit form (for instance as a collection of geometric object, of which the graph is the intersection graph), we may hope to be able to compute the biclique cover in time proportional to its size, and then apply the above lemmas.

1.2.3 Spanners

Let $t \in \mathbb{N}$ and let G be a graph. A *t-hop spanner* is a subgraph G' of G such that for any $uv \in E(G)$, there is a path of length at most t in G' between u and v. The following was observed by Conroy and Tóth [33].

▶ **Lemma 6** ([33]). *If G has a biclique cover of size s then there is a subgraph G' of G which is a 3-hop spanner with s edges.*

This is achieved by simply replacing every biclique of the decomposition by two stars, with centers on either side of the bipartition.

Biclique covers of intersection graphs of boxes in constant dimension are also used in a recent preprint by Bhore, Chan, Huang, Smorodinsky and Tóth [14] to obtain 2-hop spanners for fat axis-aligned boxes.

1.2.4 Biclique covers and range searching

Biclique covers and partitions of geometric graphs are natural byproducts of *range searching* algorithms and data structures in computational geometry. The range searching problem can be defined as that of preprocessing a collection of points in $\mathbb{R}^d$ so as to quickly answer queries about the subset of points contained in a given region, called the range. There, bicliques naturally arise as pairs of subsets of points and ranges, such that all points are contained in all ranges. Classical data structures for orthogonal range searching, where ranges are axis-aligned boxes, include segment trees and range trees [35, Chapters 5 and 10]. Halfspace and simplex range queries are typically answered using cuttings and partition trees [58, 59, 26, 60, 23]. More recently, polynomial partitioning techniques, stemming from the seminal paper from Guth and Katz [48], have been used to solve semialgebraic range queries [61, 37, 4, 19, 7]. These structures naturally yield biclique covers of the corresponding incidence graphs. In turn, running times for the offline range searching problem are closely related to incidence bounds [63], an early example of which is the Szemerédi-Trotter theorem bounding the number of incidences between points and lines in the plane. Quoting Agarwal, Ezra, and Sharir [7]: "there is a general belief that the two problems are closely related, and that the running time of (at least off-line) range queries should be almost the same as the number of incidences between points and the corresponding curves/surfaces that bound these regions". It is therefore not surprising that incidence bounds often coincide with representation complexities.

We summarize the known upper bounds stemming from this line of work. Let P be a set of points and let R be a set of geometric objects in $\mathbb{R}^d$. An *incidence graph* $I(P, R)$ is a bipartite graph with the bipartition (P, R) where there is an edge between two elements $p \in P$ and $r \in R$ if and only if $p \in r$. We henceforth assume that every set of points or geometric objects contains at most n elements. The following results are known (the $\tilde{O}$ notation omits polylogarithmic factors):

- Incidence graphs of points and axis-parallel boxes in $\mathbb{R}^d$, for some constant d, have biclique partitions of size $O(n \log^d n)$. This can be deduced from [6, Section 3] and [35, Chapter 5], for instance.
- Incidence graphs of points and halfspaces in $\mathbb{R}^d$, for some constant d, have biclique partitions of size $\tilde{O}(n^{2d/(d+1)})$ [57].
- Incidence graphs of points and hyperplanes in $\mathbb{R}^d$, for some constant d, have biclique partitions of size $\tilde{O}(n^{2d/(d+1)})$ [17].
- Incidence graphs of points and unit disks have biclique partitions of size $\tilde{O}(n^{4/3})$ [54].
- Incidence graphs of points and disks (of arbitrary radii) have biclique partitions of size $\tilde{O}(n^{15/11})$ [7].

In all these cases, the biclique covers can be constructed in time proportional to their size.

Results by Erickson [39] on Hopcroft's problem make the connection between range searching problems and biclique covers explicit. It is shown, among other results, that the running time of a type of a partitioning algorithm for the counting version of Hopcroft's problem with respect to a set of points P and hyperplanes H is bounded from below by the size of the biclique cover of $I(P, H)$.

Biclique covers also have other applications in computational geometry. For instance Agarwal and Varadarajan [8] use them to compute approximations of polygonal chains, and biclique partitions of pairs of points at bounded distance from each other are also at the heart of the expander-based optimization method proposed by Katz and Sharir [54].

A graph G is an *intersection graph* for a class of geometric objects if its vertices can be mapped to objects in the class so that two vertices are adjacent if and only if the corresponding objects have a nonempty intersection. As mentioned by Agarwal et al. [7], *interval graphs*, intersection graph of intervals on the real line, have biclique partitions of size $O(n \log n)$. More generally, it was shown by Chan [21] that intersection graphs of axis-aligned boxes in $\mathbb{R}^d$ have a biclique cover of size $O(n \log^d n)$. The *boxicity* of a graph G is the minimum d such that G is an intersection graph of axis-aligned boxes in $\mathbb{R}^d$. Hence this result shows that bounded-boxicity graphs have near-linear size biclique covers. For intersection graphs of unit disks in the plane, a bound of $\tilde{O}(n^{4/3})$ on the size of a biclique cover can be derived from the bound on the size of a biclique cover in the incidence graph between a set of points and unit disks in the plane. Segment intersection graphs have biclique covers of size $\tilde{O}(n^{4/3})$ [8, 7]. Finally, Chazelle, Edelsbrunner, Guibas, and Sharir [27] proved that the bipartite intersection graph between a set of n disjoint blue segments and a set of n disjoint red segments in the plane has a biclique cover of size $O(n \log^3 n)$.[1]

Many natural classes of geometric graphs are contained in the class of semialgebraic graphs [10, 32, 42, 71]. A graph $G = (V, E)$ is a *semialgebraic graph* of description complexity t and dimension d if the vertices in V can be mapped to points in $\mathbb{R}^d$ so that the presence of an edge is defined by the sign patterns of t polynomial functions of the corresponding pair of points. More precisely, there must exist:

- a map $\varphi : V \mapsto \mathbb{R}^d$,
- t polynomials $f_1, \ldots, f_t \in \mathbb{R}[x_1, \ldots, x_d, y_1, \ldots y_d]$ each of degree at most t in $2d$ variables,
- a boolean function $\Phi : \{\mathrm{T}, \mathrm{F}\}^{3t} \to \{\mathrm{T}, \mathrm{F}\}$,

such that for any $u, v \in V$,

$$uv \in E \Leftrightarrow \Phi\left(\{f_i(\varphi(u), \varphi(v)) < 0, f_i(\varphi(u), \varphi(v)) = 0, f_i(\varphi(u), \varphi(v)) \leq 0\}_{i \in [t]}\right) = T.$$

For example, two closed disks of center respectively (a, b) and (c, d) and radius r and s, for instance, have a nonempty intersection if and only if $f(a, b, r, c, d, s) \leq 0$, where $f(a, b, r, c, d, s) = (a-c)^2+(b-d)^2-(r+s)^2$. Hence disk intersection graphs are semialgebraic graphs of dimension $d = 3$. Note that the union of constantly many semialgebraic graphs is also a semialgebraic graph.

The following result has been proved by Do [37]. A computationally efficient version can be found in Agarwal, Aronov, Ezra, Katz, and Sharir [5, Corollary A.5].

[1] The result is not explicitly stated, but can be deduced from their method for reporting intersections, involving so-called *hereditary segment trees*. See the full version of the paper [20] for details. See also Palazzi and Snoeyink [64].

▶ **Theorem 7** ([37, 5]). *Let G be a semialgebraic graph on n vertices of constant description complexity t and constant dimension d. Then for any $\varepsilon > 0$, G has a biclique cover of size $O(n^{2d/(d+1)+\varepsilon})$, where the constants in the O depend on ε, t, and d.*

Chan, Cheng, and Zheng [24, Final Remarks] showed that the intersection graph of n algebraic arcs in the plane of constant description complexity has a biclique cover of size $O(n^{3/2+\varepsilon})$ and can be found in a similar running time for any $\varepsilon > 0$.

1.2.5 Ordered graphs and visibility graphs

Visibility is a classical theme in computational geometry [35, Chapter 15] [46, 62]. A visibility graph is typically defined on a set of points in the plane, such that two points form an edge if and only if the line segment between them does not hit any obstacle. *Polygon visibility graphs*, for instance, are defined on the set of vertices of a simple closed polygon, and two vertices are adjacent in the graph if and only if the line segment between them is contained in the polygon. A notable result due to Agarwal, Alon, Aronov, and Suri [3] states that polygon visibility graphs have biclique partitions of size $O(n \log^3 n)$. The proof involves an reduction to bichromatic segment intersection graphs [28] and their compact representation based on segment trees by Chazelle et al. [27].

In what follows we consider *ordered graphs*, defined as graphs $G = (V, E)$ where the set of vertices V is totally ordered. For simplicity, we will often assume that $V = [n] =: \{1, 2, \ldots, n\}$ with the natural ordering on $\mathbb{N}$. An ordered graph $([n], E)$ is a *capped graph* if for any four vertices $i < j < k < \ell$, if $ik \in E$ and $j\ell \in E$, then $i\ell \in E$ [34]. See Figure 2 for an illustration.

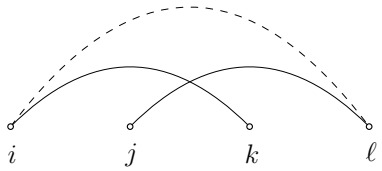


Figure 2 The forbidden ordered subgraph in a capped graph. The dashed curve represents a non-edge: If ik and $j\ell$ are edges, then $i\ell$ must be an edge.

This property is also sometimes referred to as the *X-property*, and capped graphs are also referred to as *terrain-like graphs* [43, 12, 45].

A *terrain visibility graph* is defined on the set of vertices of an x-monotone polygonal line in the plane, also called a terrain, where two vertices are adjacent in the graph if and only if the open line segment between them lies completely above the terrain [1, 40, 11, 52]. Terrain visibility graphs have applications to time series analysis [56], and have been used for instance in medical contexts [9, 72].

It is not difficult to realize that terrain visibility graphs are capped graphs. Indeed, if we are given four points in the order $i < j < k < \ell$ along the terrain such that there is a line of sight between i and k and between j and ℓ, then there must be one between i and ℓ. In fact, capped graphs also contain induced subgraphs of terrain visibility graphs, as well as visibility graphs involving points on an arbitrary, not necessary polygonal, x-monotone curve. See Figure 3 for an example.

Persistent graphs are capped graphs that also satisfy the so-called *bar property*: for any edge of the form ik such that $k \geq i + 2$, there exists j such that $i < j < k$ and both ij and jk are also edges [11, 44]. It is known that terrain visibility graphs are also persistent, and it was once conjectured that they form the same class. This conjecture was disproved by Ameer, Gibson-Lopez, Krohn, Soderman, and Wang [11].

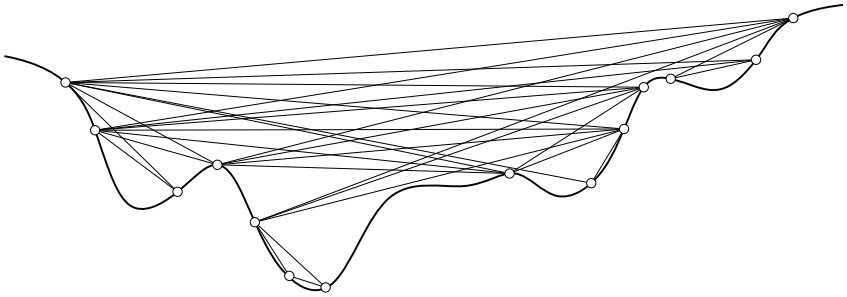


Figure 3 The visibility graph of a set of points on an x-monotone curve. When vertices are ordered from left to right, this is an example of capped graph.

Algorithms on terrain visibility graphs and terrain-like graphs have been studied by Katz, Saban, and Sharir [53], Froese and Renken [43], and De Berg, Van Beusekom, Van Mulken, Verbeek, and Wulms [36]. The Zarankiewicz problem for polygon visibility graphs was recently studied by Ackerman and Keszegh [2].

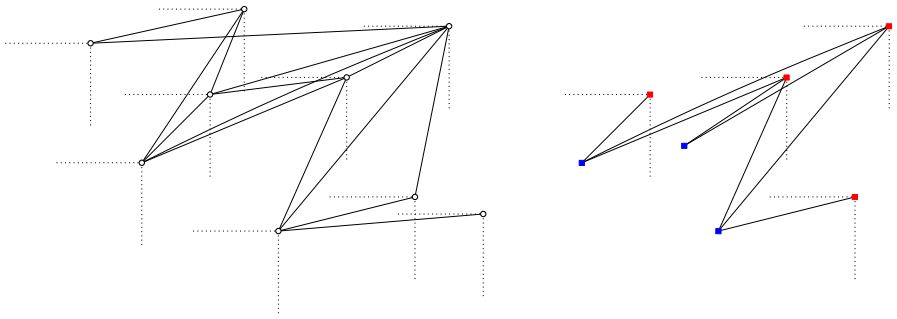
1.3 Plan of the paper

In Section 2, we construct near-linear biclique partitions for comparability graphs and bigraphs of bounded dimension. This will be used as a building block for many of the subsequent results. In Section 3, we state and prove our main result on the existence and construction of near-linear biclique partitions for any semilinear graphs. In Section 4, we give our main result on terrain-like graphs. In Section 5 we consider lower bounds on the size of biclique covers.

2 d-dimensional comparability graphs

A graph $G = (V, E)$ is a *comparability graph* if there exists a partial ordering $P(G)$ of V such that two vertices are adjacent if and only if they are comparable in $P(G)$. One may wonder if comparability graph could have small biclique covers. A simple counting argument rules this out: all bipartite graphs are comparability graphs, and there are $2^{\Omega(n^2)}$ bipartite graphs on n vertices. From Observation 3, there must exist comparability graphs all biclique covers of which have size $\Omega(n^2/\log n)$, which from Theorem 1 is not better than for general graphs.

We therefore restrict our attention to *d-dimensional comparability* graphs. A partial ordering P has dimension d if there is a collection $\mathcal{L} = \{L_1, L_2, \ldots, L_d\}$ of linear extensions of P such $P = \bigcap_{i=1}^{d} L_i$. A comparability graph G has dimension d if there is a partial ordering $P(G)$ of V of dimension d. Equivalently, a d-dimensional comparability graph is a graph $G = (V, E)$ for which there exists a map $\varphi : V \mapsto \mathbb{R}^d$ such that any two vertices $v, u \in V$ are adjacent if and only if they are comparable with respect to φ, that is either $\varphi(v) \prec \varphi(u)$ or $\varphi(v) \succ \varphi(u)$, where $a \prec b$ if and only if $a_i < b_i$ for all $i \in [d]$. We can assume without loss of generality that no pair of points in $\varphi(V)$ share a coordinate. See Figure 4 for an example of a two-dimensional comparability graph. Note that two-dimensional comparability graphs are also known as *permutation graphs* [16]. We also define a bipartite version of a d-dimensional comparability graph. A *d-dimensional comparability bigraph* is a bipartite graph $G = (L \cup R, E)$ for which there exists a map $\varphi : L \cup R \mapsto \mathbb{R}^d$ such that no pair of points in $\varphi(L) \times \varphi(R)$ share a coordinate, and such that for any pair $(\ell, r) \in L \times R$, $\ell r \in E$



(a) A two-dimensional comparability graph. **(b)** A two-dimensional comparability bigraph.

Figure 4 Comparability graphs and bigraphs.

if and only if $\varphi(\ell) \prec \varphi(r)$. We also adopt the convention that 0-dimensional comparability bigraphs are complete bipartite graphs, where $E = L \times R$. See Figure 4 for an example of a two-dimensional comparability graph. For both graph classes, we refer to the function φ as the *embedding map* of G.

We proceed by showing two key theorems concerning the size of a biclique cover of d-dimensional comparability bigraphs and graphs. The proofs rely on a simple induction for the *dominating pairs* problem [65, 22].

▶ **Theorem 8.** *Any d-dimensional comparability bigraph on n vertices has a biclique partition of size $O(n \log^d n)$.*

Proof. We prove the theorem by induction on d. For $d = 0$, the bigraph is a biclique, hence has a trivial partition of size $O(n)$.

Now let $d \geq 1$ and assume that the theorem holds for any $d' < d$. Let G be a d-dimensional comparability bigraph and let φ be its embedding map in $\mathbb{R}^d$. Let $p \in \mathbb{R}$ be a point on the dth axis such that the orthogonal hyperplane H through p divides the points in $L \cup R$ into two parts of size at most $\lceil n/2 \rceil$. Let φ_d be the restriction of φ to the dth coordinate and let $\varphi_{1,\dots,d-1}$ be the restriction of φ to the first $(d-1)$ coordinates. Let $L' \subseteq L$ be the set of vertices $\ell \in L$ for which $\varphi_d(\ell) < p$ and let $R' \subseteq R$ be the of vertices $r \in R$ for which $\varphi_d(r) > p$. Observe that $\ell r \in E$, for $\ell \in L'$ and $r \in R'$, if and only if $\varphi_{1,\dots,d-1}(\ell) \prec \varphi_{1,\dots,d-1}(r)$. Let G' be a $(d-1)$-dimensional comparability bipartite subgraph of G induced on $L' \cup R'$ with the embedding map $\varphi_{1,\dots,d-1}$ obtained by projecting those points on H. By induction, G' has a biclique partition of size $O(n \log^{d-1} n)$. It remains to recurse twice on each side of H with half the points. Let $S(n)$ be the size of a biclique partition of G. The total size $S(n)$ therefore obeys the recurrence

$$S(n) \leq 2S(\lceil n/2 \rceil) + O(n \log^{d-1} n) = O(n \log^d n).$$ ◀

The same method gives an upper bound on the size of a biclique partition of d-dimensional comparability graphs. We omit the proof here.

▶ **Theorem 9.** *Any d-dimensional comparability graph on n vertices has a biclique partition of size $O(n \log^d n)$.*

3 Semilinear graphs

Semilinear graphs were recently introduced by Basit, Chernikov, Starchenko, Tao and Tran [13], and form a subclass of semialgebraic graphs. A graph $G = (V, E)$ is a *semilinear graph of complexity t* if the vertices in V can be mapped to points in $\mathbb{R}^d$ and there exist t *linear* functions $f_1, \dots, f_t : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that the edges of G are defined by the sign patterns of those functions. More precisely, let F and T denote false and true respectively. Then G is semilinear if there exist:

- a map $\varphi : V \mapsto \mathbb{R}^d$,
- t linear functions $f_1, \dots, f_t \in \mathbb{R}[x_1, \dots, x_d, y_1, \dots y_d]$ in $2d$ variables,
- a boolean function $\Phi : \{\mathrm{T}, \mathrm{F}\}^{3t} \to \{\mathrm{T}, \mathrm{F}\}$,

such that for any $u, v \in V$,

$$uv \in E \Leftrightarrow \Phi\left(\{f_i(\varphi(u), \varphi(v)) < 0, f_i(\varphi(u), \varphi(v)) = 0, f_i(\varphi(u), \varphi(v)) \leq 0\}_{i \in [t]}\right) = T.$$

The bounded-dimensional comparability graphs of the previous section are simple examples of semilinear graphs. Interval graphs are another. Indeed, two intervals $[a, b]$ and $[c, d]$ intersect if and only if $a \leq d$ and $b \geq c$. A similar condition can be written for the intersection of two boxes in $\mathbb{R}^d$. Furthermore the union of constantly many such graphs is also a semilinear graph.

Tomon [73] observed that we can restrict the definition above to boolean formulas in disjunctive normal form, and involving only strict inequalities. A graph $G = (V, E)$ is *dnf-semilinear of complexity (t, ℓ)* if there exist a map $\varphi : V \mapsto \mathbb{R}^d$ and $t \cdot \ell$ *linear* functions $f_{i,j} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, $(i, j) \in [\ell] \times [t]$, such that for any $u, v \in V$,

$$uv \in E \Leftrightarrow \bigvee_{i \in [\ell]} \left(\bigwedge_{j \in [t]} f_{i,j}(\varphi(u), \varphi(v)) < 0 \right) = T. \tag{1}$$

As shown in [73], any semilinear graph of complexity t is a dnf-semilinear graph of complexity (t', ℓ) where t' and ℓ depend only on t, and therefore the two definitions are essentially equivalent. The second definition will be useful in our proofs.

▶ **Theorem 10.** *Let $G = (V, E)$ be a semilinear graph of constant complexity on n vertices. Then G has a biclique cover of size $O(n \operatorname{polylog} n)$.*

Proof. From the above discussion, we can assume that G is dnf-semilinear of complexity (t, ℓ), for some constants t and ℓ. Consider a function $f_{i,j}$ appearing in this definition, for some $(i, j) \in [\ell] \times [t]$. Since $f_{i,j}$ is linear, we can rewrite it as $f_{i,j}(x, y) = g_{i,j}(x) + h_{i,j}(y)$ with suitable functions $g_{i,j}, h_{i,j} : \mathbb{R}^d \to \mathbb{R}$. Let $g_i(x) = (g_{i,1}(x), \dots, g_{i,t}(x))$ and $h_i(x) = (h_{i,1}(x), \dots, h_{i,t}(x))$. Now the condition $f_{i,j}(x, y) < 0$ can be rewritten $g_{i,j}(x) < -h_{i,j}(x)$, and

$$\bigwedge_{j \in [t]} f_{i,j}(x, y) < 0 \Leftrightarrow g_i(x) \prec -h_i(y).$$

This condition defines a subgraph G_i of G whose edges are the pairs $u, v \in V$ such that either $g_i(\varphi(u)) \prec -h_i(\varphi(v))$ or $g_i(\varphi(v)) \prec -h_i(\varphi(u))$.

We now show that G_i has a biclique partition of size $O(n \log^{t+1} n)$. Let us split the set V of vertices into two subsets L and R of size at most $\lceil n/2 \rceil$. The graph whose edges are the pairs $(u, v) \in L \times R$ such that $g_i(\varphi(u)) \prec -h_i(\varphi(v))$ is a t-dimensional comparability bigraph.

Similarly, the graph whose edges are the pairs $(u, v) \in L \times R$ such that $-h_i(\varphi(u)) \succ g_i(\varphi(v))$ is also a t-dimensional comparability bigraph. These two graphs cover the edges of G_i that are in the cut (L, R). From Theorem 9, they each have a biclique cover of size $O(n \log^t n)$. It remains to cover the edges contained in L and R recursively. This yields a biclique cover of size $O(n \log^{t+1} n)$, proving our claim.

G is the union of the ℓ graphs G_i for $i \in [\ell]$. Taking the union of the biclique covers for each of the G_i yields a biclique cover of size $O(n \log^{t+1} n)$ for G, as required. ◀

Combined with Observation 2, this directly implies the result from Basit, Chernikov, Starchenko, Tao, and Tran [13].

▶ **Corollary 11** ([13]). *Let G be a semilinear graph without a $K_{t,t}$ subgraph, for some $t \in \mathbb{N}$. Then G has at most $O(n \operatorname{polylog} n)$ edges.*

Also note that the proof of Theorem 10 provides a construction algorithm running in time proportional to the output size. This holds provided that the d-dimensional comparability bigraph in Theorem 8 and the semilinear graph in Theorem 10 are given in the following implicit form: Every vertex is encoded as a point in $\mathbb{R}^d$, and the functions determining the presence of an edge are encoded in constant space.

▶ **Lemma 12.** *Given a semilinear graph G of constant complexity, we can compute a biclique partition of G of size $O(n \operatorname{polylog} n)$ in time $O(n \operatorname{polylog} n)$.*

Combining Lemma 12 with the known computational results of Lemmas 4 and 5, we obtain the following.

▶ **Theorem 13.** *Given a semilinear graph G of constant complexity on n vertices, we can compute a maximum matching of G in time $O(n^{1+\varepsilon})$, and solve the all-pairs shortest path problem on G in time $O(n^2 \operatorname{polylog} n)$.*

4 Terrain-like graphs

Recall that capped graphs, also known as terrain-like graphs, are ordered graphs such that for any four vertices $i < j < k < \ell$, if both ik and $j\ell$ are edges, then so is $i\ell$. We first define the bipartite counterpart of capped graphs, that we call *capped bigraphs*, as ordered bipartite graphs $G = (L \cup R, E)$, where the elements in L appear before the elements in R in the order, and that satisfy the same property, that is for all $i < j \in L$ and $k < \ell \in R$, we have that if $ik \in E$ and $j\ell \in E$, then $i\ell \in E$.

▶ **Lemma 14.** *Every capped bigraph is a two-dimensional comparability bigraph.*

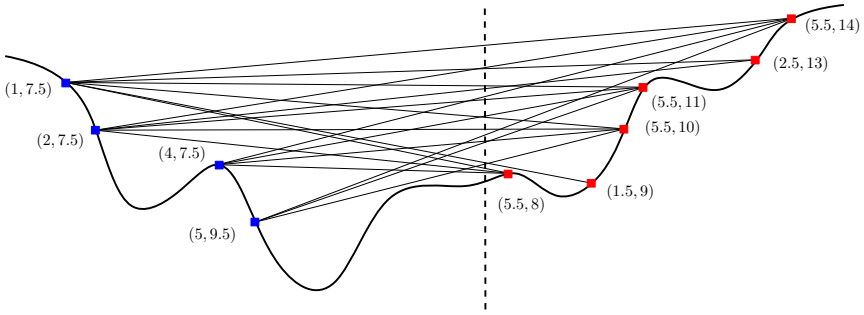
Proof. Let $G = (L \cup R, E)$ be a capped bigraph. We assume for now that G does not have any isolated vertex. Let us suppose, without loss of generality, that $L, R \subset \mathbb{N}$. Consider the map $\varphi : L \cup R \mapsto \mathbb{R}^2$ defined as

$$\varphi(\ell) = (\ell, \min\{r' \in R : \ell r' \in E\} - 1/2)$$

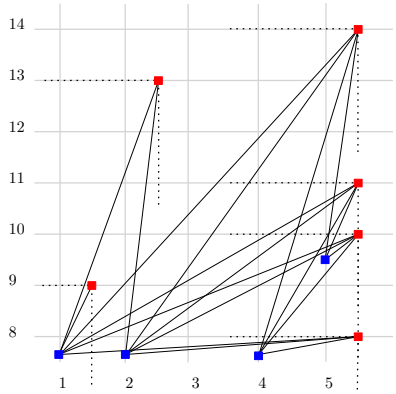
for any $\ell \in L$, and

$$\varphi(r) = (\max\{\ell' \in L : \ell' r \in E\} + 1/2, r),$$

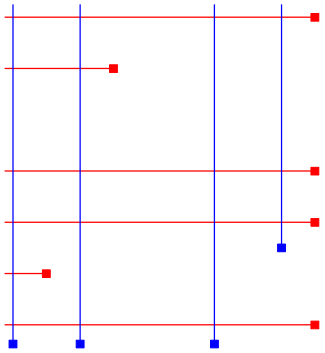
for any $r \in R$. We claim that $\ell r \in E$ if and only if $\varphi(\ell) \prec \varphi(r)$.



(a) A capped bigraph obtained from the example of Figure 3, together with the values of $\varphi(u)$ for each vertex u.



(b) A representation of the capped bigraph as a two-dimensional comparability bigraph.



(c) A representation of the capped bigraph as a two-directional orthogonal ray graph.

Figure 5 Illustration of the proof of Lemma 14.

We first show that if $\ell r \in E$, then $\varphi(\ell) \prec \varphi(r)$. Indeed, if $\ell r \in E$, it must be the case that $\ell < \max\{\ell' : \ell' r \in E\} + 1/2$. Similarly, it must be the case that $\min\{r' : \ell r' \in E\} - 1/2 < r$. Hence the point $\varphi(\ell)$ is dominated by $\varphi(r)$.

Conversely, suppose that $\varphi(\ell) \prec \varphi(r)$. Let $\ell^* = \max\{\ell' : \ell' r \in E\}$ and $r^* = \min\{r' : \ell r' \in E\}$. If either $\ell^* = \ell$ or $r^* = r$, then we have $\ell r \in E$ by definition. Let us therefore suppose that $\ell^* \neq \ell$ and $r^* \neq r$. Then, since $\varphi(\ell) \prec \varphi(r)$, we have $\ell < \ell^*$ and $r^* < r$. By definition, $\ell r^* \in E$ and $\ell^* r \in E$, hence from the X-property of capped bigraphs, it must be the case that $\ell r \in E$ as well.

Finally, note that isolated vertices can easily be added, by assigning them either a very large vertical or horizontal coordinate. ◀

The proof is illustrated in Figure 5. Note that conversely, any two-dimensional comparability bigraph $(L \cup R, E)$, as witnessed by a map φ, is a capped bigraph, where the order of the vertices in L and R is given respectively by the x and y-coordinates of the points in the image of the map φ. It suffices to observe that the X-property is satisfied. One can observe that this class of bigraphs is in fact the same as that of *interval containment bigraphs* in Huang [50] and Hell, Huang, Lin, and McConnell [49], and *two-directional orthogonal ray graphs* studied by Shrestha, Tayu, and Ueno [69]. Therefore, capped bigraphs, two-dimensional comparability bigraphs, interval containment bigraphs, and two-directional orthogonal ray graphs are one and the same class. The equivalence with two-directional

orthogonal ray graphs is immediate: Suppose that the vertical rays extend upwards and the horizontal rays extend to the left, and observe that a vertical ray v intersects a horizontal ray h if and only if the origin of v is dominated by the origin of h.

▶ **Theorem 15.** *Every capped graph on n vertices admits a biclique cover of size $O(n \log^3 n)$.*

Proof. Consider a capped graph $G = ([n], E)$ and let $L := [\lfloor n/2 \rfloor]$ and $R = [n] \setminus L$. Let $F = \{ij \in E : i \in L, j \in R\}$. By definition, $H := (L \cup R, F)$ is a capped bigraph. From Lemma 14 and Theorem 8, H admits a biclique cover of size $O(n \log^2 n)$. It remains to cover the edges of the two subgraphs $G[L]$ and $G[R]$, which are also capped graphs. This can be done recursively, yielding a total size of $S(n) \leq 2S(\lceil n/2 \rceil) + O(n \log^2 n) = O(n \log^3 n)$, as claimed. ◀

We now consider the computational problem of constructing a biclique cover of size $O(n \log^3 n)$ of a capped graph given as input. The natural encoding of the input graph can be of size proportional to the number of edges. Remember that capped graphs are ordered graphs. We can suppose that a capped graph is given in *sorted adjacency list* representation, in which not only the order of the vertices is given, but we are also given the neighbors of each vertex in sorted order. Note that sorting adjacency lists requires an additional cost of $O(|E| \log \log n)$, since the numbers to sort lie in the range $[n]$ [77].

▶ **Lemma 16.** *Given a capped graph $G = (V, E)$ in sorted adjacency list representation, we can compute a biclique partition of G of size $O(n \log^3 n)$ in time $O(\max\{|E|, n \log^3 n\})$.*

Proof. We follow the proof of Theorem 15. Given a bipartition of the vertices of G into the sets L and R, we need to efficiently compute the implicit representation of the corresponding comparability bigraph, as described in the proof of Lemma 14. This requires finding, for each vertex ℓ in L, the smallest index $r \in R$ such that $\ell r \in E$, and conversely for every vertex in R. This can be achieved by storing adjacency lists in binary search trees, for instance, which can be done in linear time if they are sorted. ◀

We can then for instance combine Lemma 16 with Lemma 4.

▶ **Theorem 17.** *Given a capped graph G on n vertices in sorted adjacency list representation, we can solve the all-pairs shortest path problem on G in time $O(n^2 \operatorname{polylog} n)$. Furthermore, after a $O(\max\{|E|, n \log^3 n\})$-time preprocessing, one can construct the breadth-first search tree rooted at any vertex in time $O(n \log^3 n)$.*

5 Lower bounds

It is known that there exist configurations of n points and n lines with $\Theta(n^{4/3})$ point-line incidences, hence that are tight examples for the Szemerédi-Trotter incidence bound; see for instance the construction from Erdős described in Edelsbrunner [38]. From Observation 2, we have that the size s of a biclique cover for a graph $G = (V, E)$ without $K_{t,t}$ for some constant t must satisfy $s \geq |E|/t$. Clearly, point-line incidence graphs do not have $K_{2,2}$ subgraphs, hence there exist such graphs on n vertices for which $s \geq \Omega(n^{4/3})$. We first prove a similar statement for incidence graphs of points and *halfplanes* in $\mathbb{R}^2$. The proof is essentially the same as that of Erickson [39, Theorem 3.4].

▶ **Lemma 18.** *There exist incidence graphs between n points and n closed lower halfplanes, any biclique cover of which has size $\Omega(n^{4/3})$.*

Proof. We consider a configuration (P, L) of n points and n lines with $\Theta(n^{4/3})$ point-line incidences. Let H be the set of closed lower halfplanes bounded by the lines of L. We denote by $I(P, H)$ the incidence graph between the points of P and the halfplanes of H.

Let $P_i \subseteq P$ and $H_i \subseteq H$ be the two sets of vertices in the ith biclique of a biclique cover of $I(P, H)$. Let $L_i \subseteq L$ be the lines bounding the halfspaces of H_i, and let us denote by $\iota(P_i, L_i)$ the number of pairs $(p, \ell) \in P_i \times L_i$ such that $p \in \ell$.

▷ Claim 19.

$$|P_i| + |H_i| \geq \iota(P_i, L_i).$$

Proof of claim. Let R_i denote the intersection of the lower halfplanes in H_i. By definition, R_i is a downward closed convex polygonal region, and $P_i \subset R_i$.

Consider the leftmost incidence (p, ℓ) in (P_i, L_i), involving the leftmost point with the leftmost line. Suppose that p is incident to more than one line. It must then be the case that ℓ does not contain any other point than p. Hence either p is incident to only one line, or ℓ contains only one point. We can therefore always remove either a point or a line and remove one incidence. The number of incidences is then at most $|P_i| + |L_i| = |P_i| + |H_i|$, as claimed. ◁

We now obtain a lower bound on the size of the biclique cover as follows:

$$\sum_i |P_i| + |H_i| \geq \sum_i \iota(P_i, L_i) \geq \iota(P, L) \geq \Omega(n^{4/3}).$$

where the second inequality is from the fact that every incidence in (P, L) is an edge of $I(P, H)$, hence must be covered by at least one of the biclique. ◀

We now turn our attention to unit disk graphs, which are perhaps among the simplest non-semilinear geometric intersection graphs.

▶ **Lemma 20.** *There exist unit disk graphs on n vertices, any biclique cover of which has size $\Omega(n^{4/3})$.*

Proof. It is enough to show that the incidence graph $I(P, H)$ in the proof of Lemma 18 can be realized with unit disks. Indeed, take the configuration (P, L) and shift every line of L upwards by a small vertical offset, so that the points involved in the incidences lie now slightly below their lines. We can now safely replace these lines by circles of the same very large radius, keeping the points of P in the same relative positions with respect to each of the lines. Let Q denote the set of centers of the circles. By a proper scaling, we can assume without loss of generality that those circles have radius two.

Observe that if a graph G has a biclique cover of size s, then any bipartite graph obtained from G by coloring the vertices in two colors and removing all monochromatic edges also has a biclique cover of size s. It is therefore sufficient to prove a lower bound on the size of a biclique cover of the bipartite intersection graph of the unit disks of centers respectively in P and Q. By construction, this is exactly the point-halfplane incidence graph $I(P, H)$, and we can now apply the result of Lemma 18. ◀

Note that a similar statement should hold for intersection graphs of translates of any smooth strictly convex body.

One may also wonder if there exists superlinear lower bounds on the size of biclique covers for families of semilinear graphs. We can easily deduce such a lower bound from a recent result from Bhore et al. [14].

▶ **Lemma 21.** *There exist graphs on n vertices and boxicity at most d, any biclique cover of which has size $\Omega(n(\log n/\log\log n)^{d-2})$.*

Proof. A construction from Bhore et al. [14] shows that there exist intersection graphs of n boxes in dimension d such that any 3-hop spanners requires $\Omega(n(\log n/\log\log n)^{d-2})$ edges. From Lemma 6, it implies the same lower bound on the size of their biclique covers. ◀

6 Open questions

Many open questions remain. We showed a few upper bounds on the size of the biclique cover for restricted families of semilinear graphs, such as intersection graphs of axis-aligned boxes and grounded L-shapes (see the full version of the paper for the definition and the bound on intersection graphs of grounded L-shapes [20]). These bounds improve upon those that can be derived from the general upper bound for semilinear graphs. A first natural question is whether any of those bounds can be improved or, alternatively, what are the achievable lower bounds on the size of a biclique cover for these graphs. The question of improving the upper bound also remains for capped graphs; do capped graphs admit $O(n\log^2 n)$-size biclique covers?

For both capped and non-jumping graphs we showed an upper bound of $O(n\log^3 n)$ on the size of their biclique cover. Recall that capped and non-jumping graphs are graphs equipped with an ordering on their vertices such that for any four vertices $i<j<k<\ell$, if ik and $j\ell$ are edges, then the edge $i\ell$ must also be present in a capped graph, and the edge jk must be present in a non-jumping graph. There is an interesting superclass of both capped and non-jumping graphs, in which given four vertices $i<j<k<\ell$ such that ik and $j\ell$ are edges, *either* the edge $i\ell$ *or* jk must be present. In particular, they contain the terrain visibility graphs considered by Katz, Saban, and Sharir [53], in which points lying strictly above the terrain are also allowed as vertices. Do these ordered graphs admit biclique covers of small size?

More generally, it would be interesting to characterize the forbidden patterns which give rise to families of (ordered) graphs for which the size of a biclique cover is $O(n\operatorname{poly}\log n)$. Note that there are forbidden patterns of size 3 for which the family of graphs which forbids them does not have such a bound. For example, 3 vertices which induce a triangle.

Finally, in this work we mostly focused on graph classes that are either arising in a geometric setting or forbid some ordered patterns. It would be interesting to understand how the size of a clique cover correlates with other width parameters of graphs, for example, the *twin-width* of a graph [15].

References

1 James Abello, Ömer Eğecioğlu, and Krishna Kumar. Visibility graphs of staircase polygons and the weak Bruhat order. I. From visibility graphs to maximal chains. *Discrete Comput. Geom.*, 14(3):331–358, 1995. doi:10.1007/BF02570710.

2 Eyal Ackerman and Balázs Keszegh. The Zarankiewicz Problem for Polygon Visibility Graphs. *arXiv preprint arXiv:2503.09115*, 2025. doi:10.48550/arXiv.2503.09115.

3 Pankaj K. Agarwal, Noga Alon, Boris Aronov, and Subash Suri. Can visibility graphs be represented compactly? *Discrete Comput. Geom.*, 12(3):347–365, 1994. doi:10.1007/BF02574385.

4 Pankaj K. Agarwal, Boris Aronov, Esther Ezra, Matthew J. Katz, and Micha Sharir. Intersection queries for flat semi-algebraic objects in three dimensions and related problems. In Xavier Goaoc and Michael Kerber, editors, *38th International Symposium on Computational Geometry, SoCG 2022, June 7-10, 2022, Berlin, Germany*, volume 224 of *LIPIcs*, pages 4:1–4:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SOCG.2022.4.

5 Pankaj K. Agarwal, Boris Aronov, Esther Ezra, Matthew J. Katz, and Micha Sharir. Intersection queries for flat semi-algebraic objects in three dimensions and related problems. *arXiv preprint 2203.10241*, 2025. doi:10.48550/arXiv.2203.10241.

6 Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. *Contemporary Mathematics*, 223(1):56, 1999. doi:10.1090/conm/223/03131.

7 Pankaj K. Agarwal, Esther Ezra, and Micha Sharir. Semi-algebraic off-line range searching and biclique partitions in the plane. In Wolfgang Mulzer and Jeff M. Phillips, editors, *40th International Symposium on Computational Geometry, SoCG 2024, June 11-14, 2024, Athens, Greece*, volume 293 of *LIPIcs*, pages 4:1–4:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SOCG.2024.4.

8 Pankaj K. Agarwal and Kasturi R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete Comput. Geom.*, 23(2):273–291, 2000. doi:10.1007/PL00009500.

9 Adeli A. Ahmadlou M., Adeli H. New diagnostic EEG markers of the Alzheimer's disease using visibility graph. *J. Neural Transm.*, 117:1099–1109, 2010. doi:10.1007/s00702-010-0450-3.

10 Noga Alon, János Pach, Rom Pinchasi, Radoš Radoičić, and Micha Sharir. Crossing patterns of semi-algebraic sets. *J. Combin. Theory Ser. A*, 111(2):310–326, 2005. doi:10.1016/j.jcta.2004.12.008.

11 Safwa Ameer, Matt Gibson-Lopez, Erik Krohn, Sean Soderman, and Qing Wang. Terrain visibility graphs: persistence is not enough. In *36th International Symposium on Computational Geometry*, volume 164 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 6, 13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.SOCG.2020.6.

12 Stav Ashur, Omrit Filtser, Matthew J. Katz, and Rachel Saban. Terrain-like graphs: PTASs for guarding weakly-visible polygons and terrains. *Comput. Geom.*, 101:Paper No. 101832, 13, 2022. doi:10.1016/j.comgeo.2021.101832.

13 Abdul Basit, Artem Chernikov, Sergei Starchenko, Terence Tao, and Chieu-Minh Tran. Zarankiewicz's problem for semilinear hypergraphs. In *Forum of Mathematics, Sigma*, volume 9, page e59. Cambridge University Press, 2021. doi:10.1017/fms.2021.52.

14 Sujoy Bhore, Timothy M. Chan, Zhengcheng Huang, Shakhar Smorodinsky, and Csaba D. Tóth. Sparse bounded hop-spanners for geometric intersection graphs. *arXiv preprint arXiv:2504.05861*, 2025. doi:10.48550/arXiv.2504.05861.

15 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable fo model checking. *ACM Journal of the ACM (JACM)*, 69(1):1–46, 2021. doi:10.1145/3486655.

16 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999. doi:10.1137/1.9780898719796.

17 Peter Braß and Christian Knauer. On counting point-hyperplane incidences. *Computational Geometry*, 25(1-2):13–20, 2003. doi:10.1016/S0925-7721(02)00127-X.

18 Sergio Cabello, Siu-Wing Cheng, Otfried Cheong, and Christian Knauer. Geometric matching and bottleneck problems. In *40th International Symposium on Computational Geometry, SoCG 2024, June 11-14, 2024, Athens, Greece*, pages 31:1–31:15, 2024. doi:10.4230/LIPICS.SOCG.2024.31.

19 Jean Cardinal and Micha Sharir. Improved algebraic degeneracy testing. *Discrete & Computational Geometry*, pages 1–19, 2024. doi:10.1007/s00454-024-00673-7.

20 Jean Cardinal and Yelena Yuditsky. Compact representation of semilinear and terrain-like graphs. *arXiv preprint arXiv:2507.00252*, 2025. doi:10.48550/arXiv.2507.00252.

21 Timothy M. Chan. Dynamic subgraph connectivity with geometric applications. *SIAM Journal on Computing*, 36(3):681–694, 2006. doi:10.1137/S009753970343912X.

22 Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ Time. *Algorithmica*, 50(2):236–243, 2008. doi:10.1007/S00453-007-9062-1.

23 Timothy M. Chan. Optimal partition trees. *Discrete Comput. Geom.*, 47(4):661–690, 2012. doi:10.1007/s00454-012-9410-z.

24 Timothy M Chan, Pingan Cheng, and Da Wei Zheng. Semialgebraic range stabbing, ray shooting, and intersection counting in the plane. In *40th International Symposium on Computational Geometry, SoCG 2024*, page 33. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, 2024. doi:10.4230/LIPIcs.SoCG.2024.33.

25 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in geometric intersection graphs. *J. Comput. Geom.*, 10(1):27–41, 2019. doi:10.20382/JOCG.V10I1A2.

26 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993. doi:10.1007/BF02189314.

27 Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. Algorithms for bichromatic line-segment problems polyhedral terrains. *Algorithmica*, 11(2):116–132, 1994. doi:10.1007/BF01182771.

28 Bernard Chazelle and Leonidas J. Guibas. Visibility and intersection problems in plane geometry. *Discret. Comput. Geom.*, 4:551–581, 1989. doi:10.1007/BF02187747.

29 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 612–623. IEEE, 2022. doi:10.1109/FOCS54457.2022.00064.

30 Fan R. K. Chung, Paul Erdős, and Joel Spencer. On the decomposition of graphs into complete bipartite subgraphs. In *Studies in pure mathematics*, pages 95–101. Birkhäuser, Basel, 1983. doi:10.1007/978-3-0348-5438-2_10.

31 David Conlon. Some remarks on the Zarankiewicz problem. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 173(1), pages 155–161. Cambridge University Press, 2022. doi:10.1017/S0305004121000475.

32 David Conlon, Jacob Fox, János Pach, Benny Sudakov, and Andrew Suk. Ramsey-type results for semi-algebraic relations. *Trans. Amer. Math. Soc.*, 366(9):5043–5065, 2014. doi:10.1090/S0002-9947-2014-06179-5.

33 Jonathan B. Conroy and Csaba D. Tóth. Hop-spanners for geometric intersection graphs. *J. Comput. Geom.*, 14(2):26–64, 2022. doi:10.20382/JOCG.V14I2A3.

34 James Davies, Tomasz Krawczyk, Rose McCarty, and Bartosz Walczak. Coloring polygon visibility graphs and their generalizations. *J. Combin. Theory Ser. B*, 161:268–300, 2023. doi:10.1016/j.jctb.2023.02.010.

35 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

36 Sarita de Berg, Nathan van Beusekom, Max van Mulken, Kevin Verbeek, and Jules Wulms. Competitive searching over terrains. In José A. Soto and Andreas Wiese, editors, *LATIN 2024: Theoretical Informatics - 16th Latin American Symposium, Puerto Varas, Chile, March 18-22, 2024, Proceedings, Part I*, volume 14578 of *Lecture Notes in Computer Science*, pages 254–269. Springer, 2024. doi:10.1007/978-3-031-55598-5_17.

37 Thao Do. Representation complexities of semialgebraic graphs. *SIAM J. Discrete Math.*, 33(4):1864–1877, 2019. doi:10.1137/18M1221606.

38 Herbert Edelsbrunner. *Algorithms in combinatorial geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1987. doi:10.1007/978-3-642-61568-9.

39 Jeff Erickson. New lower bounds for Hopcroft's problem. *Discrete & Computational Geometry*, 16(4):389–418, 1996. doi:10.1007/BF02712875.

40 William Evans and Noushin Saeedi. On characterizing terrain visibility graphs. *J. Comput. Geom.*, 6(1):108–141, 2015. doi:10.20382/jocg.v6i1a5.

41 Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. System Sci.*, 51(2):261–272, 1995. doi:10.1006/jcss.1995.1065.

42 Jacob Fox, János Pach, Adam Sheffer, Andrew Suk, and Joshua Zahl. A semi-algebraic version of Zarankiewicz's problem. *J. Eur. Math. Soc. (JEMS)*, 19(6):1785–1810, 2017. doi:10.4171/JEMS/705.

43 Vincent Froese and Malte Renken. A fast shortest path algorithm on terrain-like graphs. *Discrete Comput. Geom.*, 66(2):737–750, 2021. doi:10.1007/s00454-020-00226-8.

44 Vincent Froese and Malte Renken. Persistent graphs and cyclic polytope triangulations. *Comb.*, 41(3):407–423, 2021. doi:10.1007/S00493-020-4369-5.

45 Vincent Froese and Malte Renken. Terrain-like graphs and the median Genocchi numbers. *European J. Combin.*, 115:Paper No. 103780, 8, 2024. doi:10.1016/j.ejc.2023.103780.

46 Subir K. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, 2007. doi:10.1017/CBO9780511543340.

47 Oktay Günlük. A new min-cut max-flow ratio for multicommodity flows. *SIAM Journal on Discrete Mathematics*, 21(1):1–15, 2007. doi:10.1137/S089548010138917X.

48 Larry Guth and Nets Hawk Katz. On the Erdős distinct distances problem in the plane. *Ann. of Math. (2)*, 181(1):155–190, 2015. doi:10.4007/annals.2015.181.1.2.

49 Pavol Hell, Jing Huang, Jephian C.-H. Lin, and Ross M. McConnell. Bipartite analogues of comparability and cocomparability graphs. *SIAM J. Discrete Math.*, 34(3):1969–1983, 2020. doi:10.1137/19M1263789.

50 Jing Huang. Representation characterizations of chordal bipartite graphs. *J. Combin. Theory Ser. B*, 96(5):673–683, 2006. doi:10.1016/j.jctb.2006.01.001.

51 Stasys Jukna and Alexander S Kulikov. On covering graphs by complete bipartite subgraphs. *Discrete Mathematics*, 309(10):3399–3403, 2009. doi:10.1016/J.DISC.2008.09.036.

52 Prahlad Narasimhan Kasthurirangan. One-sided terrain guarding and chordal graphs. *Discrete Appl. Math.*, 348:192–201, 2024. doi:10.1016/j.dam.2024.01.034.

53 Matthew J. Katz, Rachel Saban, and Micha Sharir. Near-linear algorithms for visibility graphs over a 1.5-dimensional terrain. In Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms (ESA 2024)*, volume 308 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 77:1–77:17, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2024.77.

54 Matthew J. Katz and Micha Sharir. An expander-based approach to geometric optimization. *SIAM J. Comput.*, 26(5):1384–1408, 1997. doi:10.1137/S0097539794268649.

55 Oliver Kullmann and Ankit Shukla. Transforming quantified boolean formulas using biclique covers. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 372–390. Springer, 2023. doi:10.1007/978-3-031-30820-8_23.

56 Lucas Lacasa, Bartolo Luque, Fernando Ballesteros, Jordi Luque, and Juan Carlos Nuño. From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*, 105(13):4972–4975, 2008. doi:10.1073/pnas.0709247105.

57 Jiří Matoušek. Range searching with efficient hierarchical cuttings. In *Proceedings of the eighth annual symposium on Computational geometry*, pages 276–285, 1992. doi:10.1145/142675.142732.

58 Jiří Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.*, 6(5):385–406, 1991. doi:10.1007/BF02574697.

59 Jiří Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8(3):315–334, 1992. ACM Symposium on Computational Geometry (North Conway, NH, 1991). doi:10.1007/BF02293051.

60 Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10(2):157–182, 1993. doi:10.1007/BF02573972.

61 Jiří Matoušek and Zuzana Patáková. Multilevel polynomial partitions and simplified range searching. *Discrete Comput. Geom.*, 54(1):22–41, 2015. doi:10.1007/s00454-015-9701-2.

62 Joseph O'Rourke. Visibility. In *Handbook of Discrete and Computational Geometry, 3rd Edition*, chapter 33. Chapman and Hall/CRC, 2017. doi:10.1201/9781420035315.CH28.

63 János Pach and Micha Sharir. Incidences. In *Graph theory, combinatorics and algorithms*, pages 267–292. Springer, New York, 2005. doi:10.1007/0-387-25036-0_11.

64 Larry Palazzi and Jack Snoeyink. Counting and reporting red/blue segment intersections. *CVGIP: Graphical Models and Image Processing*, 56(4):304–310, 1994. doi:10.1006/cgip.1994.1027.

65 Franco P. Preparata and Michael Ian Shamos. *Computational geometry.* Texts and Monographs in Computer Science. Springer-Verlag, New York, 1985. doi:10.1007/978-1-4612-1098-6.

66 Vojtech Rödl and Andrzej Ruciński. Bipartite coverings of graphs. *Combinatorics, Probability and Computing*, 6(3):349–352, 1997. doi:10.1017/S0963548397003064.

67 Lisa Sauermann. On the speed of algebraically defined graph classes. *Advances in Mathematics*, 380:107593, 2021. doi:10.1016/j.aim.2021.107593.

68 Stephan Schwartz. An overview of graph covering and partitioning. *Discrete Math.*, 345(8):Paper No. 112884, 17, 2022. doi:10.1016/j.disc.2022.112884.

69 Anish Man Singh Shrestha, Satoshi Tayu, and Shuichi Ueno. On orthogonal ray graphs. *Discrete Appl. Math.*, 158(15):1650–1659, 2010. doi:10.1016/j.dam.2010.06.002.

70 Shakhar Smorodinsky. A survey of Zarankiewicz problems in geometry. *arXiv preprint arXiv:2410.03702*, 2024. doi:10.48550/arXiv.2410.03702.

71 Andrew Suk. Semi-algebraic Ramsey numbers. *J. Combin. Theory Ser. B*, 116:465–483, 2016. doi:10.1016/j.jctb.2015.10.001.

72 Xiaoying Tang, Li Xia, Yezi Liao, Weifeng Liu, Yuhua Peng, Tianxin Gao, and Yanjun Zeng. New approach to epileptic diagnosis using visibility graph of high-frequency signal. *Clinical EEG and Neuroscience*, 44(2):150–156, 2013. PMID: 23508995. doi:10.1177/1550059412464449.

73 István Tomon. Ramsey properties of semilinear graphs. *Israel Journal of Mathematics*, 254(1):113–139, 2023. doi:10.1007/s11856-022-2390-7.

74 Csaba D. Tóth. Weighted biclique covers. Open problem posed at the Tenth Annual Workshop on Geometry and Graphs (WoGaG'23), Bellairs Research Institute, 2023.

75 Zsolt Tuza. Covering of graphs by complete bipartite subgraphs: complexity of 0-1 matrices. *Combinatorica*, 4(1):111–116, 1984. doi:10.1007/BF02579163.

76 Jan van den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P. Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 503–514. IEEE, 2023. doi:10.1109/FOCS57990.2023.00037.

77 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. In *16th Annual Symposium on Foundations of Computer Science (Univ. California, Berkeley, Calif., 1975)*, pages 75–84. IEEE, Long Beach, CA, 1975. doi:10.1109/SFCS.1975.26.

Faster Algorithm for Second (s,t)-Mincut and Breaking Quadratic Barrier for Dual Edge Sensitivity for (s,t)-Mincut

Surender Baswana
Department of Computer Science & Engineering, IIT Kanpur, India

Koustav Bhanja
Weizmann Institute of Science, Rehovot, Israel

Anupam Roy
Department of Computer Science & Engineering, IIT Kanpur, India

Abstract

Let G be a directed graph on n vertices and m edges. In this article, we study (s,t)-cuts of second minimum capacity and present the following algorithmic and graph-theoretic results.

1. **Second (s,t)-mincut:** Vazirani and Yannakakis [ICALP 1992] designed the first algorithm for computing an (s,t)-cut of second minimum capacity using $\mathcal{O}(n^2)$ maximum (s,t)-flow computations. We present the following algorithm that improves the running time significantly. For directed integer-weighted graphs, there is an algorithm that can compute an (s,t)-cut of second minimum capacity using $\tilde{\mathcal{O}}(\sqrt{n})$ maximum (s,t)-flow computations with high probability.[1] To achieve this result, a close relationship of independent interest is established between (s,t)-cuts of second minimum capacity and global mincuts in directed weighted graphs.
2. **Minimum+1 (s,t)-cuts:** Minimum+1 (s,t)-cuts have been studied quite well recently [Baswana, Bhanja, and Pandey, ICALP 2022 & TALG 2023], which is a special case of second (s,t)-mincut. We present the following structural result and the first nontrivial algorithm for minimum+1 (s,t)-cuts.
   a. Algorithm: For directed multi-graphs, we design an algorithm that, given any maximum (s,t)-flow, computes a minimum+1 (s,t)-cut, if it exists, in $\mathcal{O}(m)$ time.
   b. Structure: The existing structures for storing and characterizing all minimum+1 (s,t)-cuts occupy $\mathcal{O}(mn)$ space [Baswana, Bhanja, and Pandey, TALG 2023]. For undirected multi-graphs, we design a directed acyclic graph (DAG) occupying only $\mathcal{O}(m)$ space that stores and characterizes all minimum+1 (s,t)-cuts. This matches the space bound of the widely-known DAG structure for all (s,t)-mincuts [Picard and Queyranne, Math. Prog. Studies 1980].
3. **Dual Edge Sensitivity Oracle:** The study of minimum+1 (s,t)-cuts often turns out to be useful in designing dual edge sensitivity oracles – a compact data structure for efficiently reporting an (s,t)-mincut after insertion/failure of any given pair of query edges. It has been shown recently [Bhanja, ICALP 2025] that any dual edge sensitivity oracle for (s,t)-mincut in undirected multi-graphs must occupy $\Omega(n^2)$ space in the worst-case irrespective of the query time. Interestingly, for undirected unweighted simple graphs, we break this quadratic barrier while achieving a non-trivial query time as follows. There is an $\mathcal{O}(n\sqrt{n})$ space data structure that can report an (s,t)-mincut in $\mathcal{O}(\min\{m, n\sqrt{n}\})$ time after the insertion/failure of any given pair of query edges.

To arrive at our results, as one of our key techniques, we establish interesting relationships between (s,t)-cuts of capacity (minimum+Δ), $\Delta \geq 0$, and maximum (s,t)-flow. We believe that these techniques and the graph-theoretic result in 2.(b) are of independent interest.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Dynamic graph algorithms; Theory of computation → Network flows

[1] $\tilde{\mathcal{O}}(.)$ hides poly-logarithmic factors.


© Surender Baswana, Koustav Bhanja, and Anupam Roy;
licensed under Creative Commons License CC-BY 4.0
33rd Annual European Symposium on Algorithms (ESA 2025).
Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman; Article No. 68; pp. 68:1–68:19

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

Keywords and phrases mincut, second mincut, compact structure, fault tolerant, sensitivity oracle, dual edges, st mincut, global mincut, characterization

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.68

Related Version *Full Version*: https://arxiv.org/abs/2507.01366

Funding The research work of Surender Baswana is partially supported by Tapas Mishra Memorial Chair at IIT Kanpur, India. The research work of Koustav Bhanja is partially supported by Merav Parter's European Research Council (ERC) grant under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 949083.

Acknowledgements This research work was partially carried out while Koustav Bhanja was pursuing his doctoral studies at the Department of CSE, IIT Kanpur, India.

1 Introduction

The concept of cut is fundamental in graph theory and has numerous real-world applications [1]. Let $G = (V, E)$ be a directed weighted graph on $n = |V|$ vertices and $m = |E|$ edges with a designated source vertex s and a designated sink vertex t. Every edge e of G is assigned with a non-negative real value as the capacity of the edge, denoted by $w(e)$. There are mainly two types of well-studied cuts in a graph – global cuts and (s,t)-cuts. A *global cut*, or simply a *cut*, of G is defined as a nonempty proper subset of V. A cut C is said to be an (s,t)-*cut* if $s \in C$ and $t \in \overline{C}(= V \setminus C)$. Every cut of G is associated with a capacity defined as follows. The *capacity* of a cut C, denoted by $c(C)$, is the sum of the capacities of every edge (x, y) satisfying $x \in C$ and $y \in \overline{C}$. A global cut (likewise (s,t)-cut) of the least capacity is called a *global mincut* (likewise (s,t)-*mincut*). Henceforth λ denotes the capacity of (s,t)-mincut.

The study of cuts from both structural and algorithmic perspectives has been an important field of research for the past six decades [21, 19, 14, 25, 36]. In this article, we provide the following two main results for (s,t)-cuts. (1) We present efficient algorithms for computing an (s,t)-cut of second minimum capacity in directed weighted graphs. After more than 30 years, our algorithms provide the first improvement by a polynomial factor over the existing result of Vazirani and Yannakakis [37]. To arrive at this result, we establish that computing an (s,t)-cut of second minimum capacity has the same time complexity as computing a global mincut. (2) We present a dual edge sensitivity oracle for (s,t)-mincut – a compact data structure that efficiently reports an (s,t)-mincut after the insertion or failure of any pair of edges. This is the first dual edge sensitivity oracle for (s,t)-mincut that occupies subquadratic space while achieving nontrivial query time in undirected unweighted simple graphs[1]. This breaks the existing quadratic lower bounds [3, 6] on the space for any dual edge sensitivity oracle for (s,t)-mincut in undirected multi-graphs[2]. We arrive at this result by designing a compact structure for storing and characterizing all *minimum+1* (s,t)-*cuts* (a special case of second minimum (s,t)-cut). This structure improves the space occupied by the existing best-known structure [3] by a linear factor.

We now present the problems addressed in this article, their state-of-the-art, and our results.

[1] A simple graph refers to a graph having no parallel edges.
[2] A multi-graph refers to an unweighted graph having parallel edges.

1.1 Faster Algorithm for Second Minimum (s,t)-cuts

An algorithmic graph problem aims at computing a structure that optimizes a given function. Examples of such classical problems are Minimum Spanning Tree, Shortest Path, Minimum Cut. Having designed (near) optimal algorithm for such problems, the next immediate objective is the following. Design an efficient algorithm that, given a structure S achieving optimal function value, computes a structure S' that differs from S and achieves the optimal or *next* optimal function value. There has been an extensive study on computing the second minimum spanning tree [8, 26, 29], the second shortest path [38, 18, 34], and second (s,t)-mincut [37]. The algorithms for these problems are so fundamental that they appear in textbooks on algorithms [13, 1]. In addition, they act as the foundation for generalizing the problem to compute k^{th} optimal structure, such as computing k^{th} minimum spanning tree, k^{th} shortest path, k^{th} (s,t)-mincut.

The problem of designing efficient algorithms for computing (s,t)-mincut (or equivalently, maximum (s,t)-flow) has been studied for more than six decades. This problem is now almost settled with the recent breakthrough result on almost linear time algorithm for maximum (s,t)-flow by Van et al. [36]. Interestingly, given an algorithm for maximum (s,t)-flow, we can compute all (s,t)-mincuts implicitly with an additional $\mathcal{O}(m)$ time, as shown by Picard and Queyranne [33]. It is, therefore, natural to redefine the second (s,t)-mincut problem as follows. Design an algorithm that computes an (s,t)-cut of second minimum capacity. For brevity, henceforth we call (s,t)-cut of second minimum capacity by second (s,t)-mincut.

Vazirani and Yannakakis [37] addressed the problem of computing an (s,t)-cut having k^{th} minimum capacity in 1992. For computing any k^{th} minimum capacity (s,t)-cut, they gave an algorithm that uses $\mathcal{O}(n^{2(k-1)})$ maximum (s,t)-flow computations. Conversely, they also argued, using NP-hardness of computing a maximum cut, that exponential dependence on k is unavoidable assuming $P \neq NP$. To arrive at their result, the key problem they addressed is the design of an algorithm that computes a second (s,t)-mincut using $\mathcal{O}(n^2)$ maximum (s,t)-flow computations. They further improve the running time by designing a faster algorithm that computes a second (s,t)-mincut using only $\mathcal{O}(n)$ maximum (s,t)-flow computations. We show that this faster algorithm is incorrect due to a serious error in its analysis. As a result, the existing best-known algorithm for computing a second (s,t)-mincut uses $\mathcal{O}(n^2)$ maximum (s,t)-flow computations [37]. We design an algorithm for the second (s,t)-mincut with a significantly improved running time as shown in the following theorem.

▶ **Theorem 1** (Second (s,t)-mincut algorithm). *For any directed graph G on n vertices with integer edge capacities, there is an algorithm that computes a second (s,t)-mincut in G using $\tilde{\mathcal{O}}(\sqrt{n})$ maximum (s,t)-flow computations with high probability.*

The two well-known minimum cuts in a graph are (s,t)-mincut and global mincut. Recently, there has been a growing interest in understanding the difference in the complexity of computing an (s,t)-mincut and computing a global mincut. For undirected weighted graphs, Li and Panigrahi [28] established that the running time of computing a global mincut differs by only poly-logarithmic factors from the running time of computing an (s,t)-mincut. In particular, the authors showed that there is an algorithm that can compute a global mincut using $\tilde{\mathcal{O}}(1)$ maximum (s,t)-flow computations with additional $\tilde{\mathcal{O}}(m)$ time. However, for directed weighted graphs, there is a *large* gap between the running time of computing an (s,t)-mincut and computing a global mincut as follows. Cen et al. [9] designed an algorithm that can compute a global mincut using $\tilde{\mathcal{O}}(\sqrt{n})$ maximum (s,t)-flow computations with additional $\tilde{\mathcal{O}}(m\sqrt{n})$ time. Interestingly, to arrive at our result in Theorem 1, we show that the complexity of computing a global mincut is the same as the complexity of computing a second (s,t)-mincut *modulo* a single maximum (s,t)-flow computation as follows, which might find many other important applications.

▶ **Theorem 2** (Equivalence between global mincut and second (s,t)-mincut)**.** *For any directed weighted graph G on n vertices and m edges, the following two assertions hold.*

1. *The problem of computing a second (s,t)-mincut is reducible to the problem of computing a global mincut in $\mathcal{O}(MF(m,n))$ time, where $MF(m,n)$ denotes the time complexity for computing a maximum (s,t)-flow in G.*
2. *The problem of computing a global mincut is reducible to the problem of computing a second (s,t)-mincut in $\mathcal{O}(m)$ time.*

▶ Remark 3. Our algorithm for computing a second (s,t)-mincut in Theorem 1 is Monte Carlo and works for graphs with integer edge capacities. However, the equivalence between second (s,t)-mincut and global mincut in Theorem 2 is deterministic and holds even for graphs with real edge capacities.

1.2 Minimum+1 (s,t)-cuts: Efficient Algorithm & Compact Structure

The cuts of capacity minimum+1, known as minimum+1 cuts, have been studied quite extensively in the past [3, 15, 6]. For (un)directed multi-graphs, a minimum+1 (s,t)-cut is a special case of second (s,t)-mincut. We present the following structural result and the first nontrivial algorithm for minimum+1 (s,t)-cuts.

Algorithm. For both minimum+1 global cuts [15] and minimum+1 (s,t)-cuts [3], there exist compact data structures. These data structures have important applications in maintaining minimum+2 edge connected components [15] and designing dual edge sensitivity oracle for (s,t)-mincut [3]. Moreover, there exist efficient algorithms [24, 4, 30] that can compute a global cut of capacity minimum+1. Unfortunately, the existing best-known algorithm for computing an (s,t)-cut of capacity minimum+1 is nothing but the algorithm for computing a second (s,t)-mincut by [37], which uses $\mathcal{O}(n^2)$ maximum (s,t)-flow computations. We present the following result as the first efficient algorithm for computing an (s,t)-cut of capacity minimum+1.

▶ **Theorem 4** (Minimum+1 (s,t)-cut algorithm)**.** *For any directed multi-graph G on n vertices and m edges, there is an algorithm that, given any maximum (s,t)-flow, computes a minimum+1 (s,t)-cut, if exists in G, in $\mathcal{O}(m)$ time.*

▶ Remark 5. The best-known algorithm for computing an (s,t)-mincut uses one maximum (s,t)-flow computation. It follows from Theorem 4 that the running time of computing a minimum+1 (s,t)-cut differs from the running time of computing an (s,t)-mincut only by additional $\mathcal{O}(m)$ time.

Structure. There are several algorithmic applications on cuts, namely, fault-tolerance [33, 14], dynamic algorithms [22, 23], edge-connectivity augmentation [31, 10] that require an efficient way to distinguish a set of cuts, say minimum cuts or minimum+1 cuts, from the rest of the cuts. A trivial way to accomplish this objective is to store each cut of the required set explicitly. However, this is totally impractical since the set of these cuts is usually quite huge. For example, the number of (s,t)-mincuts are exponential [33], and the number of global mincuts are $\Omega(n^2)$ [14]. This has led the researchers to invent the following concept. A structure H is said to *characterize* a set of cuts $\mathcal{C}$ using a property P if the following condition holds. A cut $C \in \mathcal{C}$ if and only if C satisfies property P in H. It is desirable that H is as compact as possible. Moreover, verifying if a given cut C satisfies P in H has to be as time-efficient as possible.

The design of compact structures for characterizing minimum cuts started with the seminal work of Dinitz, Karzanov, and Lomonosov [14] in 1976. In this work, the well-known cactus graph occupying $\mathcal{O}(n)$ space was invented for storing and characterizing all global mincuts. Several compact structures have been designed subsequently for storing and characterizing cuts of capacity both minimum and near minimum [33, 4, 15, 16, 3]. Quite expectedly, they are playing crucial roles in establishing many important algorithmic results [37, 4, 15, 23, 27, 3]. In particular, compact structures for storing and characterizing all minimum+1 cuts of a graph have been well-studied. For all minimum+1 global cuts in undirected multi-graphs, Dinitz and Nutov [15] constructed an $\mathcal{O}(n)$ space 2-level cactus model that stores and characterizes them. For all minimum+1 (s,t)-cuts in directed multi-graphs, the existing structure that provides a characterization occupies $\mathcal{O}(mn)$ space [3]. Unfortunately, the space occupied by this structure of [3] is significantly inferior to the following widely-known structure designed by Picard and Queyranne [33] for (s,t)-mincut. There is an $\mathcal{O}(m)$ space directed acyclic graph (DAG) that stores and characterizes all (s,t)-mincuts of any graph using 1-*transversal* cuts of this DAG. An (s,t)-cut is said to be 1-transversal if its edge-set[3] intersects any path at most once. Interestingly, we are able to achieve the $\mathcal{O}(m)$ bound on space for storing and characterizing all minimum+1 (s,t)-cuts.

An edge (u,v) is said to *contribute* to a cut C if $u \in C$ and $v \in \overline{C}$. We first establish that for any maximum (s,t)-flow f, there exists a set containing at most $n-2$ edges, called the *anchor* edges, such that for any minimum+1 (s,t)-cut C, exactly one anchor edge contributes to C. By exploiting this result, we design the following structure for storing and characterizing all minimum+1 (s,t)-cuts.

▶ **Theorem 6** (Structure for minimum+1 (s,t)-cuts). *Let G be an undirected multi-graph on n vertices and m edges with a maximum (s,t)-flow f. There is an $\mathcal{O}(\min\{m, n\sqrt{\lambda}\})$ space structure, consisting of a directed acyclic graph $\mathcal{D}$ and the set of $n-2$ anchor edges, that stores and characterizes all (s,t)-mincuts and all minimum+1 (s,t)-cuts of G as follows.*

1. *An (s,t)-cut C is an (s,t)-mincut in G if and only if C is a 1-transversal cut in $\mathcal{D}$ and no anchor edge corresponding to f contributes to C.*
2. *An (s,t)-cut C is a minimum+1 (s,t)-cut in G if and only if C is a 1-transversal cut in $\mathcal{D}$ and exactly one anchor edge corresponding to f contributes.*

For undirected graphs, the best-known structure for storing and characterizing all (s,t)-mincuts is the DAG of Picard and Queyranne [33] that occupies $\mathcal{O}(\min\{m, n\sqrt{\lambda}\})$ space, which is tight as well (refer to [20] and Lemma 12 in [22]). Interestingly, not only our structure in Theorem 6 matches the bound on space with the DAG for (s,t)-mincuts [33] but also it stores and characterizes both (s,t)-mincuts and minimum+1 (s,t)-cuts.

1.3 Dual Edge Sensitivity Oracle: Breaking the Quadratic Barrier

The study of minimum+1 (s,t)-cuts often turns out to be useful in designing elegant *dual edge sensitivity oracles* [3, 15, 6], which is defined as follows.

▶ **Definition 7** (Dual edge sensitivity oracle). *A dual edge sensitivity oracle for minimum cuts is a compact data structure that can efficiently report a minimum cut in the graph after the insertion or failure of any given pair of query edges.*

Designing sensitivity oracles for various minimum cuts of a graph has been an emerging field of research [33, 15, 16, 3, 2]. For (s,t)-mincut in multi-graphs, the DAG structure of Picard and Queyranne [33], as shown in [3], can be used to design an $\mathcal{O}(n)$ space sensitivity oracle

[3] The *edge-set* of a cut C is the set of edges with exactly one endpoint in C.

that can report an (s,t)-mincut in $\mathcal{O}(n)$ time after the failure/insertion of any single edge. It is now interesting to design a sensitivity oracle that can handle multiple failures/insertions of edges. To solve this generic problem, as argued in [3], the natural approach is to first design a dual edge sensitivity oracle for (s,t)-mincut. This approach has also been taken for various other classical graph problems, including distance and connectivity [17], traversals [32], reachability [12, 11]. Moreover, it has been observed that handling two edge failures/insertions is significantly more nontrivial than handling a single one. It also provides important insights that either expose the hardness or help in generalizing the problem for multiple failures. To extend the result of [33] from single to multiple edge failures/insertions, Baswana, Bhanja, and Pandey [3] designed the first dual edge sensitivity oracle for (s,t)-mincut occupying $\mathcal{O}(n^2)$ space in directed multi-graphs. It can report a resulting (s,t)-mincut in $\mathcal{O}(n)$ time.

Note that quadratic space data structures often pose practical challenges as n can be quite *large* for the real world networks/graphs. It thus raises the need to design sensitivity oracles for various fundamental graph problems that occupy subquadratic space [7, 35, 5]. Unfortunately, it has been shown [3, 6] that any dual edge sensitivity oracle for (s,t)-mincut in undirected multi-graphs must occupy $\Omega(n^2)$ bits of space in the worst case, irrespective of the query time. However, for undirected unweighted simple graphs, we break this quadratic barrier on the space of any dual edge sensitivity oracle while achieving nontrivial query time as follows.

▶ **Theorem 8** (Dual edge sensitivity oracle for (s,t)-mincut)**.** *Let G be an undirected unweighted simple graph on n vertices and m edges. There exists a data structure occupying $\mathcal{O}(\min\{m, n^{1.5}\})$ space that can report an (s,t)-mincut C (including the contributing edges of C) in $\mathcal{O}(\min\{m, n^{1.5}\})$ time after failure/insertion of any given pair of query edges in G.*

For the existing dual edge sensitivity oracles for (s,t)-mincut [3, 6], no nontrivial preprocessing time is known till date. We establish the following almost linear preprocessing time for our dual edge sensitivity oracle stated in Theorem 8.

▶ **Theorem 9** (Preprocessing time)**.** *For undirected unweighted simple graphs on n vertices and m edges, there is an algorithm that, given any maximum (s,t)-flow, can construct the dual edge sensitivity oracle in Theorem 8 in $\mathcal{O}(m)$ time.*

2 Organization of this article

This article is organized as follows. Basic preliminaries and notations are stated in Section 3. A limitation of an existing algorithm in [37] for computing second (s,t)-mincut is provided in Section 4. In Section 5, we design two algorithms for computing a second (s,t)-mincut in directed weighted graphs. For undirected multi-graphs, in Section 6, we establish close relationships between maximum (s,t)-flow and (s,t)-cuts of capacity beyond (s,t)-mincut, which are used as tools to arrive at the results in the following sections. For undirected multi-graphs, Section 7 contains the design of our linear space structure for storing and characterizing all $(\lambda+1)$ (s,t)-cuts. In Section 8, we design the subquadratic space dual edge sensitivity oracle for (s,t)-mincut in undirected unweighted simple graphs. Finally, we conclude in Section 9. All the omitted proofs are provided in the full version.

3 Preliminaries

By integrality of maximum (s,t)-flow [19], for integer-weighted graphs, we consider any given maximum (s,t)-flow to be integral. The following notations to be used throughout the article.

- $G \cup A$ (likewise $G \setminus A$): Graph obtained after adding (likewise removing) a set of edges A in G.
- $(\lambda + \Delta)$ (s,t)-cut: An (s,t)-cut of capacity $\lambda + \Delta$ where $\Delta \geq 0$.
- (u,v)-path : A simple directed path from vertex u to vertex v.
- f denotes a maximum (s,t)-flow in graph G.
- $c(C,H)$: Capacity of a cut C in a graph H.
- A cut C *subdivides* a set of vertices X if $C \cap X \neq \emptyset$ and $\overline{C} \cap X \neq \emptyset$.
- A cut C *separates* a pair of vertices u, v if $u \in C$ and $v \in \overline{C}$ or vice-versa.
- The *edge-set* of a cut C, denoted by $E(C)$, is the set of all edges whose endpoints are separated by C.

Let f' be any (s,t)-flow in G.

- $H^{f'}$ denotes the residual graph for any graph H corresponding to an (s,t)-flow f'.
- $f'(e)$ denotes the value of flow f' assigned to an edge e.
- $f'_{out}(C)$ and $f'_{in}(C)$: For any (s,t)-cut C, $f'_{out}(C)$ (likewise $f'_{in}(C)$) is the sum of flow assigned to all edges $e = (u,v) \in E(C)$ with $u \in C, v \in \overline{C}$ (likewise $v \in C, u \in \overline{C}$).

▶ **Lemma 10** (Conservation of flow). *For any (s,t)-cut C, $f'_{out}(C) - f'_{in}(C) = value(f')$*

▶ **Definition 11** (Quotient Graph). *A graph H is said to be a quotient graph of G if H is obtained from G by contracting disjoint subsets of vertices into single nodes.*

▶ **Definition 12** (Quotient Path). *A path P_q is said to be a quotient path of a path P if P_q is obtained from P by contracting a set of edges in P.*

Residual graph for undirected multi-graphs. Although the residual graph is defined for directed graphs [19], for undirected graphs, we define the residual graph in the following way. Let $e = (x,y)$ be any edge in an undirected multi-graph H with an (s,t)-flow f'. There exist two edges (y,x) (likewise (x,y)) in $H^{f'}$ if e carries flow in the direction x to y (likewise y to x); otherwise, there is a pair of edges (x,y) and (y,x) in $H^{f'}$.

3.1 A DAG structure for storing and characterizing all (s,t)-mincuts

In a seminal work, Picard and Queyranne [33] designed a DAG, denoted by $\mathcal{D}_{PQ}(G)$, that stores all (s,t)-mincuts in G and characterizes them as 1-transversal cuts. We now briefly describe the construction of $\mathcal{D}_{PQ}(G)$.

Construction of $\mathcal{D}_{PQ}(G)$. Let G' be the graph obtained by contracting each Strongly Connected Component (SCC) of G^f into a single node. Let $\mathbb{T}$ denote the node containing t and $\mathbb{S}$ denote the node containing s in G'. If G is an undirected graph, $\mathcal{D}_{PQ}(G)$ is the graph G'. For directed graphs, $\mathcal{D}_{PQ}(G)$ is obtained by suitably modifying G' as follows. For each node μ reachable from $\mathbb{S}$ in G', μ is contracted into node $\mathbb{S}$. Likewise, each node μ that has a path to $\mathbb{T}$, is contracted into the node $\mathbb{T}$. Given any maximum (s,t)-flow f, $\mathcal{D}_{PQ}(G)$ can be obtained in $\mathcal{O}(m)$ time and has the following property. Without causing ambiguity, we refer to $(\mathbb{S},\mathbb{T})$-cut in $\mathcal{D}_{PQ}(G)$ by (s,t)-cut.

▶ **Theorem 13** ([33]). *Let G be any directed weighted graph on m edges with a designated source vertex s and designated sink vertex t. There is an $\mathcal{O}(m)$ space DAG $\mathcal{D}_{PQ}(G)$ that stores and characterizes each (s,t)-mincut in G as follows. An (s,t)-cut C is an (s,t)-mincut in G if and only if C is a 1-transversal cut in $\mathcal{D}_{PQ}(G)$.*

4 Limitation of the Existing Algorithm for Second (s,t)-mincut

We state here a limitation of the existing algorithm given by Vazirani and Yannakakis [37] to compute a second (s,t)-mincut. The algorithm for computing an (s,t)-cut of k^{th} minimum capacity by [37] uses $\mathcal{O}(n^{2(k-1)})$ maximum (s,t)-flows. So, for $k=2$, the algorithm uses $\mathcal{O}(n^2)$ maximum (s,t)-flow computations. In the same article [37], they stated the following property.

▶ **Lemma 14** (Lemma 3.2(1) in [37]). *Let G^f be the residual graph corresponding to a maximum (s,t)-flow f in G. Let H be an SCC in G^f and u,v be a pair of vertices in H. If the capacity of the least capacity cut separating $\{u\}$ and $\{v\}$ is k in H, then the least capacity cut separating $\{s,u\}$ and $\{v,t\}$ has capacity $\lambda+k$ in G.*

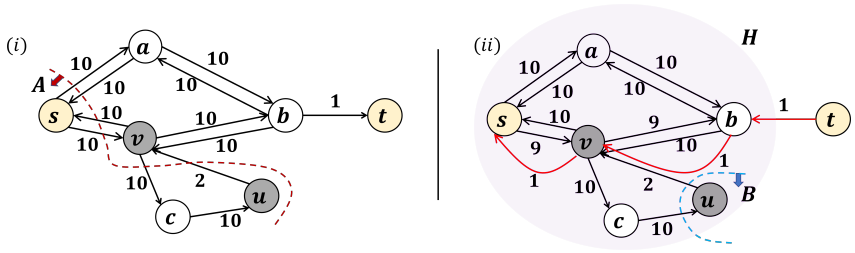


Figure 1 A counter example for Lemma 3.2(1) in [37] (i) A graph G. $V\setminus\{t\}$ is the (s,t)-mincut in G with capacity 1 and $c(A)$=22. (ii) H is an SCC in G^f and $c(B,H)=2$.

Vazirani and Yannakakis [37] used Lemma 14 to design an algorithm that computes second (s,t)-mincut using $\mathcal{O}(n)$ maximum (s,t)-flow computations (the algorithm preceding Theorem 3.3 in [37]). Unfortunately, Lemma 14 (or Lemma 3.2(1) in [37]) is not correct as shown in Figure 1(ii). H is the SCC in G^f containing vertices u and v. The least capacity cut B separating $\{u\}$ and $\{v\}$ has capacity 2 in H. By Lemma 14, the least capacity cut separating $\{s,u\}$ and $\{v,t\}$ must have capacity $2+1=3$ in G. However, as it can be verified easily using Figure 1(i), A is a least capacity cut separating $\{s,u\}$ and $\{v,t\}$ in G and its capacity is 22. Therefore, the algorithm stated above Theorem 3.3 in [37], fails to correctly output a second (s,t)-mincut.

5 Faster Algorithm for Second (s,t)-mincut

We design two efficient algorithms for computing a second (s,t)-mincut. Our first algorithm computes a second (s,t)-mincut for directed weighted graphs using $\mathcal{O}(n)$ maximum (s,t)-flow computations, which achieves the aim of Vazirani and Yannakakis [37]. This algorithm can be seen as an immediate application of the recently invented covering technique of [3]. We refer to the full version for the details of this algorithm. Our second algorithm takes a totally different approach. This approach is based on a relationship between the second (s,t)-mincuts and the global mincuts in directed weighted graphs. So, as our main result, we design an algorithm for computing a second (s,t)-mincut that uses $\tilde{\mathcal{O}}(\sqrt{n})$ maximum (s,t)-flow computations and works for directed graphs with integer edge capacities. We now provide an overview of this result.

Observe that a global mincut is not necessarily an (s,t)-cut in G. On the other hand, any second (s,t)-mincut can never be a global mincut in G. So apparently there does not seem to be any relationship between the global mincuts and the second (s,t)-mincuts of G.

Let us first work with a special case when graph G has exactly two (s,t)-mincuts – $\{s\}$ and $V \setminus \{t\}$. Suppose there exists a second (s,t)-mincut C in graph G such that C separates all the neighbors of s from all the neighbors of t. It is easy to compute a second (s,t)-mincut in this graph as follows. Compute a maximum (s,t)-flow after contracting all neighbors of s with s and all neighbors of t with t. The challenge arises when every second (s,t)-mincut has at least one contributing edge that is incident on s and/or t. We now show that the residual graph G^f plays a crucial role in overcoming this hurdle. Moreover, G^f turns out to establish the bridge between second (s,t)-mincuts and global mincuts.

In a seminal work in 1956, Ford and Fulkerson [19] established a strong duality between (s,t)-mincut and maximum (s,t)-flow, which is widely known as the *Maxflow-Mincut Theorem*. We begin by stating the following property, which is immediate from Maxflow-Mincut Theorem.

▶ **Fact 15.** *For any graph G with maximum (s,t)-flow f', every (s,t)-mincut in G is an (s,t)-cut of capacity zero in $G^{f'}$.*

It follows from Fact 15 that there is a bijective mapping between the set of all (s,t)-mincuts in G and the set of global mincuts containing s and not t in G^f. However, Fact 15 does not reveal any information on how a second (s,t)-mincut in G appears in residual graph G^f. To explore the structure of second (s,t)-mincuts in G^f, using only the conservation of flow and the construction of the residual graph, we provide a generalization of Fact 15 as follows.

▶ **Theorem 16** (Maxflow (Min+Δ)-cut Theorem)**.** *Let $G = (V,E)$ be a directed weighted graph with a designated source vertex s and a designated sink vertex t. Let f be any maximum (s,t)-flow in G and G^f be the corresponding residual graph. Let C be an (s,t)-cut in G. The capacity of C in G is $(\lambda + \Delta)$ if and only if the capacity of C in G^f is Δ, where $\Delta \geq 0$.*

Proof. Let $E_{out}(A,H)$ (likewise $E_{in}(A,H)$) denote the set of outgoing edges (likewise the set of incoming edges) of a cut A in a graph H. For any edge e in G, let $r(e)$ be the residual capacity, that is, $r(e) = w(e) - f(e)$. For any edge e in graph G^f, let $w'(e)$ denote the capacity of edge e. By construction of G^f, for each edge $e = (u,v) \in E_{out}(C,G)$ with $r(e) \neq 0$, there exists a forward edge $(u,v) \in E_{out}(C,G^f)$ with $w'(u,v) = r(e)$. Similarly, for each edge $e = (u,v) \in E_{in}(C,G)$ with $f(e) \neq 0$, there exists a backward edge $(v,u) \in E_{out}(C,G^f)$ with $w'(v,u) = f(e)$. This provides us with the following equality.

$$\begin{aligned}
\sum_{e' \in E_{out}(C,G^f)} w'(e') &= \sum_{e \in E_{out}(C,G)} r(e) + \sum_{e \in E_{in}(C,G)} f(e) \\
&= \sum_{e \in E_{out}(C,G)} (w(e) - f(e)) + \sum_{e \in E_{in}(C,G)} f(e) \\
&= \sum_{e \in E_{out}(C,G)} w(e) - \sum_{e \in E_{out}(C,G)} f(e) + \sum_{e \in E_{in}(C,G)} f(e) \\
&= c(C) - (f_{out}(C) - f_{in}(C)) = c(C) - \lambda \qquad \text{Using Lemma 10} \qquad (1)
\end{aligned}$$

Equation 1 completes the proof. ◀

It follows from Theorem 16 that every second (s,t)-mincut in G is a second (s,t)-mincut in G^f. Let the capacity of second (s,t)-mincut in G be $\lambda + \Delta_2$, where $\Delta_2 > 0$. Recall that our aim is to establish a relationship between second (s,t)-mincuts and global mincuts using G^f. So, by Theorem 16, we need to focus on global cuts of capacity exactly Δ_2 in G^f. However, observe that a global cut of capacity Δ_2 can never be a global mincut in G^f since $\Delta_2 > 0$. Moreover, there may exist *many* global cuts of capacity Δ_2 that cannot be a second (s,t)-mincut in G.

It is observed that a directed graph has global mincut capacity strictly greater than zero if it is an SCC. It turns out that there is exactly one nontrivial SCC, say H, in the residual graph G^f since G has exactly two trivial (s,t)-mincuts. Let us consider any global mincut A in H. Observe that A is not a second (s,t)-mincut in G^f since $s,t \in \overline{A}$. In fact, the capacity of any global cut A in H might be strictly less than the capacity of A in G^f. This is because of the existence of edges that are incident on s and/or t, which may contribute to A in G^f. Interestingly, exploiting the structure of G^f, the properties of SCC H, and Fact 15, we establish the following bijective mapping between the set of all second (s,t)-mincuts in G^f and the set of all global mincuts in H.

▶ **Lemma 17.** *Let C_1 be a global mincut in H and C_2 be a second (s,t)-mincut in G^f. Then,*
1. $c(C_2, G^f) = c(C_1, H)$,
2. $C_1 \cup \{s\}$ *is a second (s,t)-mincut in G^f and $C_2 \setminus \{s\}$ is a global mincut in H.*

Proof. Let us consider global mincut C_1 in graph H, and let $c(C_1, H) = \lambda_1$. Since H is an SCC, it is easy to show that $\lambda_1 > 0$. It follows from the construction of G^f that $C_1 \cup \{s\}$ is an (s,t)-cut in G^f. Observe that the edge-set of $C_1 \cup \{s\}$ in G^f and the edge-set of C_1 in H differ only on the set of edges, say E', that are incident on s and t in G^f. Since f is maximum (s,t)-flow, by Fact 15, s has outdegree zero; likewise, t has indegree zero in G^f. So, every edge in E' is an incoming edge to (s,t)-cut $C_1 \cup \{s\}$ in G^f. This implies that the contributing edges of (s,t)-cut $C_1 \cup \{s\}$ in G^f are the same as the contributing edges of cut C_1 in H. Hence, we arrive at the following equation.

$$c(C_1 \cup \{s\}, G^f) = c(C_1, H) = \lambda_1 \tag{2}$$

It is given that C_2 is a second (s,t)-mincut in G^f. Let $c(C_2, G^f) = \lambda_2$, where $\lambda_2 > 0$. It follows from Fact 15 that $\{s\}$ and $V \setminus \{t\}$ are the only (s,t)-mincuts in G^f. So, any (s,t)-cut in G^f except $\{s\}$ and $V \setminus \{t\}$ has capacity at least λ_2. Since C_1 is a cut in H, $C_1 \cup \{s\}$ is neither $\{s\}$ nor $V \setminus \{t\}$. Moreover, since $C_1 \cup \{s\}$ is an (s,t)-cut in G^f, we get $c(C_2, G^f) \leq c(C_1 \cup \{s\}, G^f)$. So, it follows from Equation 2 that $c(C_2, G^f) \leq \lambda_1$. Therefore, we arrive at the following inequality.

$$\lambda_2 \leq \lambda_1 \tag{3}$$

Observe that second (s,t)-mincut C_2 must separate a pair of vertices in H since $\{s\}$ and $V \setminus \{t\}$ are (s,t)-mincuts in G^f. Thus, the cut $C_2 \setminus \{s\}$ appears as a global cut in graph H. Thus, we can arrive at the following equation using similar arguments as in the case of Equation 2.

$$c(C_2 \setminus \{s\}, H) = c(C_2, G^f) = \lambda_2 \tag{4}$$

Since C_1 is a global mincut in H and $C_2 \setminus \{s\}$ is a global cut in H, $c(C_1, H) \leq c(C_2 \setminus \{s\}, H)$. Therefore, by Equation 4, we get the following inequality.

$$\lambda_1 \leq \lambda_2 \tag{5}$$

It follows from Equations 3 and 5 that $\lambda_2 = \lambda_1$. Evidently, $c(C_2, G^f) = c(C_1, H)$ and it completes the proof of (1). Now, by Equation 2, $c(C_1 \cup \{s\}, G^f) = \lambda_2$ and by Equation 4, $c(C_2 \setminus \{s\}, H) = \lambda_1$. Therefore, (s,t)-cut $C_1 \cup \{s\}$ is a second (s,t)-mincut in G^f and $C_2 \setminus \{s\}$ is a global mincut in H. This completes the proof of (2). ◀

The following lemma is immediate from Lemma 17 and Theorem 16.

▶ **Lemma 18.** *Suppose G has exactly two (s,t)-mincuts – $\{s\}$ and $V \setminus \{t\}$. There is an algorithm that, given any maximum (s,t)-flow in G, can compute a second (s,t)-mincut in G using one global mincut computation.*

It turns out that Lemma 18 does not immediately hold for graphs with one or more than two (s,t)-mincuts. For graphs with exactly one (s,t)-mincut, we suitably modify the SCC H in G^f. Finally, to extend our result for graphs having more than two (s,t)-mincuts, we partition the graph into a set of *roughly* disjoint subgraphs, each having at most two (s,t)-mincuts. This is obtained by applying a graph decomposition technique using DAG $\mathcal{D}_{PQ}(G)$ (Theorem 13), which was originally introduced in [3] for (s,t)-cuts of capacity $(\lambda+1)$ (Section 5.4 in [3]). This leads to Theorem 2(1). The proof of Theorem 2(2) is straightforward using standard techniques.

Cen et al. [9] recently designed an efficient algorithm for computing a global mincut in directed graphs with integer edge capacities. Their algorithm uses $\tilde{\mathcal{O}}(\sqrt{n})$ maximum (s,t)-flow computations to compute a global mincut. This result of Cen et al. [9], along with Theorem 2(1), immediately leads to Theorem 1.

6 Characterization of Minimum+1 (s,t)-cuts using Maximum flow

The widely known Maxflow-Mincut Theorem [19] provides a characterization of all (s,t)-mincuts using a maximum (s,t)-flow. However, it seems that any extension of this result might not exist for characterizing (s,t)-cuts of capacity $\lambda+1$. This is because no equivalent (s,t)-flow is known corresponding to a $(\lambda+1)$ (s,t)-cut, as stated in [3]. Interestingly, we show that, in fact, there exist close relationships between maximum (s,t)-flow and $(\lambda+1)$ (s,t)-cuts as follows.

For undirected multi-graphs, we first establish the following property for $(\lambda+k)$ (s,t)-cuts, where $k \geq 0$ is an integer, based on a maximum (s,t)-flow.

▶ **Lemma 19.** *Let C be a $(\lambda+k)$ (s,t)-cut in G, where $k \geq 0$ is an integer. Let f be any maximum (s,t)-flow in G and let $\mathcal{E} \subseteq E(C)$ be the set of edges such that $f(e) = 0$ for every edge $e \in \mathcal{E}$. Then, (**1**) $|\mathcal{E}| \leq k$ and (**2**) $|\mathcal{E}|$ is odd if and only if k is odd.*

Proof. Suppose C is a $(\lambda+k)$ (s,t)-cut in G. It follows from conservation of flow (Lemma 10), $f_{out}(C) \geq \lambda$, since f is a maximum (s,t)-flow and $f_{in}(C) \geq 0$. Let $f_{out}(C) = \lambda + j$, for any integer $j \geq 0$. Again by Lemma 10, $f_{in}(C) = j$. Therefore, $f_{out}(C) + f_{in}(C) = \lambda + 2j$. It follows that there are exactly $\lambda + 2j$ edges belonging to $E(C)$ that are carrying flow. Let $\mathcal{E} \subseteq E(C)$ be the set of remaining edges such that, for every edge $e \in \mathcal{E}$, $f(e) = 0$. In undirected graphs, every edge belonging to the edge-set of a cut is a contributing edge of the cut. Therefore, using $f_{out}(C) + f_{in}(C) = \lambda + 2j$, we arrive at the following equality.

$$|\mathcal{E}| = c(C) - (f_{out}(C) + f_{in}(C)) = \lambda + k - (\lambda + 2j) = k - 2j \tag{6}$$

It follows from Equation 6 that $|\mathcal{E}| \leq k$. In addition, since $2j$ is always an even number, this implies that $|\mathcal{E}|$ is odd if k is odd; otherwise $|\mathcal{E}|$ is even. ◀

By crucially exploiting Lemma 19, we now establish an interesting *flow-based characterization* of all the $(\lambda+1)$ (s,t)-cuts, which is of independent interest.

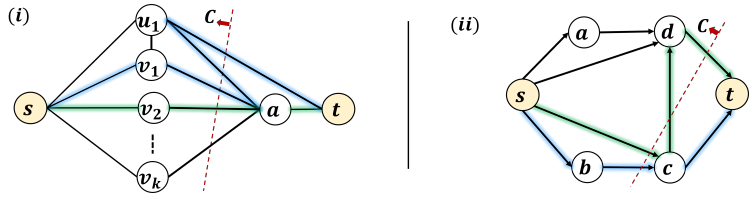


Figure 2 A colored edge represents that the edge carries flow and $\lambda = 2$. (i) Example showing that Theorem 20 cannot be generalized for $k \geq 2$ (ii) Cut C is a $(\lambda+1)$ (s,t)-cut with $k = 1$ where each edge belonging to $E(C)$ carries flow. So, $|\mathcal{E}|$ is even although k is odd.

▶ **Theorem 20** (Maxflow (Min+1)-cut Theorem)**.** *Let G be an undirected multi-graph with a designated source s and a designated sink t. Let f be any maximum (s,t)-flow in G. Then, an (s,t)-cut C in G is a $(\lambda+1)$ (s,t)-cut if and only if there exists exactly one edge $e \in E(C)$ such that*

1. *$f(e) = 0$ and*
2. *for every edge $e' \in E(C) \setminus \{e\}$, $f(e') = 1$ and e' carries flow in the direction C to $\overline{C}$.*

Proof. Suppose C is a $(\lambda+1)$ (s,t)-cut. Let $\mathcal{E} \subseteq E(C)$ be the set of edges such that $f(e) = 0$ for each edge $e \in \mathcal{E}$. By assigning the value of $k = 1$ in Lemma 19, we have $|\mathcal{E}| \leq 1$ and $|\mathcal{E}|$ is odd. Since $|\mathcal{E}|$ is odd, therefore, $|\mathcal{E}| = 1$. Since f is a maximum (s,t)-flow, by Lemma 10, $f_{out}(C) \geq \lambda$. Moreover, we have $|E(C)| = \lambda + 1$. Therefore, for every edge $e' \in E(C) \setminus \{e\}$, $f(e') = 1$ and e' carries flow in the direction C to $\overline{C}$.

We now prove the converse part. Suppose C is an (s,t)-cut satisfying properties (1) and (2). It follows that there is no edge $e' \in E(C)$ such that $f(e') = 1$ and carries flow in the direction $\overline{C}$ to C. So, $f_{in}(C) = 0$. By using Lemma 10, we have $f_{out}(C) = \lambda$. So, in $E(C)$, there are exactly λ edges that are carrying flow and exactly one edge that is carrying no flow. Therefore, $|E(C)| = \lambda + 1$. ◀

We are thankful to an anonymous reviewer for pointing out that the forward direction of Theorem 20 can also be established using the concept of edge-disjoint paths instead of using conservation of flow.

▶ Remark 21. An immediate generalization of Theorem 20 would be the following. An (s,t)-cut C is a $(\lambda + k)$ (s,t)-cut if and only if there are exactly k edges in $E(C)$ that carry zero flow and every other edge carries flow in the direction C to $\overline{C}$. However, Figure 2(i) shows that for $\lambda = 2$, (s,t)-cut C has capacity $k+2$ but there are $k-2$ edges in $E(C)$ carrying zero flow. Hence, this generalization of Theorem 20 is not possible for any $k \geq 2$.

7 Compact Structure for All Minimum+1 (s,t)-cuts

We address the following problem for undirected multi-graphs in this section.

▶ **Problem 22.** *Design a compact structure for storing and characterizing all $(\lambda+1)$ (s,t)-cuts.*

The problem of designing a compact structure for storing and characterizing all (s,t)-mincuts immediately reduces to Problem 22 by modifying the given graph as follows. Add a dummy source s' (likewise a dummy sink t') with $\lambda-1$ edges from s' to s (likewise t to t'). Interestingly, for directed multi-graphs, Baswana, Bhanja, and Pandey [3] provide a solution to Problem 22 by essentially reducing it to the problem of designing a compact structure for storing and characterizing all (s,t)-mincuts. However, their structure occupies $\mathcal{O}(mn)$ space. For undirected multi-graphs, we present a structure for storing and characterizing all $(\lambda+1)$ (s,t)-cuts that occupies $\mathcal{O}(\min\{m, n\sqrt{\lambda}\})$ space as follows.

We construct a graph G' such that every $(\lambda+1)$ (s,t)-cut of G appears as an (s,t)-mincut in G'. This will help us store and characterize all $(\lambda+1)$ (s,t)-cuts of G using a structure that stores and characterizes all (s,t)-mincuts in G'. To achieve this objective, we pursue the following simple idea – remove one edge from every $(\lambda+1)$ (s,t)-cut. A naive implementation of this idea might not work as follows. There may be a pair of edges e, e' contributing to the cut $C_1 \cap C_2$ defined by the intersection of two $(\lambda+1)$ (s,t)-cuts C_1, C_2. Even if none of the edges e, e' contribute to both C_1 and C_2, their removal will reduce the (s,t)-mincut capacity if $c(C_1 \cap C_2)$ is λ or $\lambda+1$ (refer to Figure 3(i)). In order to materialize the idea, we exploit Theorem 20, which motivates us to define a set of edges with respect to $(\lambda+1)$ (s,t)-cuts in the following way.

▶ **Definition 23** (Anchor edge). *For any given maximum (s,t)-flow f, an edge e is said to be an anchor edge if e contributes to a $(\lambda+1)$ (s,t)-cut and $f(e)=0$.*

By Maxflow-Mincut Theorem, the removal of a set of edges carrying no flow in f does not reduce the capacity of (s,t)-mincut. Moreover, by Theorem 20, for every $(\lambda+1)$ (s,t)-cut C in G, exactly one anchor edge contributes to C. Hence, every $(\lambda+1)$ (s,t)-cut, as well as (s,t)-mincut in G, appears as an (s,t)-mincut of capacity λ in $G \setminus \mathcal{A}$. It is also easy to observe that there may exist (s,t)-cuts with capacity more than $\lambda+1$ appearing as (s,t)-mincuts in $G \setminus \mathcal{A}$. However, using anchor edges $\mathcal{A}$, we can distinguish the $(\lambda+1)$ (s,t)-cuts as follows.

▶ **Lemma 24.** *An (s,t)-cut C is a $(\lambda+1)$ (s,t)-cut in G if and only if C is an (s,t)-mincut in $G \setminus \mathcal{A}$ and exactly one edge from $\mathcal{A}$ contributes to C.*

By Lemma 24, our compact structure consists of set of edges $\mathcal{A}$ and a structure that stores and characterizes all (s,t)-mincuts in $G \setminus \mathcal{A}$. It follows that the space occupied by our structure exceeds that of any structure for storing and characterizing all (s,t)-mincuts only by the size of $\mathcal{A}$. Therefore, we need to show that the set $\mathcal{A}$ is *small*.

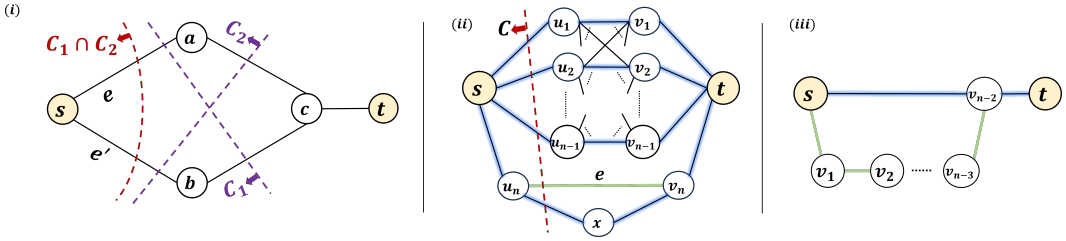


Figure 3 A blue edge carries flow and a green edge is an anchor edge. (i) $C_1 \cap C_2$ has capacity less than that of (s,t)-mincut in $G \setminus \{e, e'\}$. (ii) A graph with a given maximum (s,t)-flow satisfying $|\textsc{NoFlow}| = \Omega(n^2)$ but exactly one anchor edge e. (iii) Graph with exactly $(n-2)$ anchor edges.

7.1 Bounding the Cardinality of The Set of Anchor Edges

We now provide a tight bound on the cardinality of the set of all anchor edges $\mathcal{A}$. We present an efficient algorithm that, exploiting the properties of anchor edges, computes a set of edges $\mathcal{F}$ such that the following two properties hold.

1. The set of all anchor edges is a subset of $\mathcal{F}$.
2. The number of edges belonging to $\mathcal{F}$ is at most $n-2$.

By Definition 23, every anchor edge belongs to the set, say NoFlow, of all edges carrying zero flow in f. However, there exist graphs H such that, for any maximum (s,t)-flow in H, the cardinality of set NoFlow can be $\Omega(n^2)$, yet the number of anchor edges is only $\mathcal{O}(1)$ (refer to Figure 3(ii)). So, NoFlow provides a *loose* upper bound on the number of anchor

edges. Naturally, the question arises whether it is possible to eliminate a large number of edges from NoFlow while still keeping the set of all anchor edges intact. We answer this question in the affirmative by exploiting the following lemma.

▶ **Lemma 25.** *Let $\mathbb{H}$ be a cycle formed using a subset of edges in NoFlow. Then, no edge in $\mathbb{H}$ can be an anchor edge.*

Proof. Consider any edge $e \in \mathbb{H}$ and C be any (s,t)-cut in G such that $e \in E(C)$. Since C is a cut, C must intersect cycle $\mathbb{H}$ at least twice. Hence, C contains at least two edges that carry no flow. Therefore, it follows from Theorem 20 that C cannot be a $(\lambda+1)$ (s,t)-cut. Hence, by Definition 23, e cannot be an anchor edge. ◀

We now use Lemma 25 to construct a spanning forest $G_{\mathcal{F}}$ to provide an upper bound on the cardinality of set $\mathcal{A}$.

Construction of Spanning Forest $G_{\mathcal{F}} = (V, \mathcal{F})$. The vertex set of $G_{\mathcal{F}}$ is the same as G. Initially, there is no edge in $G_{\mathcal{F}}$. We construct graph $G_{\mathcal{F}}$ incrementally by executing the following step for each edge e in NoFlow. If a cycle is formed in $G_{\mathcal{F}} \cup \{e\}$, then by Lemma 25, e cannot be an anchor edge. Hence, we ignore edge e. Otherwise, if there is no cycle in $G_{\mathcal{F}} \cup \{e\}$, then insert edge e to $G_{\mathcal{F}}$.

It follows from the construction of $G_{\mathcal{F}}$ that $\mathcal{A} \subseteq \mathcal{F}$. Moreover, $|\mathcal{F}|$ is at most $n-1$ since $G_{\mathcal{F}}$ is a spanning forest. We now show a tight bound on $|\mathcal{A}|$, as well as $|\mathcal{F}|$, in the following lemma.

▶ **Lemma 26.** *Set $\mathcal{A}$, as well as set $\mathcal{F}$, contains at most $(n-2)$ edges.*

Proof. Let us assume to the contrary that $\mathcal{F}$ contains exactly $n-1$ edges. It follows that $G_{\mathcal{F}}$ is a spanning tree. Hence, $\mathcal{F}$ contains at least one edge from the edge-set of every (s,t)-cut in G. It implies that there exists an (s,t)-mincut C in G such that $\mathcal{F}$ contains at least one edge from the edge-set of C. By Maxflow-Mincut Theorem, for every edge $e \in E(C)$, $f(e) = 1$. Thus, $\mathcal{F}$ contains an edge e such that $f(e) = 1$, which is a contradiction since $\mathcal{F} \subseteq$ NoFlow. Since $\mathcal{A} \subseteq \mathcal{F}$, it follows that the cardinality of $\mathcal{A}$ is at most $n-2$. ◀

We show that there also exist graphs where $\mathcal{A}$ contains exactly $n-2$ edges (refer to Figure 3(*iii*)). Therefore, the bound on the set $\mathcal{A}$ given in Lemma 26 is tight.

We also show that set $\mathcal{A}$ can be obtained in $\mathcal{O}(m)$ time. Finally, using the best-known structure $\mathcal{D}_{PQ}$ for storing and characterizing all (s,t)-mincuts [33], we show that $\mathcal{D}_{PQ}(G \setminus \mathcal{A})$ occupies $\mathcal{O}(\min\{m, n\sqrt{\lambda}\})$ space, which leads to Theorem 6.

8 Dual Edge Sensitivity Oracle: Breaking the Quadratic Barrier

In this section, for undirected unweighted simple graphs, we design a subquadratic space data structure that can efficiently answer the query: report an (s,t)-mincut after the failure/insertion of any pair of edges. We assume G to be an undirected unweighted simple graph in this section. We provide an overview for handling the failure of edges in G. Note that handling the insertion of a pair of edges is along a similar line and is provided in the full version. Henceforth, let $e_1 = (x_1, y_1)$ and $e_2 = (x_2, y_2)$ be the two failed edges in G.

There is a folklore result that the residual graph G^f acts as an $\mathcal{O}(m)$ space dual edge sensitivity oracle for (s,t)-mincut that achieves $\mathcal{O}(m)$ query time. The query algorithm is derived from the *augmenting path* based algorithm for computing maximum (s,t)-flow by Ford and Fulkerson [19] (briefly explained as a warm-up below). However, it is known that

the residual graph occupies quadratic space if $m = \Omega(n^2)$. To design a subquadratic space dual edge sensitivity oracle for undirected unweighted simple graphs, instead of the residual graph G^f, we work with our compact structure, consisting of $\mathcal{D}$ and the set $\mathcal{A}$ of anchor edges, from Theorem 6. Recall that $\mathcal{D}$ is the graph $\mathcal{D}_{PQ}(G \setminus \mathcal{A})$, which is just a quotient graph of G^f. Hence, $\mathcal{D} \cup \mathcal{A}$ may fail to preserve the complete information of every path in G^f. However, we show that $\mathcal{D} \cup \mathcal{A}$ is still sufficient to answer dual edge failure queries using the same algorithm used for the folklore result using residual graph G^f.

Warm-up with residual graph. Observe that if none of the failed edges e_1, e_2 carry flow in maximum (s,t)-flow f, then the capacity of (s,t)-mincut remains unchanged in $G \setminus \{e_1, e_2\}$. Henceforth, we assume that edge e_1 always carries flow in the direction from x_1 to y_1. It follows from the construction of residual graph that there must exist a (t,s)-path P in G^f containing edge (y_1, x_1). We first reduce the flow in G using P in G^f, and then remove the edges (x_1, y_1) and (y_1, x_1) from G^f. Finally, using the concept of *augmenting paths* [19] in residual graph, we can verify in $\mathcal{O}(m)$ time whether the value of maximum (s,t)-flow becomes $\lambda - 1$ or remains λ in $G \setminus \{e_1\}$. In the residual graph corresponding to the obtained maximum (s,t)-flow in $G \setminus \{e_1\}$, repeat the same procedure for edge e_2 to verify whether failure of edge e_2 reduces the capacity of (s,t)-mincut in $G \setminus \{e_1\}$. This helps in reporting an (s,t)-mincut in the graph $G \setminus \{e_1, e_2\}$ in $\mathcal{O}(m)$ time.

Our solution using $\mathcal{D} \cup \mathcal{A}$. We now use the structure $\mathcal{D} \cup \mathcal{A}$ from Theorem 6 as a subquadratic space dual edge sensitivity oracle. $\mathcal{D}_{PQ}(G)$ (Theorem 13) can be used to design a single edge sensitivity oracle that occupies $\mathcal{O}(n)$ space [33, 3]. As observed by Baswana, Bhanja, and Pandey [3], $\mathcal{D}_{PQ}(G)$ can also handle dual edge failures for the special case when both failed edges contribute to only (s,t)-mincuts, using its reachability information. However, for handling any dual edge failures, the main difficulty arises when endpoints of both edges are mapped to the same node in $\mathcal{D}_{PQ}(G)$ [3]. Our structure $\mathcal{D} \cup \mathcal{A}$ addresses this difficulty seamlessly by first ensuring the following condition. The capacity of (s,t)-mincut changes only if the endpoints of at least one failed edge appear in different nodes of $\mathcal{D}$. This is achieved using the following property of $\mathcal{D}$.

▶ **Lemma 27.** *Any (s,t)-cut separating a pair of vertices mapped to the same node in $\mathcal{D}$ must have capacity at least $\lambda + 2$.*

Henceforth, we assume without loss of generality that endpoints of edge e_1 appear in different nodes in $\mathcal{D}$. Let us first handle the failure of edge e_1. It turns out that $\mathcal{D} \cup \mathcal{A}$ can easily report an (s,t)-mincut in $G \setminus \{e_1\}$. This exploits the following mapping of paths between G^f and $\mathcal{D} \cup \mathcal{A}$.

▶ **Lemma 28.** *Let u, v be any pair of vertices mapped to different nodes μ and ν in $\mathcal{D}$. There exists an (u,v)-path P in G^f if and only if there exists a (μ,ν)-path P_q in $\mathcal{D} \cup \mathcal{A}$. Moreover, path P_q is a quotient path of P.*

We establish Lemma 28 by using the fact that graph $\mathcal{D} \cup \mathcal{A}$ is a quotient graph of G^f by construction. Let D_1 be the graph obtained from $\mathcal{D} \cup \mathcal{A}$ after applying the query algorithm (described above) on $\mathcal{D} \cup \mathcal{A}$ for the failure of edge e_1. By following the mapping of paths in Lemma 28, let f_1 be the corresponding maximum (s,t)-flow in $G \setminus \{e_1\}$ and G^{f_1} denote the corresponding residual graph. On a high level, our technical contribution lies in showing that even after handling failure of e_1, D_1 still preserves enough information about augmenting paths in G^{f_1} that facilitates the handling of the failure of edge e_2 in $G \setminus \{e_1\}$. We now provide the overview.

To handle the failure of edge e_2, the obtained graph D_1 must satisfy the following property, which actually holds between graphs $\mathcal{D} \cup \mathcal{A}$ and G^f (refer to Lemma 27).

▶ **Lemma 29.** *D_1 is a quotient graph of G^{f_1} and for every path P in D_1, there is a path P_1 in G^{f_1} such that P is a quotient path of P_1.*

In order to establish Lemma 29, the challenging case appears when the (s,t)-mincut remains unchanged after the failure of e_1. In this case, let us first observe the change in the residual graph after reducing the flow that was passing through edge e_1. In the resulting residual graph, the query algorithm finds a path P from s to t, and flips the direction of every edge belonging to it. A similar update is executed by the query algorithm using a path P_1 in graph $\mathcal{D} \cup \mathcal{A}$, where P_1 is a quotient path of P. This could lead to the following scenario for the resulting graph D_1. There is a path P_{uv} in D_1, but there is no path P^r_{uv} in G^{f_1} such that P_{uv} is a quotient path of P^r_{uv}. In other words, Lemma 29 may fail to hold. So, we might report an incorrect (s,t)-mincut in $G \setminus \{e_1, e_2\}$. Interestingly, exploiting the SCC structure of G^{f_1}, the structure of $\mathcal{D}$, and Lemma 27, we ensure that such a scenario never occurs. This allows us to handle the failure of e_2 using D_1 exactly in the same way as handling the failure of e_1 using $\mathcal{D} \cup \mathcal{A}$. This leads to Theorem 8.

9 Conclusion

In this article, we have mainly solved the following two problems, achieving significant improvements over the existing bounds.

Firstly, we have addressed the problem of designing an efficient algorithm for computing a second (s,t)-mincut in directed integer-weighted graphs. Our algorithm achieves a running time that is $\tilde{\Omega}(n\sqrt{n})$ times faster than the existing best-known algorithm [37]. Most importantly, to arrive at the solution, a close relationship between global mincut and second (s,t)-mincut is exposed for the first time here. We believe that this connection between global cuts and (s,t)-cuts (via residual graph) should be quite useful for addressing future research problems, including the problem of designing an algorithm for computing an (s,t)-cut of k^{th} minimum capacity, for any $k > 2$, that breaks the existing bound [37].

Finally, we have addressed the problem of designing a dual edge sensitivity oracle for (s,t)-mincut in undirected unweighted simple graphs. This problem is shown to be closely related to the study of (s,t)-cuts of capacity minimum+1, which is a special case of the second (s,t)-mincut in unweighted graphs. Our dual edge sensitivity oracle breaks the recently established quadratic lower bound on space in multi-graphs [6] by a factor of $\Omega(\sqrt{n})$. The foundation of this result lies in our newly designed structure for storing and characterizing all minimum+1 (s,t)-cuts, which also improves the existing best-known structure [3] by $\Omega(n)$ factor. Now, it is interesting to see whether the query time of our dual edge sensitivity oracle can be improved and whether our approach can help us in designing a sensitivity oracle for handling more than two edge insertions/failures.

References

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

2 Surender Baswana and Koustav Bhanja. Vital edges for (s, t)-mincut: Efficient algorithms, compact structures, & optimal sensitivity oracles. In *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.17.

3 Surender Baswana, Koustav Bhanja, and Abhyuday Pandey. Minimum+ 1 (s, t)-cuts and dual-edge sensitivity oracle. *ACM Transactions on Algorithms*, 19(4):1–41, 2023. doi:10.1145/3623271.

4 András A. Benczúr. A representation of cuts within 6/5 times the edge connectivity with applications. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 92–102. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492466.

5 Koustav Bhanja. Optimal sensitivity oracle for steiner mincut. In *35th International Symposium on Algorithms and Computation (ISAAC 2024)*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ISAAC.2024.10.

6 Koustav Bhanja. Minimum+1 steiner cut and dual edge sensitivity oracle: Bridging gap between global and (s, t)-cut. In *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, July 8-11, 2025, Aarhus, Denmark*, volume 334 of *LIPIcs*, pages 27:1–27:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ICALP.2025.27.

7 Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Approximate distance sensitivity oracles in subquadratic space. *TheoretiCS*, 3, 2024. doi:10.46298/THEORETICS.24.15.

8 PM Camerini, L Fratta, and F Maffioli. The k shortest spanning trees of a graph. *Int. Rep*, pages 73–10, 1974.

9 Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Kent Quanrud. Minimum cuts in directed graphs via partial sparsification. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1147–1158. IEEE, 2021. doi:10.1109/FOCS52979.2021.00113.

10 Ruoxu Cen, Jason Li, and Debmalya Panigrahi. Augmenting edge connectivity via isolating cuts. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3237–3252. SIAM, 2022. doi:10.1137/1.9781611977073.127.

11 Diptarka Chakraborty, Kushagra Chatterjee, and Keerti Choudhary. Pairwise reachability oracles and preservers under failures. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 35:1–35:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.35.

12 Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 130:1–130:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.130.

13 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

14 Efim A Dinitz, Alexander V Karzanov, and Michael V Lomonosov. On the structure of the system of minimum edge cuts of a graph. *Studies in discrete optimization*, pages 290–306, 1976.

15 Yefim Dinitz and Zeev Nutov. A 2-level cactus model for the system of minimum and minimum+ 1 edge-cuts in a graph and its incremental maintenance. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 509–518, 1995. doi:10.1145/225058.225268.

16 Yefim Dinitz and Alek Vainshtein. The general structure of edge-connectivity of a vertex subset in a graph and its incremental maintenance. odd case. *SIAM J. Comput.*, 30(3):753–808, 2000. doi:10.1137/S0097539797330045.

17 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 506–515. SIAM, 2009. doi:10.1137/1.9781611973068.56.

18 David Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998. doi:10.1137/S0097539795290477.

19 L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

20 Zvi Galil and Xiangdong Yu. Short length versions of menger's theorem. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 499–508, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/225058.225267.

21 Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

22 Gramoz Goranci and Monika Henzinger. Efficient data structures for incremental exact and approximate maximum flow. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 69:1–69:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.69.

23 Gramoz Goranci, Monika Henzinger, and Mikkel Thorup. Incremental exact min-cut in polylogarithmic amortized update time. *ACM Trans. Algorithms*, 14(2):17:1–17:21, 2018. doi:10.1145/3174803.

24 David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 21–30, USA, 1993. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=313559.313605.

25 David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996. doi:10.1145/234533.234534.

26 N. Katoh, T. Ibaraki, and H. Mine. An algorithm for finding k minimum spanning trees. *SIAM Journal on Computing*, 10(2):247–255, 1981. doi:10.1137/0210017.

27 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019. doi:10.1145/3274663.

28 Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 85–92. IEEE, 2020. doi:10.1109/FOCS46700.2020.00017.

29 Ernst W Mayr and C Greg Plaxton. On the spanning trees of weighted graphs. *Combinatorica*, 12(4):433–447, 1992. doi:10.1007/BF01305236.

30 Hiroshi Nagamochi, Kazuhiro Nishimura, and Toshihide Ibaraki. Computing all small cuts in an undirected network. *SIAM Journal on Discrete Mathematics*, 10(3):469–481, 1997. doi:10.1137/S0895480194271323.

31 Dalit Naor, Dan Gusfield, and Charles Martel. A fast algorithm for optimally increasing the edge connectivity. *SIAM Journal on Computing*, 26(4):1139–1165, 1997. doi:10.1137/S0097539792234226.

32 Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 481–490. ACM, 2015. doi:10.1145/2767386.2767408.

33 Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *In Rayward-Smith V.J. (eds) Combinatorial Optimization II. Mathematical Programming Studies*, 13(1):8–16, 1980. doi:10.1007/BFb0120902.

34 Liam Roditty. On the k shortest simple paths problem in weighted directed graphs. *SIAM Journal on Computing*, 39(6):2363–2376, 2010. doi:10.1137/080730950.

35 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

36 Jan Van Den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023. doi:10.1109/FOCS57990.2023.00037.

37 Vijay V Vazirani and Mihalis Yannakakis. Suboptimal cuts: Their enumeration, weight and number. In *International Colloquium on Automata, Languages, and Programming*, pages 366–377. Springer, 1992. doi:10.1007/3-540-55719-9_88.

38 Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

Bandwidth vs BFS Width in Matrix Reordering, Graph Reconstruction, and Graph Drawing

David Eppstein
University of California, Irvine, CA, USA

Michael T. Goodrich
University of California, Irvine, CA, USA

Songyu (Alfred) Liu
University of California, Irvine, CA, USA

Abstract

We provide the first approximation quality guarantees for the Cuthull-McKee heuristic for reordering symmetric matrices to have low bandwidth, and we provide an algorithm for reconstructing bounded-bandwidth graphs from distance oracles with near-linear query complexity. To prove these results we introduce a new width parameter, BFS width, and we prove polylogarithmic upper and lower bounds on the BFS width of graphs of bounded bandwidth. Unlike other width parameters, such as bandwidth, pathwidth, and treewidth, BFS width can easily be computed in polynomial time. Bounded BFS width implies bounded bandwidth, pathwidth, and treewidth, which in turn imply fixed-parameter tractable algorithms for many problems that are NP-hard for general graphs. In addition to their applications to matrix ordering, we also provide applications of BFS width to graph reconstruction, to reconstruct graphs from distance queries, and graph drawing, to construct arc diagrams of small height.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Graph algorithms, graph theory, graph width, bandwidth, treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.69

Funding *David Eppstein*: Supported by NSF grant 2212129, as well for Michael Goodrich and Songyu (Alfred) Liu.

1 Introduction

When a graph has a linear layout of low bandwidth, it is natural to guess that breadth-first search produces a low-width layout. This is the underlying principle of the widely used Cuthill–McKee algorithm [13] for reordering symmetric matrices into band matrices with low bandwidth. In this paper, we quantify this principle, for the first time, by providing the first worst-case analysis of the Cuthill–McKee algorithm, proving polylogarithmic upper and lower bounds for its performance on bounded-bandwidth graphs. Using the same techniques, we also study the problem of reconstructing an unknown graph, given access to a distance oracle. We use a method based on breadth-first search to reconstruct graphs of bounded bandwidth in near-linear query complexity. Additionally, we apply our results in graph drawing, by constructing arc diagrams of low height for any graph of bounded bandwidth.

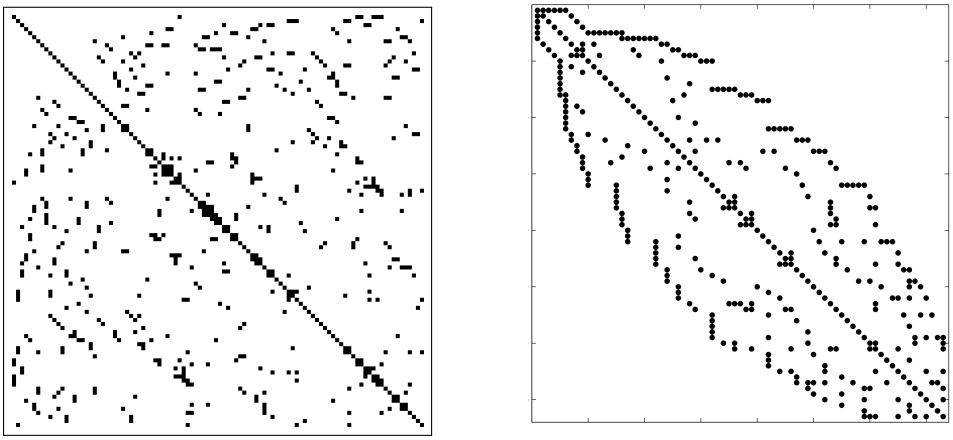
The key to our results is a natural but previously-unstudied graph width parameter, the **breadth-first-search width** or **BFS width** of a graph, which we define algorithmically in terms of the size of the layers in a breadth-first search tree. To develop our results, we prove that bandwidth and BFS width are polylogarithmically tied: the graphs of bandwidth $\leq b$, for any constant b, have BFS width that is both upper bounded (for all graphs of bandwidth $\leq b$) and lower bounded (for infinitely many graphs of bandwidth b) by functions of the form $\log^{f(b)} n$. To provide context for our contributions, we review the background of our three application areas below and then describe our new results in more detail for each.


© David Eppstein, Michael T. Goodrich, and Songyu Liu;
licensed under Creative Commons License CC-BY 4.0



Sparse Numerical Linear Algebra

A symmetric matrix, A, is called a **band matrix** if its nonzeros are confined to a **band** of consecutive diagonals centered on the main diagonal; the **bandwidth** is the number of these nonzero diagonals on either side of the main diagonal. That is, it is the maximum value of $|i - j|$ for the indexes of a nonzero matrix entry $A_{i,j}$. Any symmetric matrix corresponds to an undirected graph, with n numbered vertices, and with an edge $\{i, j\}$ for each off-diagonal nonzero entry $A_{i,j}$; the numbering of vertices describes a **linear layout** of this graph and the bandwidth of this layout is the largest difference between the positions of any two adjacent vertices in this linear layout. The bandwidth of the graph itself is the minimum bandwidth of any of its linear layouts; the goal of the Cuthill–McKee algorithm is to permute the vertices of the graph into a layout with bandwidth close to this minimum, and correspondingly to permute the rows and columns of the given matrix to produce an equivalent symmetric band matrix of low bandwidth. See Figure 1. Such a reordering, for instance, can be used in sparse numerical linear algebra to reduce fill-in (zero matrix elements that become nonzero) when applying Gaussian elimination to the matrix.



(a) (b)

Figure 1 Illustrating the Cuthill-McKee algorithm. (a) A sparse matrix; (b) a sparse matrix reordered according to the Cuthill-McKee algorithm. Nonzero entries are shown as black spots. Left: public-domain image by Oleg Alexandrov; right: image by Wikipedia user Kxx licensed under the CC BY-SA 3.0 license.

Graph bandwidth is NP-complete [20] and hard to approximate to within any constant factor, even for trees of a very special form [15]. For any fixed b, testing whether bandwidth $\leq b$ is polynomial, but with b in the exponent of the polynomial [25]. Therefore, practitioners have resorted to heuristics for bandwidth, rather than exact optimization algorithms. Notable among these are the Cuthill–McKee algorithm [13], and the closely related reverse Cuthill–McKee algorithm [21]; both are heavily cited and widely used for this task. More sophisticated methods have also been used and tested experimentally [34].

The best known polynomial time algorithm provides a $O(\log^3 n\sqrt{\log \log n})$-approximation factor for bandwidth on general graphs with high probability using a semi-definite relaxation [16]. A randomized polynomial time $O(\log^{2.5} n)$-approximation algorithm is known for the special cases of trees and chordal graphs using caterpillar decomposition [24]. In contrast, the Cuthill–McKee and reverse Cuthill–McKee algorithms are much simpler since they only rely on BFS and are deterministic. Despite their simplicity, we are unaware of any

past work providing worst-case performance guarantees for the Cuthill–McKee and reverse Cuthill–McKee algorithms. The only theoretical upper bound we are aware of regarding these algorithms concerns random matrices whose optimal bandwidth is sufficiently large, for which this algorithm will produce a constant factor approximation. The same work also provides a logarithmic lower bound on the approximation factor for matrices with optimal bandwidth 2, weaker than our result [35]. Instead, we are interested in deterministic guarantees for these algorithms for matrices of low optimal bandwidth. We prove the following results:

- For $n \times n$ matrices of bounded optimal bandwidth $\leq b$, we show that the Cuthill–McKee algorithm and the reverse Cuthill–McKee algorithm will find a reordering of bandwidth upper bounded by a function of the form $\log^{f(b)} n$. Equivalently, for n-vertex graphs of bandwidth $\leq b$, these algorithms will find a linear layout of bandwidth $\leq \log^{f(b)} n$.
- We show that there exist matrices for which this polylogarithmic dependence is necessary. For every polylogarithmic bound $\log^k n$ on the bandwidth, there is a constant bandwidth bound b such that, for infinitely many n, there exist matrices of optimal bandwidth $\leq b$ that cause both the Cuthill–McKee algorithm and the reverse Cuthill–McKee algorithm to produce reordered matrices of bandwidth $\Omega(\log^k n)$.

We remark that, for any graph,

$$O(1) \text{ BFS width} \implies O(1) \text{ bandwidth} \implies O(1) \text{ pathwidth} \implies O(1) \text{ treewidth [28]},$$

and graphs of bandwidth b have tree-depth $O(b \log n/b)$ [23]. Thus, many algorithmic results based on width parameters such as pathwidth, treewidth, and tree-depth are also available for BFS-width. However, both pathwidth and tree-depth have approximation algorithms with fixed polylogarithmic approximation ratios [6, 19], better than the polylogarithmic ratio with a variable exponent depending on the optimal bandwidth that we prove for the Cuthill–McKee algorithm.

Graph Reconstruction

We also provide new results for **graph reconstruction**, a well-motivated problem with numerous applications, such as learning road networks [2] and communication network mapping [1]. Suppose we are given the vertex set but not the edge set of a graph G. The **graph reconstruction** problem is to determine all the edges of G, by asking queries about G from an oracle, with the goal of minimizing the **query complexity**, the number of queries to the oracle [27, 32]. Various oracles can be used. In particular, we consider the **distance oracle** from several previous works [27, 32]. This oracle takes a pair of vertices and returns the number of edges on a shortest path between the two given vertices. Following previous work [27, 32], the graph is assumed to be connected, undirected, and unweighted. It is also generally assumed to have bounded degree, because distinguishing an n-vertex star from a graph with one additional edge between the leaves of the star would have a trivial quadratic lower bound. For reconstruction from distance oracles, the following results are known:

- For graphs of maximum degree Δ, there is a randomized algorithm with query complexity $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log\log n)$, which is $\tilde{O}(n^{3/2})$ when $\Delta = O\big(\text{polylog}(n)\big)$ [27].[1]
- Chordal graphs can be reconstructed in randomized query complexity $\tilde{O}(n)$ when $\Delta = O(\log\log n)$ [27]. The k-chordal graphs can be reconstructed in randomized query complexity $O_{\Delta,k}(n \log n)$ when $\Delta = O(1)$ [4].

[1] We use $\tilde{O}(*)$ to denote asymptotic bounds that ignore polylogarithmic factors.

- Outerplanar graphs can be reconstructed in randomized query complexity $\tilde{O}(n)$ when Δ is polylogarithmic [27].
- For reconstructing uniformly random Δ-regular graphs with maximum degree $\Delta = O(1)$, there is a randomized algorithm using $\tilde{O}(n)$ queries in expectation [32].

Our new results in this area are that the graphs of bounded bandwidth can be reconstructed deterministically in query complexity $\tilde{O}(n)$, by a very simple algorithm. More generally, the graphs of BFS width B can be reconstructed deterministically in query complexity $O(nB)$. No additional assumption on Δ is needed here because bounded bandwidth and bounded BFS width both automatically imply bounded degree.

Graph Drawing

Graph drawing is the study of methods for visualizing graphs; see, e.g., [5,30]. One type of graph visualization tool that has received considerable attention is the **arc diagram** of a graph, $G = (V, E)$, where the vertices of G are laid out as points on a straight line and edges are drawn as semicircular arcs or straight-line segments (for consecutive points) joining pairs of such points. See Figure 2.

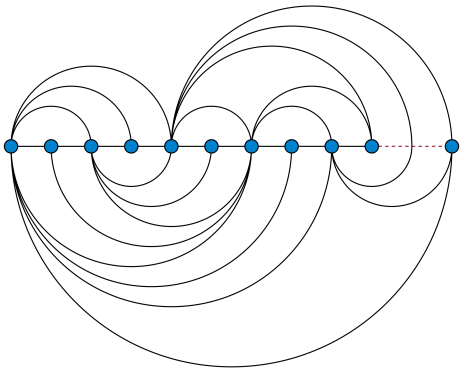


Figure 2 An example arc diagram. Public-domain image by David Eppstein.

Arc diagrams have received considerable attention, both as a visualization tool and as a topic of study in discrete mathematics; see, e.g., [9,11,14,30,31,36]. As a direct consequence of Theorem 14, when an n-vertex graph has an arc diagram of bounded height, we can find a drawing with polylogarithmic height. Here, the height of an arc diagram, with vertices placed at integer positions along the x-axis, is just half the bandwidth of the linear layout describing the vertex placement.

2 Preliminaries

We define BFS width as follows. From a given undirected graph, $G = (V, E)$, choose arbitrarily a starting vertex v in G and perform a breadth-first search of G from v. The edges traversed during this search define a breadth-first search tree T such that the depth of each vertex in T equals its unweighted distance from v, the number of edges in a shortest path from v. We define the **layer** $L_i(v)$ to consist of all vertices at distance i from v; if $u \in L_i(v)$, we say that u has **layer number** i. The maximum size of any $L_i(v)$ set is the BFS width of G from v and the BFS width of G is the maximum BFS width taken over all choices of the starting vertex v in G. That is, we have the following formal definition of BFS width building upon the definition of width in layered graph drawing [5].

▶ **Definition 1** (BFS Width). *Let $G = (V, E)$ be a graph. A **layering** of G is a partition of V into subsets $L_0, L_1, \ldots$, and the **width** of G for this layering is $\max_{i \in \mathbb{Z}_{\geq 0}} |L_i|$. Any vertex, $v \in V$, defines a layering by letting $L_i(v) = \{u \in V : d(v, u) = i\}$. The **BFS width of G from v**, denoted $\mathbf{bfsw}(G, v)$, is defined as $\mathbf{bfsw}(G, v) = \max_{i \in \mathbb{Z}_{\geq 0}} |L_i(v)|$. The **BFS width of G**, denoted $\mathbf{bfsw}(G)$, is defined as $\mathbf{bfsw}(G) = \max_{v \in V} \mathbf{bfsw}(G, v) = \max_{v \in V} \max_{i \in \mathbb{Z}_{\geq 0}} |L_i(v)|$.*

We also define the **minimum BFS width**, $\mathbf{bfsw}_{\min}(G) = \min_{v \in V} \mathbf{bfsw}(G, v)$. These two BFS width parameters can differ significantly, and we will show that there exist graphs such that $\mathbf{bfsw}(G)/\mathbf{bfsw}_{\min}(G) = \Omega(\log n)$. Clearly, $\mathbf{bfsw}(G)$ and $\mathbf{bfsw}_{\min}(G)$ can both easily be computed in $O\big(n(n+m)\big)$ time for any graph, G, with n vertices and m edges by performing n breadth-first searches; see, e.g., [12, 22]. In contrast, it is NP-hard to compute other well-known width parameters for a graph, including bandwidth [18], pathwidth, and treewidth [7]. Nevertheless, we prove that bounded BFS width implies bounded bandwidth for the graph, which, in turn, implies bounded pathwidth and treewidth for the graph.

Some Preliminary Facts About Bandwidth and BFS Width

Bandwidth

Two lower bounds on bandwidth are as follows:

▷ Claim 2 (Degree lower bound [10]). $\mathbf{bw}(G) \geq \lceil \Delta/2 \rceil$, where Δ is the graph's maximum degree.

▷ Claim 3 (Local density lower bound [24]). $\mathbf{bw}(G) \geq D(G) \coloneqq \max_{v \in V, d} \left\lceil \frac{|N(v,d)|}{2d} \right\rceil$, where $D(G)$ is the local density of G, and $N(v, d)$ is vertices at distance at most d from v (excluding v).

When v is a maximum degree vertex and $d = 1$, this implies the degree lower bound.

BFS Width

Recall that a **caterpillar tree** is a tree in which every vertex is within distance 1 of a central path; i.e., the central path contains every vertex of degree 2 or more; see, e.g., [26]. Caterpillar trees have applications in chemistry and physics [17], and they are the graphs of pathwidth one [33]. A simple, but useful, fact regarding BFS width is the following:

▶ **Theorem 4.** *If G is a caterpillar tree with maximum degree, $\Delta \geq 2$, then $\mathbf{bfsw}(G) \leq 2(\Delta - 1)$.*

Proof. To upper bound $\mathbf{bfsw}(G)$, we want to find a root vertex v that can maximize the BFS width of G from v. Choose a vertex, v, that is not one of the end points on the central path as the root for a BFS tree, T. Then layer 1 will have size at most Δ and each vertex of degree Δ in G that is not v will correspond to an interior vertex in T with at most $\Delta - 1$ children. All degree Δ vertices are on the central path. The maximum layer size occurs in T when two such vertices are in the same layer. The layer below it has $2(\Delta - 1)$ children. ◀

We also have the following.

▶ **Theorem 5.** *For any graph, G, its bandwidth $\mathbf{bw}(G) \leq 2\big(\mathbf{bfsw}_{\min}(G)\big) - 1$.*

Proof. Let T be the BFS tree for G rooted at a vertex, v, that achieves the minimum BFS width, $\mathbf{bfsw}_{\min}(G) = \mathbf{bfsw}(G, v)$. By definition, every layer in T has at most $\mathbf{bfsw}_{\min}(G)$ vertices. It is well-known that in a graph, G, with a BFS tree, T, every non-tree edge, (u, v),

connects two vertices whose layer numbers differ by at most 1; see, e.g., [12,22]. Thus, we can number the vertices of T as $1, 2, \ldots, n$ so as to have nondecreasing layer numbers. The BFS ordering is one way to achieve this. This numbering defines a linear layout of the vertices. The maximum distance on the x-axis between two endpoints of an edge, therefore, will be at most that determined by the first vertex in a layer i and the last vertex in a layer $i+1$, which can be at most $2\big(\mathbf{bfsw}_{\min}(G)\big) - 1$. ◀

This immediately implies the following.

▶ **Corollary 6.** *For any graph, G,* $\mathbf{bw}(G) \leq 2\big(\mathbf{bfsw}(G)\big) - 1$.

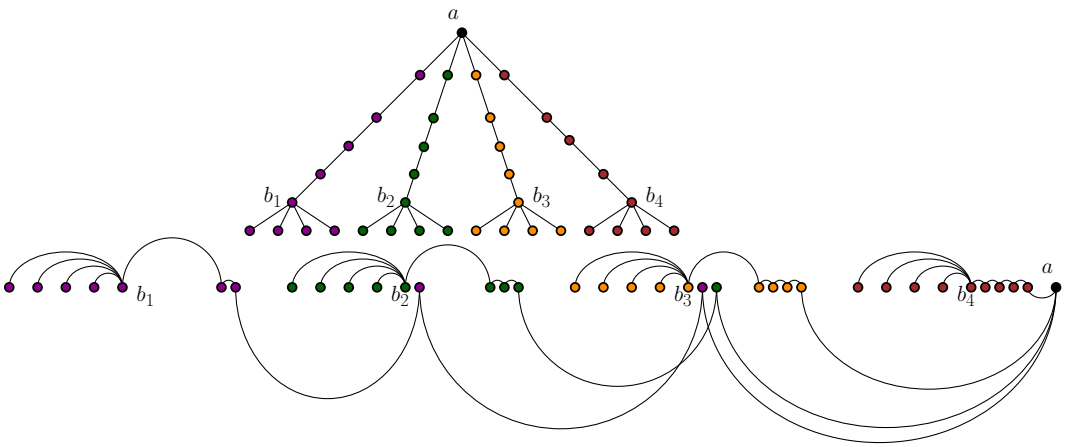


Figure 3 An example graph with $\mathbf{bfsw}(G) = \Theta(n)$ and $\mathbf{bfsw}_{\min}(G) = \Theta(n)$ and a linear layout that achieves its optimal bandwidth $\mathbf{bw}(G) = \Theta(\sqrt{n})$.

Given a linear layout of the vertices of a graph with respect to bandwidth, we say that the **length** of an edge, $\{u, v\}$, is one plus the number of vertices between u and v.

▶ **Theorem 7.** *There is a graph, G, with n vertices such that its bandwidth* $\mathbf{bw}(G) = \Theta(\sqrt{n})$ *but* $\mathbf{bfsw}(G) = \Theta(n)$ *and* $\mathbf{bfsw}_{\min}(G) = \Theta(n)$.

Proof. The example is illustrated in Figure 3. Take a star with $\sqrt{n}$ leaves and subdivide each edge $\sqrt{n}$ times. Then add a star with $\sqrt{n}$ leaves at the end of each path. This example first appeared in [24] but they only gave a lower bound on bandwidth and did not compute the bandwidth. By the degree lower bound, $\mathbf{bw}(G) = \Omega(\sqrt{n})$. Regardless of which vertex is chosen as the root for BFS, BFS will produce a layout with bandwidth $\Theta(n)$. Now it remains to show a linear layout with bandwidth $O(\sqrt{n})$.

The path from a to vertex b_i and the star at b_i are considered as one block and we call this block B_i. Suppose we have already placed $B_{\sqrt{n}-1}, B_{\sqrt{n}-2}, \ldots, B_{i+1}$. Now we describe how to place B_i. All $\sqrt{n}$ leaves of b_i are placed on its left. Next we place the path from a to b_i. Starting from a, we place one vertex to the right of $b_{\sqrt{n}-1}, b_{\sqrt{n}-2}, \ldots, b_{i+1}$. This step uses $\sqrt{n} - i - 1$ vertices. The remaining path segment with $i+1$ vertices are placed to the right of b_i.

Next we analyze the bandwidth. The edge between b_i and its leftmost leaf has length $\sqrt{n}$. Any vertex b_i introduces a path vertex to the right of $b_{\sqrt{n}-1}$. In total there are $\sqrt{n} - 2$ vertices between $b_{\sqrt{n}-1}$ and its parent in the tree. Block $B_{\sqrt{n}}$ has $2\sqrt{n} + 1$ vertices. Thus, the length of the first edge on path ab_1 is at most $4\sqrt{n}$. The linear layout has bandwidth $O(\sqrt{n})$. ◀

This shows that there are graphs for which the linear layout produced by BFS is off by a factor of $\Theta(\sqrt{n})$.

As noted above, there are many algorithmic applications with respect to graphs with bounded bandwidth, which, in turn, implies bounded pathwidth and treewidth. Theorem 5 implies that a graph with bounded minimum BFS width has bounded bandwidth. Thus, any polynomial-time algorithm for graphs with bounded bandwidth, pathwidth, or treewidth (e.g., see [3,8]) also applies to any graph with bounded BFS width for one of its vertices, with the added benefit that we can determine the minimum BFS width for a graph in polynomial time. Unfortunately, as the following theorem shows, bounded treewidth does not imply bounded BFS width.

▶ **Theorem 8.** *There is a graph, G, with n vertices such that its treewidth* $\mathbf{tw}(G) = 1$ *but* $\mathbf{bfsw}(G) = n - 1$ *and* $\mathbf{bfsw}_{\min}(G) = n - 2$.

Proof. Let G be the star graph with internal vertex v. The BFS tree, T, from the vertex v will have $n - 1$ vertices on its layer 1. For any vertex $w \neq v$, the BFS tree, T, from w will have v on its layer 1, and the remaining $n - 2$ vertices of G on its layer 2. ◀

In these subsequent sections, we sometimes drop the graph, G, in width notations when the context is clear. When the graph is a tree with root v, a natural layering is defined by distances to the root and we use the term width to denote the BFS width of G from v.

3 Graphs of Bounded Bandwidth and Polylogarithmic BFS Width

In this section, we show how to construct a graph to prove that when bandwidth is bounded, BFS width can be as large as polylogarithmic in the number of vertices. This graph also proves that BFS width and minimum BFS width can differ by a logarithmic factor. The graph that we use to establish this lower bound is an arbitrarily large binary tree. Our construction is an induction based on defining trees in terms of a level parameter, k, which is the primary inductive term, plus an independent height parameter, h, which determines the size, n, of the tree.

Level 1. A level 1 tree, $T_{1,i}$, is simply a path, which defines a tree when viewed as being rooted at one of the ends of this path. Thus, $\mathbf{bw}(T_{1,i}) = 1$ and $\mathbf{bfsw}(T_{1,i}) = 2$. Before next jumping to the general case, for $k > 1$, let us first describe the level 2 trees, which introduce a constructive pattern we repeat for the general case.

Level 2. A level 2 tree $T_{2,j}$ is defined for height, $h = 2^j$, for any integer, $j > 1$. We start with $j + 1$ level 1 trees, where $T_{1,i}$ has height $h_i = 2^i - 1$, for $i = 0, 1, \ldots, j$. We also define another level 1 tree of height $h = 2^j$, which we call the **spine**. We connect each tree, $T_{1,i}$, with an edge to a vertex v_i on the spine so that every leaf in the resulting tree has the same depth. See Figure 4.

Thus, each subtree "hanging off" the spine is a path with height $h_i = 2^i - 1$, where $i = 0, 1, \ldots, j$. The height of the level 2 tree is $h = 2^j$, where j is independent of the level number 2. As mentioned above, all leaves in a level 2 tree have the same depth; hence, the number of edges on the spine between two consecutive roots must be the height difference between the corresponding subtrees: $d(v_{i+1}, v_i) = h_{i+1} - h_i = 2^i = h_i + 1$, where d is the shortest path distance. Therefore, when we interleave a subtree, $T_{1,i}$, and the spine edges between v_{i+1} and v_i in the linear layout, we can alternately take one vertex from each. This is illustrated in Figure 4. This layout has bandwidth 2, which is the minimum possible.

Let us now consider the width (BFS width from the root) of a level 2 tree, $T_{2,j}$. This width lower bounds BFS width. Since $h = 2^j$, $j = \log h$ where log is base 2. The total number of subtrees is $j + 1$, each contributing one vertex to the widest layer. The spine also

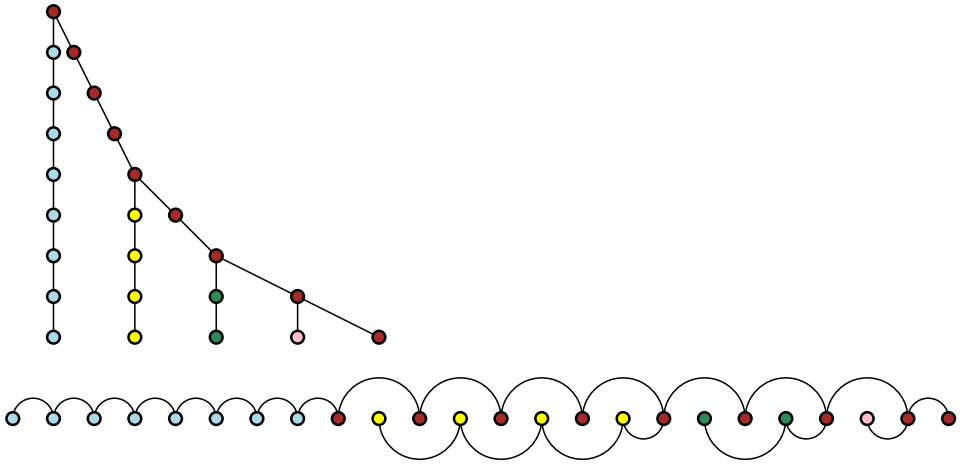


Figure 4 Top: the level 2 tree is constructed by attaching level 1 trees on a spine. The heights of these level 1 subtrees approximately follow a geometric sequence. Bottom: an arc diagram illustrating the linear layout of the level 2 tree. Each edge is a semicircle. The maximum diameter of these semicircles equals the bandwidth of the linear layout. This linear layout achieves the minimum bandwidth 2.

has one vertex at the widest layer. Therefore, the width of $T_{2,j}$ is $w = j + 1 + 1 = \log h + 2$. We want to express this width in terms of the number of vertices, n. Any vertex of this tree is either in a subtree or on the spine. The spine has height h and $h + 1$ vertices. Subtrees have $n_s = \sum_{i=0}^{j}(h_i + 1) = \sum_{i=0}^{j} 2^i = 2h - 1$ vertices. Thus, $n = h + 1 + 2h - 1 = 3h$; hence, we can define a level 2 tree $T_{2,j}$ to have an arbitrarily large number, n, of vertices. Therefore, $T_{2,j}$ has bandwidth $\mathbf{bw}(T_{2,j}) = 2$, and width, $w = j+2$, which implies $\mathbf{bfsw}(T_{2,j}) = \Omega(\log n)$, where $n = 3h = 3 \cdot 2^j$ can be arbitrarily large.

Following the structural pattern we used to build a level 2 tree, to construct a tree at level k, we take trees from level $(k - 1)$ whose heights approximately follow a geometric sequence and attach them to a path, which we again call the **spine**. These level $(k-1)$ trees are called **subtrees**. Their placement on the spine is such that the leaves of these trees have the same depth in the level k tree, thus achieving their width in a single (last) level of a breadth-first layering. To get a bounded bandwidth layout of the level k tree, we interleave vertices from the spine and level $(k - 1)$ subtrees. These constructions are illustrated in Figure 5.

Let us, therefore, consider the bandwidth and BFS width of a level k tree. Suppose the subtree below vertex v_i (excluding v_i) is $T_{k-1,i}$. Given this notational background, let us prove the following theorem, which is one of the two main technical theorems in this paper:

▶ **Theorem 9** (Lower bound on BFS width)**.** *For each non-negative integer, k, and for arbitrarily large values of n, there exists a (level k) tree, T, with n vertices that satisfies:*

$$\mathbf{bw}(T) = O_k(1),$$
$$\mathbf{bfsw}(T) = \Omega(\log^k n),$$

where $O_k(1)$ denotes a number that depends on k but is bounded when k is bounded.

Proof. We construct the level k tree to have an arbitrarily large integer, n, of vertices, by induction, from level $(k - 1)$ trees. To keep track of the tree and subtree sizes, we let $n_{k,j}$ denote the size of the tree, $T_{k,j}$, we are constructing (to roughly have height that is arbitrarily large as a power of j). The base case is the construction given above, which in this notation,

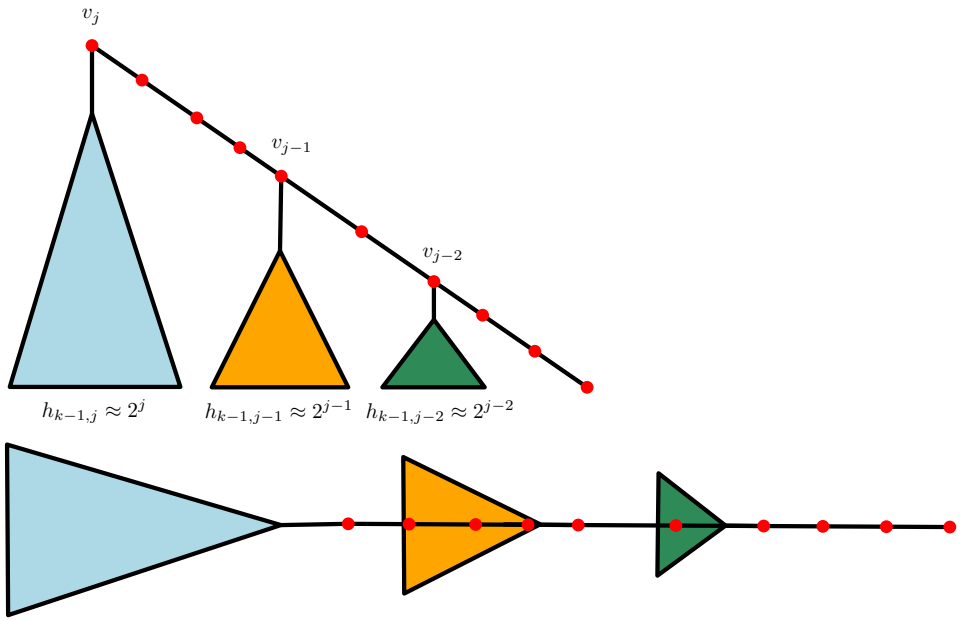


Figure 5 Top: the level k tree is constructed by attaching level $(k-1)$ trees on a spine. The heights of these level $(k-1)$ subtrees approximately follow a geometric sequence. The leaves of these subtrees have the same depth in the level k tree. Bottom: a diagram illustrating the linear layout of the level k tree. This does not necessarily achieve the minimum possible bandwidth. Intuitively, this is the level k tree rooted at the rightmost vertex of the spine. We interleave vertices from the spine and level $(k-1)$ subtrees. Vertices from a level $(k-1)$ subtree are arranged according to their linear layout from the previous level.

gives a level 2 tree with $n_{2,j} = 3h_{2,j}$ vertices and height $h_{2,j} = 2^j$ and to have bandwidth 2 and BFS width at least $d_2 j + 2$, for the constant, $d_2 = 1$. Accordingly, in our analysis below, we focus on the dominant components sufficient for our asymptotic analysis.

Inductive step. Suppose, for $i = 0, 1, 2, \ldots, j$, we have a level $(k-1)$ tree, $T_{k-1,i}$, with height $h_{k-1,i} = 2^i + k - 3$ having width $w_{k-1,i}$ and $n_{k-1,i}$ vertices such that

$$n_{k-1,i} = c_{k-1} \cdot 2^i + p_{k-1}(i), \qquad w_{k-1,i} = d_{k-1} i^{k-2} + q_{k-1}(i), \qquad \mathbf{bw}'(T_{k-1,i}) = O_{k-1}(1),$$

where $\mathbf{bw}'$ is the bandwidth of the linear layout we construct, c_{k-1} and d_{k-1} are constants, and p_{k-1} and q_{k-1} are polynomial functions with degree $k-3$. Let us now construct a level k tree where the height of the leftmost subtree is $h_{k-1,j} = 2^j + k - 3$, so this level k tree has height $h_{k,j} = 2^j + k - 2$. Using Faulhaber's formula (e.g., see Knuth [29]), we can characterize the total number of vertices in all the subtrees as

$$n_s = \sum_{i=0}^{j} (c_{k-1} \cdot 2^i + p_{k-1}(i)) = c_{k-1} \cdot (2^{j+1} - 1) + p_k(j),$$

where p_k is a polynomial function with degree $k-2$. The spine has $h_{k,j} + 1$ vertices. The total number of vertices in this level k tree, therefore, is

$$\begin{aligned} n_{k,j} &= n_s + h_{k,j} + 1 = c_{k-1} \cdot (2^{j+1} - 1) + p_k(j) + 2^j + k - 2 + 1 \\ &= (2c_{k-1} + 1) \cdot 2^j + p'_k(j) = c_k \cdot 2^j + p'_k(j), \end{aligned}$$

where p'_k is a polynomial function with degree $k-2$, and $c_k \coloneqq 2c_{k-1} + 1$. Since $c_2 = 3$, we can solve this recurrence relation to yield $c_k = 2^k - 1$.

Bandwidth. Let us next upper bound the bandwidth, $\mathbf{bw}(T_{k,j})$, of the tree, $T_{k,j}$. We interleave each subtree $T_{k-1,i}$ and the spine path between v_{i+1} and v_i in the linear layout. $T_{k-1,i}$ has height $h_{k-1,i}$ and $n_{k-1,i}$ vertices. The spine path $v_{i+1}v_i$ has length $d(v_{i+1}, v_i) = h_{k-1,i+1} - h_{k-1,i} = 2^i$. We use the linear layout of subtree $T_{k-1,i}$ obtained from the previous level and insert vertices from the spine path. Let $r_i \coloneqq n_{k-1,i}/2^i = c_{k-1} + p_{k-1}(i)/2^i$. We alternately place $\lceil r_i \rceil$ vertices from $T_{k-1,i}$ and one vertex from the spine path $v_{i+1}v_i$, until all vertices from $T_{k-1,i}$ have been placed, in which case we place the remaining vertices from $v_{i+1}v_i$ consecutively. Any edge connecting two vertices from $T_{k-1,i}$ with length equal to $\mathbf{bw}'(T_{k-1,i})$ may have at most $\left\lceil \frac{\mathbf{bw}'(T_{k-1,i})}{\lceil r_i \rceil + 1} \right\rceil$ spine vertices inside it. Any edge connecting two vertices from the spine path $v_{i+1}v_i$ has at most $\lceil r_i \rceil$ vertices from $T_{k-1,i}$ inside it. These are the only two factors that cause the bandwidth to increase from level $(k-1)$ to level k, so the recurrence relation is

$$\mathbf{bw}'(T_{k,j}) \leq \max_{i=0}^{j} \max \left(\mathbf{bw}'(T_{k-1,i}) + \left\lceil \frac{\mathbf{bw}'(T_{k-1,i})}{\lceil r_i \rceil + 1} \right\rceil, \lceil r_i \rceil + 1 \right).$$

Since $\lim_{i \to \infty} r_i = c_{k-1}$, for large values of i, $r_i < c_{k-1} + 1$, and for small values of i, r_i can be upper bounded by another constant. Thus, this level k linear layout has bandwidth $O_k(1)$.

BFS width. BFS width is lower bounded by width, which can be computed using Faulhaber's formula (e.g., see Knuth [29]):

$$w_{k,j} = \sum_{i=0}^{j} (d_{k-1} \cdot i^{k-2} + q_{k-1}(i)) = \frac{d_{k-1}}{k-1} j^{k-1} + q_k(j) = d_k j^{k-1} + q_k(j),$$

where $d_k \coloneqq \frac{d_{k-1}}{k-1}$ and q_k is a polynomial function with degree $k-2$. Since $d_2 = 1$, we can solve this recurrence relation to yield $d_k = \frac{1}{(k-1)!}$.

We have therefore shown that the tree $T_{k,j}$ has height $h_{k,j} = 2^j + k - 3$ and $n_{k,j}$ vertices, with

$$\begin{aligned} n_{k,j} &= c_k \cdot 2^j + p'_k(j), \\ \mathbf{bw}(T_{k,j}) &= O_k(1), \quad \text{and} \\ \mathbf{bfsw}(T_{k,j}) &\geq w_{k,j} = d_k j^{k-1} + q_k(j) = \Omega(\log^{k-1} n_{k,j}). \end{aligned}$$

◀

Thus, a graph with bounded bandwidth can have polylogarithmic BFS width. Also:

▶ **Theorem 10** (Lower bound on minimum BFS width). *For each non-negative integer, k, and for arbitrarily large values of n, there exists a (level k) tree, T, with n vertices that satisfies:*

$$\begin{aligned} \mathbf{bw}(T) &= O_k(1), \\ \mathbf{bfsw}_{\min}(T) &= \Omega(\log^k n), \end{aligned}$$

where $O_k(1)$ denotes a number that depends on k but is bounded when k is bounded.

Proof. A construction similar to Figure 5 can be used to place the root at one end of the linear layout. Taking two reflected copies of the resulting tree, connected at the root, gives a lower bound for minimum BFS width (with different constants than Theorem 9). ◀

▶ **Theorem 11.** *For arbitrarily large values of n, there exists a binary tree, T, with n vertices that satisfies* $\mathbf{bfsw}(T)/\mathbf{bfsw}_{\min}(T) = \Omega(\log n)$.

Proof. The level 2 tree, $T = T_{2,j}$, is originally rooted at the leftmost spine vertex v_j. Consider rooting T at the rightmost spine vertex v_s, as in the bottom of Figure 4. The width of the new tree is 2. Thus,

$$\begin{aligned}
&\mathbf{bfsw}(T, v_j) = j + 2,\\
&\mathbf{bfsw}(T, v_s) = 2,\\
&\frac{\mathbf{bfsw}(T)}{\mathbf{bfsw}_{\min}(T)} \geq \frac{\mathbf{bfsw}(T, v_j)}{\mathbf{bfsw}(T, v_s)} = \frac{j+2}{2} = \frac{\log\left(\frac{n}{3}\right) + 2}{2}, \quad \text{and}\\
&\frac{\mathbf{bfsw}(T)}{\mathbf{bfsw}_{\min}(T)} = \Omega(\log n).
\end{aligned}$$

◀

4 Polylogarithmic BFS Width for Graphs of Bounded Bandwidth

In this section, we show that if bandwidth is bounded, then BFS width is at most polylogarithmic in the number of vertices. The construction in Section 3 shows that this upper bound is tight. The proof follows a structure similar to our lower-bound construction. First, we make several simplifying assumptions with respect to the graph, which we make without loss of generality (wlog). Next, we identify spines and subtrees as in our construction above. Then we use strong induction to prove our bound.

The inductive hypothesis is as follows: for arbitrarily large values of n, any graph on n vertices with bandwidth $\mathbf{bw} \leq k$, for a fixed constant, $k \geq 1$, has BFS width $\mathbf{bfsw} = O\big(\log^{k-1}(n)\big)$. The base case, $k = 1$, is clearly true, since a connected graph with bandwidth 1 is a path. For the induction step, suppose that if any graph, G, on n vertices has bandwidth $\mathbf{bw}(G) \leq k$, for a fixed constant, $k \geq 1$, then $\mathbf{bfsw}(G) = O\big(\log^{k-1}(n)\big)$. To show the induction hypothesis holds for $k+1$, we consider a widest BFS tree of G, and show that there are at most a logarithmic number of subtrees in T and each subtree has width $O\big(\log^{k-1}(n)\big)$. Consequently, when these subtrees are combined, the overall width remains polylogarithmic, with a higher (constant) exponent.

Suppose we have a linear layout of an n-vertex graph, G, that achieves minimum bandwidth, let T be a widest BFS tree of G, and let r be the root of T. We want to show that T has width $O\big(\log^{k}(n)\big)$. Thus, without changing the BFS width, we can delete non-tree edges from G as well as all vertices and edges below the first widest layer of T. This may reduce the number of vertices, but, wlog, let us nevertheless refer to the reduced number of vertices as n and the reduced tree as T, since we are interested in an upper bound in terms of the number of vertices in the original graph. Thus, let us focus our attention on the reduced tree, T, instead of the original graph.

Furthermore, we can make extreme points of the linear layout leaves of T. Suppose this is not the case. Wlog, we consider two scenarios:

1. The leftmost vertex v of the linear layout is the root. If the root only has one child, then we can remove it and make its child the new root. After repeating this step, either the new leftmost vertex is not the root, or it is the root with at least two children. In the latter case, we can take the subtree rooted at one child and flip its linear layout around the root. Thus, extreme points of the linear layout will not be the root.
2. The rightmost vertex v of the linear layout is neither the root nor a leaf. All leaves of the subtree T_v rooted at v are to the left of v in the linear layout. We can flip this subtree around v to move non-root vertices of T_v to the right side of v. After this, we check the new rightmost vertex $w \in T_v$. If it is not a leaf, then $T_w \subset T_v$ and $v \notin T_w$. Since the subtree T_w contains fewer vertices than T_v, each iteration of this process reduces the size of the subtree rooted at the rightmost vertex. Therefore, after repeating this process finitely many times, the rightmost vertex will be a leaf.

After these operations, extreme points of the linear layout are now leaves of T. Note that the width of the tree does not change and the bandwidth of this new ordering cannot be greater than before. The **left spine** is defined as the path from the root to the leftmost vertex, and the **right spine** is defined similarly. All non-spine vertices belong to a **subtree** "hanging off" the spine. In this section, it is more convenient to treat the root vertices of these subtrees as the intersection of subtrees and the spine. Now we upper bound the number of subtrees that contribute to the width of T.

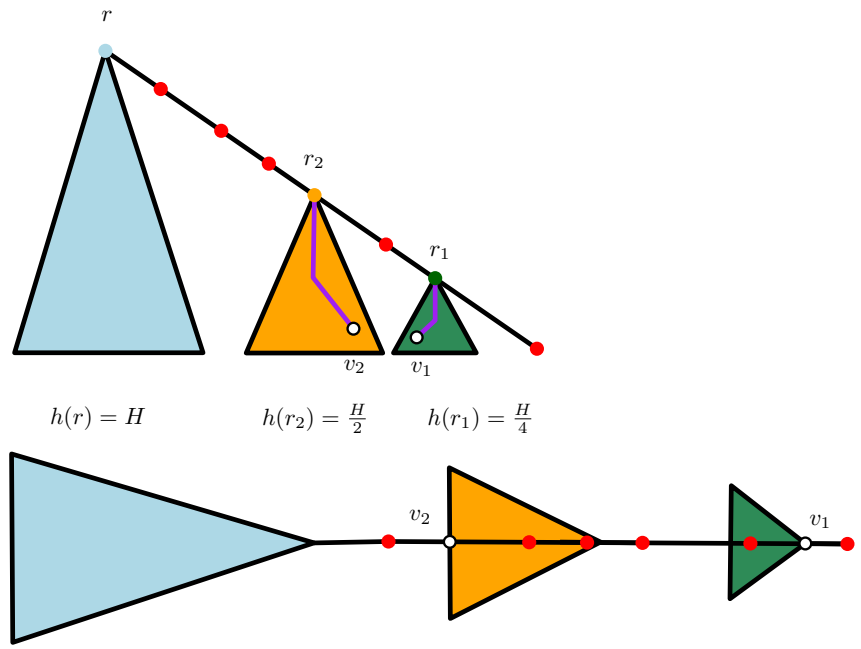


Figure 6 Proof of Lemma 12 illustrated. When $i = 1$, S_1 is the set of all subtrees with height $h(r_s) \in I := [\frac{H}{4}, \frac{H}{2}]$. In this figure, it contains the two subtrees on the right. We want to upper bound the total number of vertices in these subtrees.

▶ **Lemma 12.** *The number of subtrees that have at least one leaf at the widest layer of the tree T is $O(\log n)$.*

Proof. Because of symmetry, we only bound the number of subtrees whose root is on the right spine. Suppose the tree T rooted at r has height H. For integer $i = 0, 1, \ldots, \lfloor \log H \rfloor$, let S_i be the set of all subtrees with height $h(r_s) \in I := [\frac{H}{2^{i+1}}, \frac{H}{2^i}]$ where r_s is the root of a subtree. See Figure 6. We do not need to consider subtrees with height 0.

Now we upper bound the total number of vertices in all subtrees in S_i. Consider the position of these subtrees in the linear layout. Suppose the rightmost vertex is v_1 and it belongs to the subtree rooted at r_1, and the leftmost vertex v_2 belongs to the subtree rooted at r_2. Recall that d denotes the shortest path distance in the graph. For any root r_s of a subtree in S_i, $d(r, r_s) = H - h(r_s) \in H - I := [H - \frac{H}{2^i}, H - \frac{H}{2^{i+1}}]$. The shortest paths from r to all such root vertices are part of the right spine. Thus,

$$
\begin{aligned}
&d(r_1, r_2) \le |H - I| = |I| \\
&d(v_1, v_2) \le d(v_1, r_1) + d(r_1, r_2) + d(r_2, v_2) \le h(v_1) + |I| + h(v_2) \\
&\qquad \le \frac{H}{2^i} + \left(\frac{H}{2^i} - \frac{H}{2^{i+1}}\right) + \frac{H}{2^i} = \frac{3H}{2^i} - \frac{H}{2^{i+1}} = \frac{5H}{2^{i+1}}.
\end{aligned}
$$

Because of the bounded bandwidth, the number of vertices between v_1 and v_2 in the linear layout is at most $\frac{5H}{2^{i+1}}(\mathbf{bw})$.

Each subtree with height in I must have at least $\frac{H}{2^{i+1}}$ vertices to reach the minimum height. Consequently, the total number of subtrees with height in any given interval I is at most a constant and this completes the proof. ◀

Next, we prove that removing spine vertices from the linear layout reduces the bandwidth of subtrees. Since the roots of these subtrees also belong to the spine, each subtree may decompose into a forest. Given that bounded bandwidth implies bounded maximum degree, the number of trees in each resulting forest is bounded. If these trees have smaller bandwidth, we can apply the inductive hypothesis to show that they have width $O\big(\log^{k-1}(n)\big)$. Thus, the original subtree also has width $O\big(\log^{k-1}(n)\big)$. Thus, the following lemma is essential to the induction:

▶ **Lemma 13.** *If the tree T has bandwidth* **bw***, then each tree in the forests obtained after removing the spine has bandwidth at most* $\mathbf{bw} - 1$.

Proof. The proof outline is as follows. By definition, the extreme points of the original linear layout are spine vertices and thus subtrees with one vertex. We denote them as v_l and v_r respectively. Consider any subtree T_v rooted at $v \neq v_l, v_r$. After removing the spine vertices, the linear layout for T naturally defines a linear layout for $T_v \setminus v$ by preserving the relative position of its vertices. We show that the maximum length of any edge of $T_v \setminus v$ will become less than **bw**.

In the linear layout for T, the length of any edge is at most **bw**. If an edge has length equal to **bw**, we say that this edge is saturated. Let S be the set of all saturated edges of $T_v \setminus v$. If this is an empty set, the proof is complete. Otherwise, suppose there is one edge $e \in S$ with no spine vertex inside it. v_l and v_r are connected through edges of the spine and one of these edges must enclose the two endpoints of e. The length of e is **bw**, so the length of this edge is greater than **bw**, and this is a contradiction. Thus, there is a spine vertex inside every saturated edge of $T_v \setminus v$. After removing the spine vertices, all edges of $T_v \setminus v$ have length at most $\mathbf{bw} - 1$. $T_v \setminus v$ may be a forest, so this upper bounds the bandwidth of each connected component. ◀

With these lemmas, we can apply strong induction on bandwidth, which gives us:

▶ **Theorem 14** (Upper bound on BFS width)**.** *For any constant, $k \geq 1$, and n-vertex graph, G, if* $\mathbf{bw}(G) = k$*, then* $\mathbf{bfsw}(G) = O\big(\log^{k-1}(n)\big)$.

5 Applications

In this section, we prove the application results claimed in the introduction.

5.1 Numerical Linear Algebra

For the purposes of this section, the only facts we need about the Cuthill–McKee and reverse Cuthill–McKee algorithms are:

- The bandwidth of a reordered matrix PAP^T, where A is any symmetric matrix and P is any permutation matrix, is the same as the bandwidth of the linear layout of the graph of nonzeros of A obtained by laying out the vertices of this graph in the order given by permutation P.
- Both the Cuthill–McKee and reverse Cuthill–McKee algorithms, applied to a matrix A choose their orderings (interpreted as linear layouts of graphs) by the breadth-first layer of the graph of nonzeros of A, and then carefully permute the vertices within each layer.

We can directly relate the performance of these algorithms to BFS width:

▶ **Theorem 15.** *Let G be the graph of nonzeros of a symmetric matrix A, and let v be an arbitrary vertex of G. Then the Cuthill–McKee and reverse Cuthill–McKee algorithms, using a breadth-first layering rooted at v, produce layouts of bandwidth $\Theta\big(\mathbf{bfsw}(G,v)\big)$.*

Proof. Let u be a vertex in the widest layer of the breadth-first layering of G, chosen among the vertices of this layer to be the one that the Cuthill–McKee or reverse Cuthill–McKee algorithm places farthest from the previous layer. Then the positions of u and its BFS ancestor differ by at least $\mathbf{bfsw}(G,v)$, so the bandwidth of the layout is at least $\mathbf{bfsw}(G,v)$. Every edge in G extends over at most two consecutive layers, and has endpoints whose positions differ by at most $2\mathbf{bfsw}(G,v)-1$, so the bandwidth of the layout is at most $2\mathbf{bfsw}(G,v)-1$. ◀

By combining Theorem 15 with Theorem 14, we have:

▶ **Theorem 16.** *If a symmetric matrix A can be reordered to have bounded bandwidth, the Cuthill–McKee and reverse Cuthill–McKee algorithms will produce a reordered matrix PAP^T of at most polylogarithmic bandwidth.*

Note that this theorem holds even if we do not know the optimal bandwidth of the matrix A. Moreover, the bound of Theorem 16 is is tight, up to the dependence of the polylogarithmic bound on the bandwidth; by combining Theorem 15 with Theorem 10, we have:

▶ **Theorem 17.** *For any k, there exist $n \times n$ symmetric matrices A of bounded bandwidth (for arbitrarily large n) for which the Cuthill–McKee and reverse Cuthill–McKee algorithms produce reordered matrices PAP^T of bandwidth $\Omega(\log^k n)$.*

Theorems 16 and 17 provide the first non-trivial deterministic analyses for the Cuthill-McKee algorithm and its reversal.

5.2 Graph Reconstruction

To reconstruct graphs of low bandwidth or low BFS width, we apply Algorithm 1.

Algorithm 1 Reconstructing graphs with low BFS width.

```
v ← an arbitrary vertex in V                          ▷ this is the root of the BFS tree
for u ∈ V \ {v} do
    Query(v, u)                                       ▷ this is a distance query
Ê ← {{a, b} : {a, b} ⊆ V \ {v} ∧ |d(v, a) − d(v, b)| ≤ 1}
for {a, b} ∈ Ê do
    Query(a, b)
return {{u, v} : u ∈ V ∧ d(v, u) = 1} ∪ {{a, b} : {a, b} ∈ Ê ∧ d(a, b) = 1}
```

Less formally, Algorithm 1 first picks a vertex v as the root of the BFS tree. Then, it obtains distances from this root to every other vertex to determine their layer number. It constructs the set of all vertex pairs from the same layer or in two consecutive layers, and queries each such pair. Recall that in the proof of Theorem 5, non-tree edges connect two vertices whose layer numbers differ by at most one, so each pair of adjacent vertices in G must belong to the set of queried pairs. The algorithm then returns the set of all pairs for which a

query found two vertices to be at distance one. [4] used a similar approach to reconstruct trees and k-chordal graphs. They divided the graph into BFS layers and reconstructed edges in the same layer and in two consecutive layers, but querying each pair would incur a high query complexity for those graph classes.

▶ **Theorem 18.** *For any graph G with n vertices and BFS width B, Algorithm 1 reconstructs G with deterministic query complexity $O(nB)$.*

Proof. It takes $n-1$ queries to find the distances from the chosen root v and partition the vertices into layers. Then, the algorithm makes at most $3B-1$ queries involving each remaining vertex, between it and at most $3B-1$ other vertices in its layer and the two adjacent layers. This double-counts the queries, so the total number of queries is at most $n-1+(n-1)(3B-1)/2 = O(nB)$. ◀

▶ **Corollary 19.** *For any constant bound b on the bandwidth of graphs, Algorithm 1 reconstructs n-vertex graphs of bandwidth $\leq b$ with deterministic query complexity $\tilde{O}(n)$.*

Proof. This follows immediately from Theorem 18 and from the relation between bandwidth and BFS width of Theorem 14. ◀

5.3 Graph Drawing

In our graph drawing application, we specifically focus on arc diagrams, which were used to study the crossing number problem [31] and visualize structure in strings [36]. In an arc diagram, vertices of a graph are positioned along a straight (e.g., horizontal) line, and edges are semicircles drawn on either side of the line [31], joining pairs of points (possibly using straight-line segments for consecutive points). Examples are shown above in Figures 2 and 4.

To analyze such diagrams, we constrain the vertices to be placed at consecutive integer coordinates. Thus, the horizontal line is a linear layout of the vertices. The drawing fits within a bounding box $n-1$ units wide and $b/2$ units tall, where n is the number of vertices in the graph and b is the bandwidth of the given layout.

We immediately obtain the following result:

▶ **Theorem 20.** *Suppose that a given graph G has n vertices and has an arc diagram of bounded height h. Construct an arc diagram for G by ordering the vertices by their breadth-first layer numbers, starting from an arbitrary root vertex. Then if h is bounded by a constant, the height of the arc diagram that we construct will be bounded by $O\big(\text{polylog}(n)\big)$.*

References

1 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Mapping networks via parallel kth-hop traceroute queries. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:21, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.STACS.2022.4.

2 Ramtin Afshar, Michael T. Goodrich, and Evrim Ozel. Efficient exact learning algorithms for road networks and other graphs with bounded clustering degrees. In Christian Schulz and Bora Uçar, editors, *20th International Symposium on Experimental Algorithms (SEA 2022)*, volume 233 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.SEA.2022.9.

3 Mugurel Ionut Andreica. A dynamic programming framework for combinatorial optimization problems on graphs with bounded pathwidth, 2012. arXiv:0806.0840.

4 Paul Bastide and Carla Groenland. Optimal distance query reconstruction for graphs without long induced cycles, October 2024. arXiv:2306.05979 [cs]. doi:10.48550/arXiv.2306.05979.

5 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

6 Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995. doi:10.1006/jagm.1995.1009.

7 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.

8 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, May 2008. doi:10.1093/comjnl/bxm037.

9 Michael Burch, Kiet Bennema Ten Brinke, Adrien Castella, Ghassen Karray Sebastiaan Peters, Vasil Shteriyanov, and Rinse Vlasvinkel. Dynamic graph exploration by interactively linked node-link diagrams and matrix visualizations. *Visual Computing for Industry, Biomedicine, and Art*, 4:1–14, 2021. doi:10.1186/s42492-021-00088-8.

10 Julia Böttcher, Klaas P. Pruessmann, Anusch Taraz, and Andreas Würfl. Bandwidth, expansion, treewidth, separators and universality for bounded-degree graphs. *European Journal of Combinatorics*, 31(5):1217–1227, July 2010. doi:10.1016/j.ejc.2009.10.010.

11 Jean Cardinal, Michael Hoffmann, Vincent Kusters, Csaba D. Tóth, and Manuel Wettstein. Arc diagrams, flip distances, and Hamiltonian triangulations. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 30 of *LIPIcs*, pages 197–210. Schloss Dagstuhl, 2015. doi:10.4230/LIPIcs.STACS.2015.197.

12 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022.

13 E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, ACM '69, pages 157–172, New York, NY, USA, August 1969. Association for Computing Machinery. doi:10.1145/800195.805928.

14 Alberto Debiasi, Bruno Simoes, and Raffaele De Amicis. Schematization of node-link diagrams and drawing techniques for geo-referenced networks. In *International Conference on Cyberworlds (CW)*, pages 34–41, 2015. doi:10.1109/CW.2015.68.

15 Chandan Dubey, Uriel Feige, and Walter Unger. Hardness results for approximating the bandwidth. *Journal of Computer and System Sciences*, 77(1):62–90, January 2011. doi:10.1016/j.jcss.2010.06.006.

16 John Dunagan and Santosh Vempala. On euclidean embeddings and bandwidth minimization. In Michel Goemans, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, pages 229–240, Berlin, Heidelberg, 2001. Springer. sentenceCase:1. doi:10.1007/3-540-44666-4_26.

17 Sherif El-Basil. Applications of caterpillar trees in chemistry and physics. *Journal of Mathematical Chemistry*, 1(2):153–174, July 1987. doi:10.1007/BF01205666.

18 Uriel Feige. Coping with the NP-hardness of the graph bandwidth problem. In Magnús M. Halldórsson, editor, *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, volume 1851 of *Lecture Notes in Computer Science*, pages 10–19. Springer, 2000. doi:10.1007/3-540-44985-X_2.

19 Uriel Feige, Mohammadtaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In *Proc. 37th ACM Symposium on Theory of Computing (STOC 2005)*, pages 563–572, 2005. doi:10.1145/1060590.1060674.

20 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

21 Alan George and Joseph W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, 1981.

22 Michael T. Goodrich and Roberto Tamassia. *Algorithm Design and Applications*. Wiley, 2015.

23 Hermann Gruber. On balanced separators, treewidth, and cycle rank. *J. Combinatorics*, 3(4):669–681, 2012. doi:10.4310/JOC.2012.v3.n4.a5.

24 Anupam Gupta. Improved bandwidth approximation for trees and chordal graphs. *Journal of Algorithms*, 40(1):24–36, July 2001. doi:10.1006/jagm.2000.1118.

25 Eitan M. Gurari and Ivan Hal Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the MINCUT linear arrangement problem. *Journal of Algorithms*, 5(4):531–546, 1984. doi:10.1016/0196-6774(84)90006-3.

26 Frank Harary and Allen J. Schwenk. The number of caterpillars. *Discrete Mathematics*, 6(4):359–365, 1973. doi:10.1016/0012-365X(73)90067-8.

27 Sampath Kannan, Claire Mathieu, and Hang Zhou. Graph reconstruction and verification. *ACM Transactions on Algorithms*, 14(4):40:1–40:30, August 2018. doi:10.1145/3199606.

28 Haim Kaplan and Ron Shamir. Pathwidth, bandwidth, and completion problems to proper interval graphs with small cliques. *SIAM J. Comput.*, 25(3):540–561, 1996. doi:10.1137/S0097539793258143.

29 Donald E. Knuth. Johann Faulhaber and sums of powers. *Mathematics of Computation*, 61(203):277–294, 1993. doi:10.2307/2152953.

30 Ales Komarek, Jakub Pavlik, and Vladimir Sobeslav. Network visualization survey. In Manuel Núñez, Ngoc Thanh Nguyen, David Camacho, and Bogdan Trawiński, editors, *Computational Collective Intelligence: 7th International Conference, ICCCI 2015, Madrid, Spain, September 21–23, 2015, Proceedings, Part II*, volume 9330 of *Lecture Notes in Computer Science*, pages 275–284, Cham, 2015. Springer. doi:10.1007/978-3-319-24306-1_27.

31 S. Masuda, K. Nakajima, T. Kashiwabara, and T. Fujisawa. Crossing minimization in linear embeddings of graphs. *IEEE Transactions on Computers*, 39(1):124–127, January 1990. doi:10.1109/12.46286.

32 Claire Mathieu and Hang Zhou. A simple algorithm for graph reconstruction. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.ESA.2021.68.

33 Andrzej Proskurowski and Jan Arne Telle. Classes of graphs with restricted interval models. *Discrete Mathematics & Theoretical Computer Science*, 3(4):167–176, 1999. doi:10.46298/dmtcs.263.

34 Ilya Safro, Dorit Ron, and Achi Brandt. Multilevel algorithms for linear ordering problems. *ACM J. Exp. Algorithmics*, 13:Article 1.4, 2009. doi:10.1145/1412228.1412232.

35 Jonathan S. Turner. On the probable performance of heuristics for bandwidth minimization. *SIAM Journal on Computing*, 15(2):561–580, May 1986. doi:10.1137/0215041.

36 M. Wattenberg. Arc diagrams: visualizing structure in strings. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, pages 110–116, October 2002. ISSN: 1522-404X. doi:10.1109/INFVIS.2002.1173155.

Improved Parallel Derandomization via Finite Automata with Applications

Jeff Giliberti
University of Maryland, College Park, MD, USA

David G. Harris
University of Maryland, College Park, MD, USA

Abstract

A central approach to algorithmic derandomization is the construction of small-support probability distributions that "fool" randomized algorithms, often enabling efficient parallel (NC) implementations. An abstraction of this idea is fooling polynomial-space statistical tests computed via finite automata [Sivakumar STOC'02]; this encompasses a wide range of properties including k-wise independence and sums of random variables.

We present new parallel algorithms to fool finite-state automata, with significantly reduced processor complexity. Briefly, our approach is to iteratively sparsify distributions using a work-efficient lattice rounding routine and maintain accuracy by tracking an aggregate weighted error that is determined by the Lipschitz value of the statistical tests being fooled.

We illustrate with improved applications to the Gale-Berlekamp Switching Game and to approximate MAX-CUT via SDP rounding. These involve further several optimizations, such as the truncation of the state space of the automata and FFT-based convolutions to compute transition probabilities efficiently.

2012 ACM Subject Classification Theory of computation → Parallel algorithms; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Parallel Algorithms, Derandomization, MAX-CUT, Gale-Berlekamp Switching Game

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.70

Related Version *Full Version*: `https://arxiv.org/abs/2411.18028`

Funding *Jeff Giliberti*: JG gratefully acknowledges financial support by the Fulbright U.S. Graduate Student Program, sponsored by the U.S. Department of State and the Italian-American Fulbright Commission. This content does not necessarily represent the views of the Program.

Acknowledgements Thanks to Mohsen Ghaffari and Christoph Grunau for explaining the works [19] and [20].

1 Introduction

A fundamental problem in the theory of computation is to *derandomize* existing randomized algorithms. Such randomized algorithms typically use a large number of independent random bits. One main genre of derandomization is the construction of a probability distribution which is much smaller (of polynomial instead of exponential size), to "fool" the randomized algorithm. That is, the behavior of relevant statistics should be similar when presented with fully-independent random bits vs. bits drawn from a small, carefully-constructed correlated distribution. This probability space can be searched exhaustively, and in parallel, to find a specific input with desired properties.


© Jeff Giliberti and David G. Harris;
licensed under Creative Commons License CC-BY 4.0



A simple and popular example of this technique comes from a probability space with k-wise independence, for constant k [2, 30, 32, 35]. This preserves basic statistical properties such as means and variances [7, 42]. There are many other constructions for more-advanced properties, e.g., near-k-wise independence [3, 5, 14, 17, 38], probability spaces fooling halfspaces or polytopes [33, 41, 23, 22], ordered branching programs [12, 15, 37, 27, 40] etc.

A significant abstraction of these methods is the construction of probability spaces that fool statistical properties (also called "tests") computed by finite automata [10, 24, 31, 36, 39, 40, 43]. Such tests are ubiquitous in randomized algorithm analysis, encompassing k-wise independence, sums of random variables, and many other properties. For example, they are used in deterministic parallel algorithms for finding balanced independent sets in graphs [26], for applications of the Lovász Local Lemma in combinatorics [25], for covering and packing integer programs [6, 44], for undirected connectivity problems [40], and more.

1.1 Our contribution

We describe new parallel algorithms to fool automata, with significantly reduced processor complexity. The analysis is also simplified because we can cleanly separate out problem-specific optimizations from the general lattice discrepancy problems that are at the core of the algorithm. A summary of our algorithm performance is as follows:

▶ **Theorem 1** (Simplified). *Consider a probability space Ω on n binary random variables and a collection of ℓ statistical tests over Ω, each with η possible states and Lipschitz value λ.*

There is a deterministic algorithm to generate a distribution D of size $\varepsilon^{-2}\operatorname{polylog}(n, \eta, \ell, 1/\varepsilon)$ to simultaneously "fool" all statistical tests to absolute weighted error $\lambda\varepsilon$. *It uses $\tilde{O}(n\ell\eta\varepsilon^{-2} + n\ell\eta^{\omega})$ processors and polylogarithmic time, where ω is the exponent of matrix multiplication.*

By way of comparison, the algorithm of [24] would require roughly $O(n^3\ell^2\eta^4\varepsilon^{-2})$ processors to construct a distribution of size $O(\ell\eta^2\epsilon^{-2})$. Our full results are more general, easily allowing for non-binary alphabets, or more complex automata types; see Theorem 5 for details. We emphasize that the algorithm is still typically more efficient even for applications with no particular Lipschitz bounds.

We illustrate this framework through two prototypical applications: the Gale-Berlekamp Switching Game and SDP rounding for approximate MAX-CUT.

▶ **Theorem 2** (Gale-Berlekamp Switching Game). *Given an $n \times n$ matrix A, there is a deterministic parallel algorithm using $\tilde{O}(n^{3.5})$ processors and polylogarithmic time to find $x, y \in \{-1, +1\}^n$ satisfying $\sum_{i,j} A_{i,j}x_iy_j \geq (\sqrt{2/\pi} - o(1))n^{3/2}$.*

It is interesting that the Gale-Berlekamp analysis revolves around *anti-concentration* bounds, which are precisely the opposite of discrepancy-minimization. Our algorithm for this problem beats the $n^{5+o(1)}$ complexity of the optimized algorithm of [24].

▶ **Theorem 3** (MAX-CUT Approximation). *Let $\varepsilon > 0$ be an arbitrary constant. Given an n-vertex m-edge graph $G(V, E)$, there is a deterministic parallel algorithm using $\tilde{O}(mn^3)$ processors and polylogarithmic time to find an $\alpha(1 - \varepsilon)$ approximate MAX-CUT of G, where $\alpha \approx 0.878$ is the Goemans-Williamson approximation constant [21].*

The SDP relaxation of MAX-CUT can be approximated in polylogarithmic time and near-linear work by [1], so we will be concerned with rounding the SDP solution. A rounding procedure was presented in [43], which however required a very large number of processors. We drastically improve it, getting closer to the *sequential* $\tilde{O}(n^3)$ runtime of [9].

The complexity bounds in Theorems 2 and 3 do not depend on fast matrix multiplication; the runtimes are valid even using naive multiplication ($\omega = 3$). Although thy are still not fully work-efficient, this makes progress toward practical deterministic parallel algorithms.

In a broader perspective, our construction differ in various aspects from classical derandomization via (sequential) PRGs. While our automata framework does not provide a PRG, it seems better suited to efficient parallel derandomization. It is known that, by connections to Boolean circuits [11], a PRG for log space-bounded computation can be simulated in PRAM in polylog time with a polynomial number of processors. However, relying on the equivalence between PRAM and Boolean circuits leads to very high processors complexities (e.g., Nisan's log space-bounded PRG uses at least $\Omega(n^{45})$ processors [39]). Therefore, using known PRGs constructions would either require high polynomials (if logspace bounded) or not parallelize well. In addition to providing fast parallelization, our construction returns a distribution with seed length $O(\log 1/\varepsilon + \log\log(n, \eta, \ell)$, which is line with known PRGs.

1.2 Technical approach

Our work follows the same general outline as previous algorithms for fooling automata, built from two main subroutines Reduce and Fool. The Fool algorithm (Section 5) recursively invokes Reduce (Section 4) to construct a fooling distribution, beginning with single-step automaton transitions and progressively scaling up to multi-step distributions. After $\log_2 n$ levels, the final distribution effectively fools n-step walks in the automata.

We introduce several novelties such as a work efficient Reduce subroutine and an analysis of Fool based on a different measure of error. Here, we attempt here to provide some high-level intuition on the technical content of this paper.

Distribution error tracking. Prior work on automata fooling [24, 36] considered a notion of unweighted absolute error, where the goal was to fool all pairs of (start, end) states. We replace this with an *aggregated* notion: the weight of a final state corresponds to the value of the associated final outcome (e.g., the final count for a counter automaton). The value of an intermediate state is then the expected weight of the final state under a random suffix. This is similar to a scheme used in the context of pseudorandom generators and branching programs [12, 15], where it is taken advantage of sub-maximal influence on the final expected value. The processor complexity is determined by the *Lipschitz value* of each state: how much the expected final weight changes with a single step of the automaton.

REDUCE subroutine. This subroutine takes as input a distribution $E = D^1 \times D^2$ over automata walks (drivestreams) with a weight function w; it outputs a refined distribution D with significantly smaller support while maintaining a weighted measure of closeness to E. This leverages the connection between automata fooling and lattice approximation noted by [36] and applies the work-efficient lattice approximation of [19]. Our key approach to reduce the processor complexity is to sparsify E – without materializing it – in $\log_2 |E|$ steps.

In each step, we model the problem of rounding the i^{th} bit of the elements in D as a lattice-discrepancy problem, which is then solved by invoking [19]. This step resembles the PRG of INW [28]; there, one replaces the concatenation of two independent random walks with correlated random walks, using the derandomized square instead of the algorithm of [19].

Further optimizations. In both of these applications, and many others, the relevant automata are counters which track the running sum of certain statistics. Instead of keeping track of these exactly, we truncate the counters to within a few standard deviations of their means.

This reduces the state space of the automata by roughly a square-root factor. Doing this while also preserving relevant Lipschitz properties of the automata is technically challenging and requires significant analysis. This optimization will be discussed in Section 6.

The approximate MAX-CUT application (Section 7.2) requires a few additional problem-specific optimizations and arguments. In particular, to simulate the SDP rounding procedure efficiently, we (i) work with discretized truncated Gaussians and quantized counters, (ii) use a pessimistic estimator with a better Lipschitz constant, and (iii) we compute the transition matrices via FFT's to exploit their convolution structure. We believe that these optimizations may generalize to other (SDP) rounding procedures and other settings as well.

2 Preliminaries

For an input of size N, our goal is to develop deterministic PRAM algorithms with $\text{poly}(N)$ processor complexity and $\text{polylog}(N)$ runtime. Unless stated otherwise, we assume that all relevant algorithms run in *deterministic polylogarithmic time* as a function of their processor and input complexities. For processor complexity and other parameters, we use $\tilde{O}$ notation: we say that $f(x) \leq \tilde{O}(g(x))$ if $f(x) \leq g(x) \cdot \text{polylog}(N, g(x))$ processors, where N is the size of all algorithm inputs. Throughout, we use log for logarithm in base e and lg for base 2. We write $\lceil x \rfloor$ for rounding to the nearest integer.

Throughout, proofs that are omitted (or sketched) appear in the full version.

2.1 Basic definitions for automata

The underlying probability space Ω is defined by drawing a sequence $\vec{r} = (r_0, \ldots, r_{n-1})$, where each r_t is independently drawn from a distribution Ω_t over an arbitrary alphabet. We consider an automaton F with a state space S. At each timestep $t = 0, 1, \ldots, n-1$, the automaton in state $s \in S$ receives an input r_t and transitions to state $F(r_t, s)$.

For times t, t', we define $\Omega_{t,t'} = \Omega_t \times \Omega_{t+1} \times \cdots \times \Omega_{t'-1}$. In this context, we call $h = t' - t$ the *horizon* and the pair (t, h) as the *window*. We refer to a vector $\vec{r} = (r_t, r_{t+1}, \ldots, r_{t'-1}) \in \Omega_{t,t'}$ as a *drivestream*. We define $F(\vec{r}, s)$ to be the result of transiting from time t to t' under $\vec{r}$. We write $\vec{r} \in \Omega_{t,t'}$ to indicate that each $r_j : j = t, \ldots, t'-1$ has non-zero probability in Ω_j.

For a distribution D on drivestreams, we denote the *transition matrix* as T_D, i.e. $T(s, s')$ is the probability of transiting from state s to s' for a drivestream $\vec{r} \sim D$. For brevity, we also write $T_{t,t'} := T_{\Omega_{t,t'}}$ and, for any weight function $w : S \to \mathbb{R}$, we also define

$$T_D(s, w) = \sum_{s' \in S} T_D(s, s') w(s').$$

Throughout, we define $\eta = |S|$ and $\sigma = \sum_{t=0}^{n-1} |\Omega_t|$ where $|\Omega_t|$ is the size of Ω_t.

Our goal is to find a polynomial-size distribution D on drivestreams that "fools" the automaton. That is, the behavior of the automaton given a drivestream $r \sim D$ should be similar to its behavior when given a drivestream $r \sim \Omega$. We will follow a strategy of [12] and measure error via a weight function $W : S \to \mathbb{R}$ over final states. It will be necessary to measure the "smoothness" of W as a function of the drivestream values. Formally, we consider two, closely related, notions:

▶ **Definition 4** (Lipschitz value/confusion of a weight functions). *Let $W : S \to \mathbb{R}$ be a weight function for automaton F.*

- *The* Lipschitz value *at state s and time t is defined by*

$$\mathfrak{L}(s, t) = \max_{\vec{r}_1, \vec{r}_2} \left| W(F(\vec{r}^1, s)) - W(F(\vec{r}^2, s)) \right|$$

where the maximum is over $\vec{r}^1, \vec{r}^2 \in \Omega_{t,n}$ which differ in only the t^{th} coordinate.

- *The* confusion *at state s and time t is defined by*

$$\mathfrak{C}(s,t) = \max_{r_1,r_2\in\Omega_t} \left|T_{t,n}(F(r_1,s),W) - T_{t,n}(F(r_2,s),W)\right|.$$

Colloquially, these are the maximum change to *actual weight* or the *expected weight* of the final state due to changing drivestream entry t. Note that $\mathfrak{C}(s,t) \le \mathfrak{L}(s,t)$.

We define the *total variability* $\mathfrak{V}(s)$ of a starting state $s \in S$ as:

$$\mathfrak{V}(s) := \sum_{t=0}^{n-1} \max_{\vec{r}\in\Omega_{0,t}} \mathfrak{C}(F(\vec{r},s),t)$$

With this rather technical definition, we can state our algorithm concretely:

▶ **Theorem 5.** *The algorithm* Fool *takes as input a parameter $\varepsilon > 0$, and produces a distribution D with*

$$|T_D(s,W) - T_\Omega(s,W)| \le \varepsilon\mathfrak{V}(s) \qquad \text{for all states } s.$$

The distribution D has size $\varepsilon^{-2}\operatorname{polylog}(n,\eta,\sigma,1/\varepsilon)$. The cost of Fool *is $\tilde{O}\big(n\eta/\varepsilon^2 + \eta\sigma\big)$ processors, plus the cost of computing the expectations $T_{t,n}(s,W)$ for states $s \in S$ and times $t = 0,\dots,n$ (more details on this step later.)*

2.2 Lattice approximation

The problem of automata fooling is closely linked to lattice approximation, defined as follows:

▶ **Definition 6** (Lattice Approximation Problem (LAP))**.** *Given an $m \times n$ matrix A and fractional vector $\vec{u} \in [0,1]^n$, the objective is to compute an integral vector $\vec{v} \in \{0,1\}^n$ to minimize the discrepancies $D_k = |\sum_{j=1}^n A_{kj}(u_j - v_j)|$.*

Intuitively, this models the process of "rounding" each random bit (represented by u_j) to a static zero or one (represented by v_j). The work [19] provides a parallel algorithm with near-optimal discrepancy as well as complexity; we summarize it as follows:

▶ **Theorem 7** (Theorem 1.3 of [19])**.** *Suppose that $A_{kj} \in [0,1]$ for all k,j. There is a deterministic parallel algorithm for the LAP with $D_k \le O(\sqrt{\mu_k \log m} + \log m)$ for all k, where $\mu_k = \sum_{j=1}^n A_{kj}u_j$. The algorithm uses $\tilde{O}(n+m+nnz(A))$ processors, where $nnz(A)$ denotes the number of non-zero entries in the matrix A.*

It will be convenient to allow for the discrepancy matrix A to take arbitrary real values.

▶ **Proposition 8.** *Let $\Delta_k = \max_j |A_{kj}|$ for each row k. There is a deterministic parallel algorithm for the LAP where $D_k \le O(\sqrt{\Delta_k\tilde{\mu}_k \log m} + \Delta_k \log m) \le O(\Delta_k(\sqrt{n\log m} + \log m))$ for all k, where $\tilde{\mu}_k = \sum_{j=1}^n |A_{kj}|u_j$. The algorithm uses $\tilde{O}(n+m+nnz(A))$ processors.*

Proof. Construct a $2m \times n$ matrix $\tilde{A}$, where for each row $k = 0,\dots,m-1$ of A, the matrix $\tilde{A}$ has $\tilde{A}_{2k,j} = \max\{0, \frac{A_{kj}}{\Delta_k}\}$ and $\tilde{A}_{2k+1,j} = \max\{0, \frac{-A_{kj}}{\Delta_k}\}$. Then apply Theorem 7 to $\tilde{A}$. ◀

3 Overview of algorithms and data structures

The subroutine REDUCE (Section 4) is the technical core of the automata-fooling construction: given an input distribution E over drivestreams and a weight function w, it returns a distribution D which is close to E (measured in terms of w), but which has much smaller support. Importantly, through the use of appropriate data structures, the cost of REDUCE may be significantly less than the total size of E itself. Later, the final algorithm FOOL (Section 5) will be defined by repeated calls to REDUCE over successively larger time-horizons.

Concretely, we store each distribution D over window (t, h) as an array of drivestreams $D[0], \ldots, D[\ell - 1] \in \Omega_{t,t+h}$ with associated probabilities $p_D(0), \ldots, p_D(\ell - 1)$. Here $\ell = |D|$ is the *size* of D. By adding dummy zero entries, we can assume that $|D|$ is a power of two.

For a bitstring b of length at most $\lg |D|$, we denote by $D[b*]$ the induced distribution consisting of all the drivestreams in D whose indices start with b. So $D[b*] = D[b0*] \cup D[b1*]$, where $b0$ and $b1$ refer to concatenating a 1 or 0 bit to b. Similarly, $p_D(b*) = \sum_{a \in D[b*]} p_D(a)$.

We define the *Prediction Problem* for a distribution D and a weight function $w : S \to \mathbb{R}$ as follows: we need to produce a data structure $\mathcal{Q}(D, w)$, which can answer the following two types of queries: (i) given any bitstring b, return the value $p_D(b*)$; (ii) given (b, s) where b is a bitstring and s is a state, return the value $T_{D[b*]}(s, w)$. Each query should take polylogarithmic time and processors. In either case, b can take on any length $\ell \leq \lg |D|$.

▶ **Observation 9.** *Let D be a distribution on window (t, h). The Prediction Problem for D and a weight function w can be solved with $\tilde{O}(|D|h\eta)$ processors.*

The most important case is for a Cartesian product, which will be used in Section 5. Critically, we can use the structure within a distribution to avoid materializing it explicitly.

▶ **Proposition 10.** *Given distributions D^1, D^2 on windows (t_1, h) and $(t_1 + h, h)$ respectively, the Prediction Problem for distribution $E = D^1 \times D^2$ and any weight function w can be solved with $\tilde{O}(\eta h(|D^1| + |D^2|))$ processors.*

4 The REDUCE Algorithm

To reiterate, the goal of REDUCE (Algorithm 1) is to take an input distribution E and weight function w, and produce a smaller distribution D which is close to it. Note that here w will not be the given weight function over final states W.

For intuition, consider the following process. Draw m elements $D[0], \ldots, D[m - 1]$ randomly and independently with replacement from the support of the distribution E, wherein $D[i] = v$ is selected with probability proportional to $p_E(v)$. Then set $p_D[i] = 1/m$, so the process is unbiased. Via standard concentration bounds, appropriate choices of m ensure that $T_D(s, w) \approx T_E(s, w)$. The algorithm REDUCE is based on derandomizing this process via a slowed-down simulation: to compute D, we iteratively compute distributions $D_0, D_1, \ldots, D_\ell = D$ by fixing the i^{th} bit level of each entry in D_{i-1}.

Algorithm 1 REDUCE(E, ε, w).

Set $\ell = \lg |E|$ and $m = \frac{C\ell^2 \log \eta}{\varepsilon^2}$ for a constant C.
Solve Prediction Problem for distribution E
Initialize the multiset $H_0 := \{m \text{ empty bitstrings}\}$
for $i = 0, \ldots, \ell - 1$ **do** ▷ Fix i^{th} bit
    Formulate LAP $\mathcal{L}$ for H_i.
    $\vec{v} \leftarrow$ solve $\mathcal{L}$ via Proposition 8 with sampling rate $u_b = \frac{p_E(b1*)}{p_E(b*)}$ for each $b \in H_i$
    $H_{i+1} \leftarrow \{bv_b : b \in H_i\}$ ▷ Concatenate vector $\vec{v}$ as i^{th} bit level
parfor $j \in \{0, \ldots, m-1\}$ **do** ▷ Convert bitstring (index) to drivestream (value)
    Set $D[j] = E[H_\ell[j]]$ and set probability $p_D[j] = 1/m$.
return D

In order to measure distribution error, we introduce the following key definition:

▶ **Definition 11** (Sensitivity). *For a function $w : S \to \mathbb{R}$ and state s, the* sensitivity $\alpha(s, w, E)$ *is:*

$$\alpha(s, w, E) := \max_{\vec{r} \in E} w(F(\vec{r}, s)) - \min_{\vec{r} \in E} w(F(\vec{r}, s))$$

The following is our main result analyzing the algorithm:

▶ **Theorem 12.** *The algorithm* REDUCE *runs in $\tilde{O}((h + \eta)/\varepsilon^2)$ processors, plus the cost of solving the Prediction Problem for E. The final distribution D is uniform with size $O(\frac{\log \eta \log^2 |E|}{\varepsilon^2})$. For large enough constant C, the distribution D satisfies*

$$|T_D(s, w) - T_E(s, w)| \le \varepsilon \alpha(s, w, E) \qquad \text{for each state } s$$

Proof. Let us fix E, w; for brevity, we write $\alpha_s := \alpha(s, w, E)$ for each state s.

Each multiset H_i contains m bitstrings $H_i[0], \ldots, H_i[m-1]$ of length i. Consider the distributions D_i obtained by drawing a bitstring $b \in H_i$ uniformly at random and then drawing drivestream from $E[b*]$. In particular, $D_0 = E$ and the distribution returned by REDUCE is precisely $D_\ell = D$. We have the following equation for every state s:

$$T_{D_i}(s, w) = \frac{1}{m} \sum_{b \in H_i} T_{E[b*]}(s, w).$$

Now, to analyze a given step i, observe that H_{i+1} is obtained by appending a bit v_b to each bitstring $b \in H_i$. We expose the choice of the next bit in D_i and D_{i+1} as follows

$$T_{D_i}(s, w) = \frac{1}{m} \sum_{b \in H_i} \left[\frac{p_E(b1*)}{p_E(b*)} \cdot T_{E[b1*]}(s, w) + \frac{p_E(b0*)}{p_E(b*)} \cdot T_{E[b0*]}(s, w) \right]$$

$$T_{D_{i+1}}(s, w) = \frac{1}{m} \sum_{b \in H_i} \left[v_b \cdot T_{E[b1*]}(s, w) + (1 - v_b) \cdot T_{E[b0*]}(s, w) \right].$$

Thus we can calculate the difference between probabilities for D_i and D_{i+1} as:

$$T_{D_{i+1}}(s, w) - T_{D_i}(s, w) = \frac{1}{m} \sum_{b \in H_i} \left(\frac{p_E(b1*)}{p_E(b*)} - v_b \right) \left(T_{E[b1*]}(s, w) - T_{E[b0*]}(s, w) \right). \tag{1}$$

In light of Eq. (1), we apply Proposition 8 where each state s corresponds to a constraint row k with entries $A_{kb} = T_{E[b1*]}(s,w) - T_{E[b0*]}(s,w)$, and with $u_b = \frac{p_E(b1*)}{p_E(b*)}$ for all b. It is evident that, after solving the Prediction Problem for E, the values $T_{E[bx*]}, p_E(bx*)$ for the Lattice Approximation Problem at Line 6 can be generated using $O(\eta m)$ processors. Furthermore, the maximum spread of the values $w(F(\vec{r},s))$ over $\vec{r}$ is at most α_s, so $\Delta_k = \max_k |A_{kb}| \le \alpha_s$. Since the matrix has η rows and m columns, Proposition 8 gives:

$$\left|T_{D_{i+1}}(s,w) - T_{D_i}(s,w)\right| \le O(\frac{\alpha_s\sqrt{m\log\eta} + \alpha_s\log\eta}{m})$$

By our choice of m, this is at most $\varepsilon\alpha_s/\ell$. Over all iterations, this gives the desired bound

$$\left|T_D(s,w) - T_E(s,w)\right| = \sum_{i=0}^{\ell-1}\left|T_{D_{i+1}}(s,w) - T_{D_i}(s,w)\right| \le \varepsilon\alpha_s. \qquad \blacktriangleleft$$

5 The FOOL Algorithm

We build the automata-fooling distribution via the algorithm Fool (Algorithm 2).

Algorithm 2 Fool(ε) (Assume n is a power of two).

1: Set $D_{0,t} = \Omega_t$ for each $t = 0,\dots,n-1$
2: Determine approximate transition vectors $\hat{V}_t(s) \approx T_{t,n}(s,W) : s \in S$ for all $t = 0,\dots,n$.
3: **for** $i = 0,\dots,\lg n - 1$ **do**
4: **parfor** $t \in \{0, 2h, 4h, 8h, n-2h\}$ where $h = 2^i$ **do**
5: $D_{i+1,t} \leftarrow$ Reduce$(D_{i,t} \times D_{i,t+h}, \delta, \hat{V}_{t+2h})$ for $\delta = \frac{\varepsilon}{20(1+\lg n)}$
return final distribution $D = D_{\lg n,0}$

The algorithm first finds the "expected value" vectors V_t for each distribution $\Omega_{t,n}$. This may take advantage of problem-specific automaton properties, and it may have some small error. We will also describe a few "generic" methods to calculate these probabilities exactly.

The main loop fools distributions on time horizons $h = 1, 2, 4, 8, \dots$ in a bottom-up fashion, and merges them together using the Reduce procedure from the previous section. Specifically, at each level i, it executes Reduce in parallel to combine $D_{i,t}$ and $D_{i,t+h}$ to obtain $D_{i+1,t}$. Note that we never materialize the Cartesian product of $D_{i,t}$ and $D_{i,t+h}$.

▶ **Theorem 13.** *The cost of Fool is $\tilde{O}(n\eta/\varepsilon^2 + \eta\sigma)$ processors, plus the cost of computing the vectors $\hat{V}_t : t = 0,\dots,n$. The final distribution $D_{\lg n,0}$ has size $O(\frac{\log^5(n\eta\sigma/\varepsilon)}{\varepsilon^2})$.*

Let us now proceed to analyze the error of the final distribution D. For purposes of analysis, we define the *exact* transition vector $V_t = T_{t,n}(s,W)$ and its approximation error by $\beta = \max_{s,t}|V_t(s) - \hat{V}_t(s)|$.

▶ **Theorem 14.** *For any state $s \in S$, the final distribution D returned by Fool(ε, W) satisfies*

$$|T_D(s,W) - T_\Omega(s,W)| \le \varepsilon\mathfrak{V}(s) + 3\beta n$$

Proof Sketch. For any state s and times $k \ge t$, let $a_{s,t,k}$ be the maximum value of $\alpha(s', V_{k+1}, \Omega_k)$ over $s' = F(\vec{r},s) : \vec{r} \in \Omega_{t,k}$. We show by induction on i that each $D_{i,t}$ for any s satisfies

$$|T_{D_{i,t}}(s, V_{t+2^i}) - T_{t,t+2^i}(s,W)| \le 3(2^i-1)\beta + 2i\delta \sum_{k=t}^{t+2^i-1} a_{s,t,k} \qquad (2)$$

The base case $i = 0$ is vacuous since $D_{0,t} = \Omega_t$. The final case $i = \lg n, t = 0$ establishes the claimed result, since $\delta = \frac{\varepsilon}{20(1+\lg n)}$ and $\mathfrak{V}(s) = \sum_{t=0}^{n-1} a_{s,0,t}$. $\blacktriangleleft$

As we have mentioned, there can be problem-specific shortcuts to compute (or approximate) the transition matrices T and resulting expected-value vectors V_t. There is the following generic method to compute them via matrix multiplication.

▶ **Proposition 15.** *All vectors $V_t = T_{t,n}(s, W)$, can be computed with $\tilde{O}(n\eta^\omega)$ processors, where ω is the exponent of any efficiently-parallelizable matrix-multiplication algorithm.*

In particular, for the Coppersmith-Winograd algorithm [16], we have $\omega \le 2.38$. (See [29] for further details on parallel implementation.)

Moreover, if we can compute all transition matrices $T_{t,t+h} : h = 2^i, t = j2^i$, then we can compute all vectors $V_t = T_{t,n}(\cdot, W)$ exactly (for any W) with $\tilde{O}(n\eta^2)$ processors.

Therefore, when dealing with multiple statistical tests, we have the following result.

▶ **Corollary 16.** *Consider statistical tests $i = 1, \dots, \ell$, each computed by its own automaton F_i on a state space S_i of size $|S_i| = \eta_i$, with its own weight function W_i. When combined into a single automaton, the resulting automaton F has the following properties:*

1. *The total statespace is $\eta = \sum_i \eta_i$.*
2. *Each starting state s for automaton i has $\mathfrak{V}(s) = \mathfrak{V}_i(s)$, where $\mathfrak{V}_i$ denotes the value of $\mathfrak{V}$ for weight function W_i and automaton F_i.*
3. *The exact vectors V_t can be calculated in $\tilde{O}(n \sum_i \eta_i^\omega)$ processors.*

In particular, the FOOL algorithm has processor complexity $\tilde{O}\big(n\eta/\varepsilon^2 + \sigma\eta + n\sum_i \eta_i^\omega\big)$, and the resulting distribution D has $|T_D(s, W) - T_\Omega(s, W)| \le \varepsilon \mathfrak{V}_i(s)$ for each state s of automaton i.

6 Reducing the state space for counter automata

Given a weight function W, let us consider a statistic of the form

$$W\big(\sum_t f_t(r_t)\big) \qquad \text{for functions } f_t : \Omega_t \to \mathbb{Z}$$

We refer to such statistics as *counters*. They are ubiquitous in randomized algorithm; they are often used directly, and can also be combined together into higher-order statistics.

In the automata-fooling setting, it is easy to compute $\sum_t f_t(r_t)$ by tracking all possible values for the running sum. However, this is often overkill: typically, the running sum is confined to a much smaller window, roughly the standard deviation around the mean. We can take advantage of this by constructing an automaton which only maintains the running sum within "typical" values close to the mean, along with a "reject" state for some exponentially-rare deviations. Let us define some relevant parameters for the counter.

$$\mu_t = \mathbb{E}_{r\sim\Omega_t}[f_t(r)], \quad M_t = \max_{r\in\Omega_t} |f_t(r) - \mu_t|, \quad \kappa = \sum_{t=0}^{n-1} \mathbb{V}\mathrm{ar}_{r\sim\Omega_t}[f_t(r)], \quad M = \max_{t\in[n]} M_t$$

Given some error parameter $\delta > 0$, we choose a value B with $B \ge \lceil 100(1 + M + \sqrt{\kappa}) \log(n/\delta) \rceil$. In this context, we refer to B as the *span* of the automaton.

▶ **Observation 17.** *For a drivestream $\vec{r}$ drawn randomly from Ω, it holds with probability at least $1 - \delta$ that $\left|\sum_{i=t}^{t'} (f(r_i) - \mu_i)\right| < 0.15B$ for all times t, t'.*

Proof. Apply Bernstein's inequality and take a union bound over t, t'. ◀

In light of Observation 17, our strategy will be to construct a truncated automaton $\tilde{F}$, which only stores the running sum within a window of $\pm B$ from the running mean. Define $a_t = \lceil \sum_{i=0}^{t} \mu_i \rfloor$ for each value t. The truncated automaton has state space $\tilde{S} = \{-B, -B+1, \ldots, B-1, B\} \cup \{\perp\}$; given input r_t, it updates its state c to a new state c' as:

$$c' = \begin{cases} c + f(r_t) - a_t + a_{t-1} & \text{if } c \neq \perp \text{ and } |c + f(r_t) - a_t + a_{t-1}| \leq B \\ \perp & \text{if } c = \perp \text{ or } |c + f(r_t) - a_t + a_{t-1}| > B \end{cases}$$

Each integer state c at time t corresponds to a running sum $c + a_t$. The state $\perp$ represents a "reject" or "rare" state where the running sum has deviated too far from its mean. We define a related potential function $\phi : \tilde{S} \to [0, 1]$ as

$$\phi(c) = \begin{cases} 1 & \text{if } |c| \leq B/3 \\ (2B/3 - |c|)/(B/3) & \text{if } B/3 < |c| < 2B/3 \\ 0 & \text{if } |c| > 2B/3 \text{ or } c = \perp \end{cases}$$

Intuitively, ϕ measures how close the current state c is to a reject state. It can be used to "damp" the weight function W and make a gradual transition to the reject state.

For purposes of analysis, it is useful to compare $\tilde{F}$ with the "truthful" automaton F. The potential function ϕ can also be applied on the original state space S, where we use the convention that any states $|s| > B$ corresponds to the reject state $\perp$ of $\tilde{S}$.

Relevant properties of $\tilde{F}$ such as computing its transition matrix or its weight variability can be derived from F up to a small error. Indeed, our application to Gale-Berlekamp Switching Game will use precisely this approach. For the MAX-CUT application, we will need to combine multiple truncated automata; this will require a slightly different damping function and analysis.

▶ **Proposition 18.** *Let W be a weight function for automaton F and define*

$$\tilde{W}(c) = \begin{cases} 0 & \text{if } c = \perp \\ \phi(c)W(c) & \text{if } c \neq \perp, \end{cases} \qquad \Delta = \max_{c \in S} |W(c)|, \qquad \tilde{\Delta} = \max_{\substack{c \in S: \\ |c - \mu_t| \leq 2B}} |W(c)|.$$

Let $T, \tilde{T}$ denote the transition matrices for automata $F, \tilde{F}$ respectively.

1. *For any time t and state $c \in S$, there holds $|\tilde{T}_{t,n}(c, \tilde{W}) - T_{t,n}(c, \tilde{W})| \leq \delta\Delta$.*
2. *For the starting state $c = 0$ and time $t = 0$, there holds $|\tilde{T}_{t,n}(c, \tilde{W}) - T_{t,n}(c, W)| \leq \delta\Delta$.*
3. *For any time t and state c, there holds*

$$\tilde{\mathfrak{C}}(c, t) \leq \mathfrak{L}(c, t) + 2\Delta\delta + 6\tilde{\Delta}M_t/B,$$

where $\tilde{\mathfrak{C}}$ denotes the confusion with respect to automata $\tilde{F}$ and weight function $\tilde{W}$, while $\mathfrak{L}$ denotes the Lipschitz value with respect to automaton F and weight function W.

Note that, in both these cases, we use the convention that if $|c| > B$, then c corresponds to the reject state of automaton $\tilde{F}$.

7 Applications

We present the derandomization of two basic problems: the Gale-Berlekamp Switching Game (Section 7.1) and approximate MAX-CUT via SDP rounding (Section 7.2). Through our new construction, we improve the deterministic processor complexity for both problems.

7.1 Gale-Berlekamp Switching Game

The Gale-Berlekamp Switching Game is a classic combinatorial problem: given an $n \times n$ matrix A with entries $A_{ij} = \pm 1$, we want to find vectors $x, y \in \{-1, +1\}^n$ to maximize the imbalance $I = \sum_{i,j} A_{i,j} x_i y_j$. An elegant randomized algorithm of [4], based on the Central Limit Theorem, gives imbalance of $I \geq (\sqrt{2/\pi} - o(1))n^{3/2}$. This was derandomized in [24], using automata-fooling with a processor complexity of $n^{5+o(1)}$.

▶ **Theorem 19.** *There is a parallel deterministic algorithm to find $\vec{x}, \vec{y} \in \{-1, +1\}^n$ with imbalance $I \geq (\sqrt{2/\pi} - o(1))n^{3/2}$ using $\tilde{O}(n^{3.5})$ processors.*

In order to show Theorem 19, following [8] and [4], we set $x_i = 1$ if $\sum_j A_{ij} y_j > 0$, and $x_i = -1$ otherwise. This gives

$$\sum_{i,j} A_{i,j} x_i y_j = \sum_i x_i \sum_j A_{i,j} y_j = \sum_i \Big| \sum_j A_{i,j} y_j \Big|.$$

As shown in [13], for $\vec{y}$ uniformly drawn from $\Omega = \{-1, +1\}^n$, we have $\mathbb{E}[|\sum_j A_{i,j} y_j|] \geq \sqrt{2n/\pi} - o(\sqrt{n})$ for each i. This is the statistical test we want to fool.

We will choose our drivestream values to be $r_t = y_t$, with Ω_t the uniform distrubition on $\{-1, 1\}$ For each value i, we have a truncated counter automaton $\tilde{F}_i$ to track $\sum_j A_{ij} r_j$. This uses the construction of Section 6 with $\delta = n^{-10}$, where $M = 1$ and $\kappa = \sum_j A_{ij}^2 = n$ and $B = \tilde{O}(\sqrt{n})$. This automaton uses the weight function

$$\tilde{W}(c) = \begin{cases} 0 & \text{if } c = \perp \\ \phi(c)|c| & \text{if } c \neq \perp \end{cases}$$

▶ **Proposition 20.** *Fix a row i and automaton $\tilde{F} = \tilde{F}_i$, and correspondingly let F be the full "truthful" automaton. Let $T, \tilde{T}$ be the transition matrices for $F, \tilde{F}$ respectively.*

1. *For any time t and state s of F, we have $|T_{t,n}(s, \tilde{W}) - \tilde{T}_{t,n}(s, \tilde{W})| \leq 2\delta n$.*
2. *The final state $\tilde{z} = \tilde{F}(\vec{y}, 0)$ for $\vec{r} \sim \Omega$ satisfies $\mathbb{E}[\tilde{W}(\tilde{z})] \geq \sqrt{2n/\pi} - o(\sqrt{n})$.*
3. *The weight function $\tilde{W}$ for automaton $\tilde{F}$ has total variability $O(n)$.*

Proof. We apply here Proposition 18 with $W = |c|$. Clearly here $\mathfrak{L}(c, t) = 2$ and $M_t \leq 1$ for all c, t. Also, we calculate $\Delta = n, \tilde{\Delta} \leq \tilde{O}(\sqrt{n})$. With Proposition 18(a), this proves (a). Similarly, by Proposition 18(b), we have

$$\mathbb{E}[\tilde{W}(\tilde{z})] \leq \delta n + \mathbb{E}_\Omega[W(z)] \leq o(\sqrt{n}) + (\sqrt{2/\pi} - o(1)) \cdot \sqrt{n}$$

where z is the final state for the automaton F.

Finally, for (c), we use the definition of total variability and Proposition 18(c) to get:

$$\mathfrak{V}(c) = \sum_{t=0}^{n-1} \max_{\vec{r} \in \Omega_{0,t}} \tilde{\mathfrak{C}}(\tilde{F}(\vec{r}, c), t) \leq \sum_{t=0}^{n-1} (\mathfrak{L}(c, t) + 2\Delta\delta + 6\tilde{\Delta} M_t / B) = \tilde{O}(n). \qquad \blacktriangleleft$$

The complexity now follows from Corollary 16 with $\varepsilon = 1/\sqrt{n \log n}$, where we have n automata each of state space $\hat{\eta}_i = \tilde{O}(\sqrt{n})$. By applying Proposition 20, we get that for any sequence $\vec{y}$ drawn from the resulting distribution D and automaton $\tilde{F}_i$,

$$\begin{aligned} |\tilde{T}_D(0, \tilde{W}) - T_\Omega(0, W)| &\leq |\tilde{T}_D(0, \tilde{W}) - \tilde{T}_\Omega(0, \tilde{W})| + |\tilde{T}_\Omega(0, \tilde{W}) - T_\Omega(0, W)| \\ &\leq \varepsilon \mathfrak{V}(0) + o(\sqrt{n}) = o(\sqrt{n}) \end{aligned}$$

Therefore, for any row i, the final state $\tilde{z}$ satisfies

$$\mathbb{E}_D\Big[\big|\sum_j A_{ij}y_j\big|\Big] \geq \mathbb{E}_\Omega[W_i(\tilde{z})] - o(\sqrt{n}) \geq (\sqrt{2/\pi} - O(\epsilon))\sqrt{n}$$

By searching the space exhaustively, we can find a specific sequence $\vec{y}$ satisfying the desired imbalance bounds. This concludes the proof of Theorem 19.

7.2 Approximate MAX-CUT

The MAX-CUT problem for a graph $G = (V, E)$ is to find a vertex set $S \subseteq V$ to maximize the total weight of edges with exactly one endpoint in S. The seminal work of Goemans & Williamson [21] showed that MAX-CUT can be approximated to a factor of $\alpha \approx 0.878$ by rounding its semi-definite programming (SDP) formulation. Moreover, the integrality gap of the SDP is precisely α [18], and assuming the Unique Games Conjecture no better approximation ratio is possible for polynomial-time algorithms [34].

A sequential deterministic α-approximation with $\tilde{O}(n^3)$ runtime was shown in [9]. This relies on the method of conditional expectation, which is hard to parallelize. The work of [43] used the automata derandomization framework to get a parallel deterministic $\alpha(1-\varepsilon)$-approximation algorithm. The main downside of this approach is the huge polynomial processor complexity (on the order of n^{100}). We give an improved analysis with our new automata-fooling construction and some other optimizations.

To review, note that MAX-CUT can be formulated as the following integer program:

$$\max \quad \frac{1}{2}\sum_{(i,j)=e\in E} w_e(1 - v_i \cdot v_j) \quad \text{s.t.} \quad v_i \in \{-1, 1\}, i \in [n].$$

This can be relaxed to the following SDP:

$$\max \quad \frac{1}{2}\sum_{(i,j)=e\in E} w_e(1 - v_i \bullet v_j) \quad \text{s.t.} \quad v_i \in [-1, 1]^n, ||v_i||_2 = 1, i \in [n],$$

where $\bullet$ denotes the inner product in $\mathbb{R}^n$.

We can round a given SDP solution v by drawing independent standard Gaussian random variables $X_0, \ldots, X_{n-1}$ and constructing the cut $S = \{i \mid v_i \bullet X \geq 0\}$, which gives cutsize W

$$\begin{aligned}\mathbb{E}[W] &= \sum_{(i,j)=e\in E} w_e \Pr[(i,j) \text{ is cut}] = \sum_{(i,j)=e\in E} w_e \Pr[\operatorname{sgn}(v_i \bullet X) \neq \operatorname{sgn}(v_j \bullet X)] \\ &= \sum_{(i,j)=e\in E} w_e \frac{\arccos(v_i \bullet v_j)}{\pi} \geq \alpha \cdot \mathrm{OPT},\end{aligned}$$

where OPT denotes the size of the maximum cut. Following [43], we will derandomize this by fooling each statistical test $\Pr[\operatorname{sgn}(v_i \bullet X) \neq \operatorname{sgn}(v_j \bullet X)]$.

▶ **Theorem 21.** *There is a deterministic parallel algorithm to find an $\alpha(1-\varepsilon)$-approximate rounding of a MAX-CUT SDP solution with $\tilde{O}(\frac{mn^3}{\varepsilon^4})$ processors.*

Via the algorithm of [1] to solve the SDP, this gives the following main result:

▶ **Corollary 22.** *For arbitrary constant $\varepsilon > 0$, we can find an $\alpha(1-\varepsilon)$-approximate MAX-CUT solution using $\tilde{O}(mn^3)$ processors and polylogarithmic time.*

The remainder of the section is devoted to showing Theorem 21. To avoid cumbersome calculations, we will show an algorithm for a $(1 - O(\varepsilon))$-approximation, assuming ε is smaller than any needed constants. For readability, $\tilde{O}()$ suppresses any polylog(n/ε) terms.

As a starting point, consider the following process. We draw n independent standard Gaussian variables $X_0, \dots, X_{n-1}$. For each edge (i, j), we have a statistical test to read these values in order and compute $c_i = v_i \bullet X$ and $c_j = v_j \bullet X$. We then apply the weight function $W_{ij}(c_i, c_j)$ as the indicator variable that $\operatorname{sgn}(c_i) = \operatorname{sgn}(c_j)$.

In order to apply our derandomization framework efficiently, we make a number of changes to this basic strategy: (i) we define the drivestream value r_t to be a discretized truncated Gaussian; (ii) we further quantize each term $v_{ik}r_k$ in the computation of the sum $c_i = \sum_k v_{ik}r_k$; (iii) we apply the optimization of Section 6 to reduce the state space for each counter automaton; and (iv) we modify the weight function to a pessimistic estimator with better Lipschitz properties.Let us fix two quantization parameters for some constant $C > 0$

$$\gamma = \frac{\varepsilon}{C\sqrt{n \log(n/\varepsilon)}}, \qquad R = C \log(n/\varepsilon)$$

Concretely, each drivestream value r_k is derived by truncating a standard Gaussian to within $\pm R$, and rounding to the nearest multiple of γ. Then, each term in the product $v_{ik}r_k$ is rounded to the nearest multiple of γ. Since this comes up frequently in the analysis, let us denote this "rounded" inner product by: $v \star r = \gamma \sum_k \lceil \frac{1}{\gamma} v_k r_k \rfloor$. Given this definition, we will form our cut set by setting $S = \{i : v_i \star r \geq 0\}$.

▶ **Lemma 23.** *For appropriately chosen C, there is a coupling between random variables X and Y such that, for any unit vector $u \in \mathbb{R}^n$, there holds $\Pr\big(|u \bullet X - u \star r| > \varepsilon\big) \leq (\varepsilon/n)^{10}$.*

Observe that the scaled sum $\frac{1}{\gamma}(v_i \star r)$ is an integer-valued counter. We can apply the method of Section 6 to construct a "vertex" automaton $\tilde{F}_i$ that tracks the running sum within a window. We record a few parameters and observations about this automaton.

▶ **Proposition 24.** *The automaton $\tilde{F}_i$ can be implemented with the following parameters:*

$$M_t \leq \tilde{O}(|v_{it}|/\gamma), \quad M \leq \tilde{O}(1/\gamma), \quad \kappa \leq \tilde{O}(1/\gamma^2), \quad \delta = (\varepsilon/n)^{10}, \quad B = \tilde{\Theta}(1/\gamma).$$

From these vertex-automata, we construct an edge-automaton $\tilde{F}_{ij}$ for each edge $(i, j) \in E$. Automaton $\tilde{F}_{ij}$ keeps track of the joint states of $\tilde{F}_i, \tilde{F}_j$, i.e. the truncated running sums $v_i \star r$ and $v_j \star r$. It uses the following weight function for the state $s = (c_i, c_j)$:

$$\tilde{W}_{ij}(s) = \begin{cases} 0 & \text{if } c_i = \perp \text{ or } c_j = \perp \\ 0 & \text{if } \operatorname{sgn}(c_i) = \operatorname{sgn}(c_j) \\ \min\{1, \frac{|c_i - c_j|}{\varepsilon}\}\phi_i(c_i)\phi_j(c_j) & \text{otherwise} \end{cases}$$

Note that whenever $\tilde{W}_{ij}(s) > 0$, we have $\operatorname{sgn}(v_i \star r) \neq \operatorname{sgn}(v_j \star r)$, i.e. edge ij is cut.

▶ **Proposition 25.** *For an edge ij, let $\tilde{F} = \tilde{F}_{ij}$, and consider the "truthful" automaton $F = F_{ij}$ (which uses the full untruncated automata for vertices i and j). Let $T, \tilde{T}$ denote the transition matrices for $F, \tilde{F}$ respectively.*

1. *For any time t and state s of F, there holds $|T_{t,n}(s, \tilde{W}) - \tilde{T}_{t,n}(s, \tilde{W})| \leq 2\delta$.*
2. *The final automaton state $\tilde{z} = \tilde{F}(\vec{r}, 0)$ for $\vec{r} \sim \Omega$ satisfies $\mathbb{E}[\tilde{W}(\tilde{z})] \geq \frac{\arccos(v_i \bullet v_j)}{\pi} - O(\varepsilon)$.*
3. *For drivestreams $\vec{r}^1, \vec{r}^2$ differing in coordinate t, the final states $z^1 = F(\vec{r}^1, 0), z^2 = F(\vec{r}^2, 0)$ of automaton F have $|\tilde{W}(z^1) - \tilde{W}(z^2)| \leq \tilde{O}(\varepsilon^{-1}(|v_{it}| + |v_{jt}|))$.*
4. *The weight function $\tilde{W}$ has total variability $\tilde{O}(\sqrt{n}/\varepsilon)$ for automaton $\tilde{F}$.*

Proof. The proof is very similar to Proposition 18; see full version. ◀

Next, we discuss how to approximate the transition matrices for the automata. In light of Proposition 25(a), we will compute vectors $\hat{V}_t(s) = T_{t,n}(s, W)$ for the full automaton F_{ij}. This is much easier than computing the transition matrices for $\tilde{F}_{ij}$, since we do not need to track reject states. In particular, this achieves error $\beta = 2\delta$ compared to $V_t = \tilde{T}_{t,n}(s, W)$.

▶ **Proposition 26.** *For each automaton F_{ij}, the vectors $T_{t,n}(s, W) : t = 0, \dots, n$, can be computed with $\tilde{O}(n^3\varepsilon^{-4})$ processors.*

Proof Sketch. We will recursively compute transition matrices $T_{t,h}$ for $h = 1, 2, 4 \dots$, and t divisible by h. The critical observation here is that since F_{ij} is a pair of counters, we only need to keep track of the probability distribution on the difference pairs

$$\Big(\sum_{k=t}^{t+h-1} \lceil v_{ik} r_k / \gamma \rfloor, \sum_{k=t}^{t+h-1} \lceil v_{jk} r_k / \gamma \rfloor \Big).$$

This is effectively a convolution, computable via an FFT. ◀

We are now ready to compute the total complexity. Each probability distribution Ω_t has size $\tilde{O}(\gamma^{-1}) = \tilde{O}(\sqrt{n}/\varepsilon)$, thus $\sigma = \tilde{O}(n^{1.5}/\varepsilon)$. The reduced total number of states of an edge-automaton F_{ij} is $\tilde{O}(1/\gamma^2) = \tilde{O}(n/\varepsilon^2)$ and the weight function W_{ij} has total variability $\tilde{O}(\sqrt{n}/\varepsilon)$. So we run Fool with parameter $\varepsilon' = \varepsilon/\sqrt{n}$, and the fooling process has cost $\tilde{O}(m\frac{n^3}{\varepsilon^4} + m\frac{n^{2.5}}{\varepsilon^3})$. Consider the resulting distribution D. The edge i, j is only cut if $\tilde{W}_{ij}(s) > 0$. For each F_{ij}, by Theorem 14, the resulting final state z satisfies

$$\Pr_{\vec{r} \sim D}(\text{edge } ij \text{ cut}) \geq \mathbb{E}_{\vec{r} \sim D}[W_{ij}] \geq \mathbb{E}_{\vec{r} \sim \Omega}[W_{ij}] - O(\varepsilon) - O(\beta n)$$

Here Proposition 25(a) gives $\beta \leq 2\delta \ll \varepsilon/n$, and Proposition 25(b) gives $\mathbb{E}_{\vec{r} \sim \Omega} \geq \frac{\arccos(v_i \bullet v_j)}{\pi} - O(\varepsilon)$. Thus, overall, the edge i, j is cut with prob. at least $\frac{\arccos(v_i \bullet v_j)}{\pi} - O(\varepsilon)$. Summing over all edges, the total expected weight of the cut edges is

$$\sum_{ij \in E} w_e \frac{\arccos(v_i \bullet v_j)}{\pi} - \sum_{ij \in E} w_e O(\varepsilon)$$

The first term is at least $\alpha \cdot \text{OPT}$. The second term is at most $O(\varepsilon \cdot \text{OPT})$ since $\text{OPT} \geq \sum_e w_e/2$. By searching D exhaustively, we can find a cut satisfying these bounds.

References

1. Zeyuan Allen-Zhu, Yin Tat Lee, and Lorenzo Orecchia. Using optimization to obtain a width-independent, parallel, simpler, and faster positive SDP solver. In *Proc. 27th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1824–1831, 2016. doi:10.1137/1.9781611974331.ch127.
2. Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.
3. Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992. doi:10.1002/rsa.3240030308.
4. Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2015. URL: http://cs.nyu.edu/cs/faculty/spencer/nogabook/nogabook.html.

5 Yossi Azar, Rajeev Motwani, and Joseph Naor. Approximating probability distributions using small sample spaces. *Combinatorica*, 18(2):151–171, 1998. doi:10.1007/PL00009813.

6 Nikhil Bansal, Nitish Korula, Viswanath Nagarajan, and Aravind Srinivasan. Solving packing integer programs via randomized rounding with alterations. *Theory of Computing*, 8(24):533–565, 2012. doi:10.4086/toc.2012.v008a024.

7 Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *Proc. 35th annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–287, 1994. doi:10.1109/SFCS.1994.365687.

8 Bonnie Berger. The fourth moment method. *SIAM Journal on Computing*, 26(4):1188–1207, 1997. doi:10.1137/S0097539792240005.

9 Ankur Bhargava and S Rao Kosaraju. Derandomization of dimensionality reduction and SDP based algorithms. In *Workshop on Algorithms and Data Structures*, pages 396–408. Springer, 2005. doi:10.1007/11534273_35.

10 Manuel Blum and Oded Goldreich. Towards a computational theory of statistical tests. In *Proc. 33rd annual Symposium on Foundations of Computer Science (FOCS)*, pages 406–416, 1992. doi:10.1109/SFCS.1992.267749.

11 Allan Borodin. On relating time and space to size and depth. *SIAM journal on computing*, 6(4):733–744, 1977. doi:10.1137/0206054.

12 Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudayoff. Pseudorandom generators for regular branching programs. *SIAM Journal on Computing*, 43(3):973–986, 2014. doi:10.1137/120875673.

13 Thomas A Brown and Joel H Spencer. Minimization of ±1 matrices under line shifts. In *Colloquium Mathematicum*, volume 1, pages 165–171, 1971.

14 Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. Improved algorithms via approximations of probability distributions. In *Proc. 26th annual ACM Symposium on Theory of Computing (STOC)*, pages 584–592, 1994. doi:10.1145/195058.195411.

15 Lijie Chen, William M. Hoza, Xin Lyu, Avishay Tal, and Hongxun Wu. Weighted pseudorandom generators via inverse analysis of random walks and shortcutting. In *Proc. 64th annual Symposium on Foundations of Computer Science (FOCS)*, pages 1224–1239, 2023. doi:10.1109/FOCS57990.2023.00072.

16 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proc. 19th annual ACM Symposium on Theory of Computing (STOC)*, pages 1–6, 1987. doi:10.1145/28395.28396.

17 Guy Even, Oded Goldreich, Michael Luby, Noam Nisan, and Boban Veličkovic. Approximations of general independent distributions. In *Proc. 24th annual ACM Symposium on Theory of Computing (STOC)*, pages 10–16, 1992. doi:10.1145/129712.129714.

18 Uriel Feige and Gideon Schechtman. On the optimality of the random hyperplane rounding technique for MAX CUT. *Random Structures & Algorithms*, 20(3):403–440, 2002. doi:10.1002/rsa.10036.

19 Mohsen Ghaffari and Christoph Grunau. Work-efficient parallel derandomization II: Optimal concentrations via bootstrapping. In *Proc. 56th annual ACM Symposium on Theory of Computing (STOC)*, pages 1889–1900, 2024. doi:10.1145/3618260.3649668.

20 Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. Work-efficient parallel derandomization I: Chernoff-like concentrations via pairwise independence. In *Proc. 64th annual Symposium on Foundations of Computer Science (FOCS)*, pages 1551–1562, 2023. doi:10.1109/FOCS57990.2023.00094.

21 Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995. doi:10.1145/227683.227684.

22 Parikshit Gopalan, Raghu Meka, Omer Reingold, Luca Trevisan, and Salil Vadhan. Better pseudorandom generators from milder pseudorandom restrictions. In *Proc. 53rd annual Symposium on Foundations of Computer Science (FOCS)*, pages 120–129, 2012. doi:10.1109/FOCS.2012.77.

23 Parikshit Gopalan, Raghu Meka, Omer Reingold, and David Zuckerman. Pseudorandom generators for combinatorial shapes. In *Proc. 43rd annual ACM Symposium on Theory of Computing (STOC)*, pages 253–262, 2011. doi:10.1145/1993636.1993671.

24 David G Harris. Deterministic parallel algorithms for bilinear objective functions. *Algorithmica*, 81:1288–1318, 2019. doi:10.1007/s00453-018-0471-0.

25 David G Harris. Deterministic algorithms for the Lovász local lemma: simpler, more general, and more parallel. *Random Structures & Algorithms*, 63(3):716–752, 2023. doi:10.1002/rsa.21152.

26 David G Harris, Ehab Morsy, Gopal Pandurangan, Peter Robinson, and Aravind Srinivasan. Efficient computation of sparse structures. *Random Structures & Algorithms*, 49(2):322–344, 2016. doi:10.1002/rsa.20653.

27 William M Hoza, Edward Pyne, and Salil Vadhan. Pseudorandom generators for unbounded-width permutation branching programs. In *Proc. 12th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 185, pages 7:1–7:20, 2021. doi:10.4230/LIPIcs.ITCS.2021.7.

28 Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 356–364, 1994. doi:10.1145/195058.195190.

29 Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley Publishing Company, 1992.

30 Anatole Joffe. On a set of almost deterministic k-independent random variables. *the Annals of Probability*, 2(1):161–162, 1974.

31 David R Karger and Daphne Koller. (De)randomized construction of small sample spaces in NC. *Journal of Computer and System Sciences*, 55(3):402–413, 1997. doi:10.1006/jcss.1997.1532.

32 Howard Karloff and Yishay Mansour. On construction of k-wise independent random variables. In *Proc. 26th annual ACM Symposium on Theory of Computing (STOC)*, pages 564–573, 1994. doi:10.1007/BF01196134.

33 Zander Kelley and Raghu Meka. Random restrictions and PRGs for PTFs in Gaussian space. In *Proc. 37th Computational Complexity Conference (CCC)*, 2022. doi:10.4230/LIPIcs.CCC.2022.21.

34 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007. doi:10.1137/S0097539705447372.

35 Michael Luby. A simple parallel algorithm for the maximal independent set problem. In *Proc. 17th annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1985. doi:10.1145/22145.22146.

36 Sanjeev Mahajan, Edgar A Ramos, and KV Subrahmanyam. Solving some discrepancy problems in NC. *Algorithmica*, 29(3):371–395, 2001. doi:10.1007/s004530010046.

37 Raghu Meka, Omer Reingold, and Avishay Tal. Pseudorandom generators for width-3 branching programs. In *Proc. 51st annual ACM Symposium on Theory of Computing (STOC)*, pages 626–637, 2019. doi:10.1145/3313276.3316319.

38 Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. In *Proc. 22nd annual ACM Symposium on Theory of Computing (STOC)*, pages 213–223, 1990. doi:10.1145/100216.100244.

39 Noam Nisan. Pseudorandom generators for space-bounded computations. In *Proc. 22nd annual ACM Symposium on Theory of Computing (STOC)*, pages 204–212, 1990. doi:10.1145/100216.100242.

40 Noam Nisan. RL $\subseteq$ SC. In *Proc. 24th annual ACM Symposium on Theory of Computing (STOC)*, pages 619–623, 1992. doi:10.1145/129712.129772.

41 Ryan O'Donnell, Rocco A Servedio, and Li-Yang Tan. Fooling Gaussian PTFs via local hyperconcentration. In *Proc. 52nd annual ACM Symposium on Theory of Computing (STOC)*, pages 1170–1183, 2020. doi:10.1145/3357713.3384281.

42 Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995. doi:10.1137/S089548019223872X.

43 D Sivakumar. Algorithmic derandomization via complexity theory. In *Proc. 34th annual ACM Symposium on Theory of Computing (STOC)*, pages 619–626, 2002. doi:10.1145/509907.509996.

44 Aravind Srinivasan. New approaches to covering and packing problems. In *Proc. 12th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 567–576, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365535.

Simpler Universally Optimal Dijkstra

Ivor van der Hoog
IT University of Copenhagen, Denmark

Eva Rotenberg
IT University of Copenhagen, Denmark

Daniel Rutschmann
IT University of Copenhagen, Denmark

Abstract

Let G be a weighted (directed) graph with n vertices and m edges. Given a source vertex s, Dijkstra's algorithm computes the shortest path lengths from s to all other vertices in $O(m + n \log n)$ time. This bound is known to be worst-case optimal via a reduction to sorting. Theoretical computer science has developed numerous fine-grained frameworks for analyzing algorithmic performance beyond standard worst-case analysis, such as instance optimality and output sensitivity. Haeupler, Hladík, Rozhoň, Tarjan, and Tětek [FOCS '24] consider the notion of universal optimality, a refined complexity measure that accounts for both the graph topology and the edge weights. For a fixed graph topology, the universal running time of a weighted graph algorithm is defined as its worst-case running time over all possible edge weightings of G. An algorithm is universally optimal if no other algorithm achieves a better asymptotic universal running time on any particular graph topology.

Haeupler, Hladík, Rozhoň, Tarjan, and Tětek show that Dijkstra's algorithm can be made universally optimal by replacing the heap with a custom data structure. Their approach builds on Iacono's [SWAT '00] working-set bound $\phi(x)$. This is a technical definition that, intuitively, for a heap element x, counts the maximum number of simultaneously-present elements y that were pushed onto the heap whilst x was in the heap. They design a new heap data structure that can pop an element x in $O(1 + \log \phi(x))$ time. They show that Dijkstra's algorithm with their heap data structure is universally optimal.

In this work, we revisit their result. We use a simpler heap property that we will call timestamp optimality, where the cost of popping an element x is logarithmic in the number of elements inserted between pushing and popping x. We show that timestamp optimal heaps are not only easier to define but also easier to implement. Using these time stamps, we provide a significantly simpler proof that Dijkstra's algorithm, with the right kind of heap, is universally optimal.

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

Keywords and phrases Graph algorithms, instance optimality, Fibonnacci heaps, simplification

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.71

Related Version *Full Version*: https://arxiv.org/abs/2504.17327

Funding *Ivor van der Hoog*, *Eva Rotenberg*, and *Daniel Rutschmann* are grateful to the Carlsberg Foundation for supporting this research via Eva Rotenberg's Young Researcher Fellowship CF21-0302 "Graph Algorithms with Geometric Applications". This work was supported by the the VILLUM Foundation grant (VIL37507) "Efficient Recomputations for Changeful Problems" and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987.



© Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Let $G = (V, E)$ be a (di)graph with edge weights w, n vertices, and m edges. Let s be a designated source vertex. In the *single-source shortest path* problem, the goal is to sort all the vertices in V by the length of the shortest path from s to v. Dijkstra's algorithm with a Fibonacci heap solves this problem in $O(m + n \log n)$ time, which is worst-case optimal.

However, worst-case optimality can be overly coarse as a performance measure. A stronger notion, called *universal optimality*, requires an algorithm to run as fast as possible on *every* graph topology. Recently, Haeupler, Hladík, Rozhoň, Tarjan, and Tětek [6] proved that Dijkstra's algorithm, when paired with a suitably designed heap, achieves universal optimality. The paper was well-received, as it won the best paper award at FOCS'24. Their contribution is threefold: First, they observe that Dijkstra's efficiency relies on a heap supporting Push and DecreaseKey in amortized constant time, and Pop in $O(\log n)$ time. They refine this analysis using the *working-set bound* by Iacono [9] (we forward reference Definition 1). Intuitively, for a heap element x, the working set size $\phi(x)$ is the maximum number of elements y that are simultaneously present in the heap while x resides in it (excluding all y' that already were in the heap). Haeupler et al. [6] design a new heap data structure for in the word-RAM model in which the Pop operation for element x takes $O(1 + \log \phi(x))$ time.

Secondly, they show, for any fixed graph topology $G = (V, E)$ and designated source s, a universal lower bound of $\Omega(m + n + \sum_{x_i \in V \setminus \{s\}} \log \phi(x_i))$. This lower bound matches the running time of Dijkstra's algorithm, when it is equipped with their heap. They thereby prove that Dijkstra's algorithm is universally optimal.

Third, they refine their analysis by distinguishing between edge weight comparisons and all other operations. This argument is technical, and we only refer to it on a high level. For completeness, we explain this result in great detail in the full version. Imagine $m_G \leq m$ to be some technical parameter that only depends on G. Note that, for any fixed weighted graph, Dijkstra's algorithm always performs the same sequence of push and pop operations on the heap. Therefore, the weighted graph uniquely determines the working set size of each element. For a fixed graph topology G, for any weighting w of G with corresponding working set sizes $\phi(x_i)$, they prove a universal lower bound of $\Omega(m_G + \sum_{x_i \in V \setminus \{s\}} (1 + \log(\phi(x_i))))$ on the number of weight comparisons used to solve the single shortest path problem. They adapt Dijkstra's algorithm so that the number of edge weight comparisons performed is also asymptotically universally optimal. A recent paper by Haeupler et al. [7] shows a simple instance optimal result for the bidirectional problem: Compute, given G and two vertices (s, t), the shortest st-path. Although the problems are related, the techniques and bounds are different, and thus, we do not consider the st-path problem in the paper at hand.

Contribution. We offer a significantly more concise proof of Dijkstra's universal optimality. Our main insight is to replace the working-set bound with a conceptually simpler measure called *timestamps*.[1] We maintain a heap H and global time counter that increments after H.Push. For each heap element x_i, let a_i and b_i denote the time it was pushed and popped, respectively. We define H to be *timestamp optimal* if popping x_i takes $O(1 + \log(b_i - a_i))$ time. Timestamps are not only conceptually simpler than working-set sizes, timestamp optimality is also considerably simpler to implement.

[1] We note that the original work by Iacono [10] included several notions of working-set bounds. One of these notions, not present in [6, 9], but used in [5], matches our notion of timestamp optimality. By working set, we explicitly refer to the definition used in [6, 9]. We use the more distinct term timestamps to properly distinguish between the concepts in this work and [6]. This also highlights that our notion yields a family of intervals $[a_i, b_i]$, which is more convenient than having just the working set sizes $\phi(x_i)$.

We design a heap data structure that supports Push and DecreaseKey in amortized constant time, and Pop in amortized $O(1 + \log(b_i - a_i))$ time. Using timestamps, we show a universal lower bound of $\Omega(m + \sum_{x_i \in V \setminus \{s\}}(1 + \log(b_i - a_i)))$ for any fixed graph topology $G = (V, E)$ and source s. We consider this to be our main contribution, as it offers a significantly simpler proof that Dijkstra's algorithm can be made universally optimal. Finally, our formulation of timestamps allow us to apply recent techniques for universally optimal sorting [5, 11] to show that Dijkstra's algorithm, with our timestamp optimal heap, matches our universal lower bound. Additionally, we show in the full version that our algorithm is equally general to that of [6], as it can be made universally optimal with respect to edge weight comparisons. While this proof is indeed new, different, and perhaps simpler, it is not self-contained; therefore, we relegate it to the full version for the sake of completeness.

2 Preliminaries

Let $G = (V, E)$ be a (directed) graph and let w assign to each edge a positive weight. For ease of notation, we say that G has $n + 1$ vertices and m edges. In the *single-source shortest path* problem (SSSP), the input consists of G, w, and a designated source vertex s. The goal is to order all n vertices in $V \setminus \{s\}$ by the length of the shortest path from s to each vertex $v \in V$. We follow [6] and assume that the designated source s can reach every vertex in $V \setminus \{s\}$ and that no two vertices are equidistant from s.

For weighted graph algorithms A that solves SSSP, the running time $\text{Runtime}(A, G, s, w)$ depends on the graph topology $G = (V, E)$, the designated source vertex s, and the edge weights w. Let $\mathbb{G}_n$ denote the set of all (directed) graph topologies $G = (V, E)$ with $|V| = n + 1$. For a fixed graph topology G, let $\mathbb{W}(G)$ denote the (infinite) set of all possible edge weightings of G. Then, for a fixed input size n, the worst-case running time of an algorithm A that solves the single-source-shortest path problem is a triple maximum:

$$\text{Worst-case}(A, n) := \max_{G \in \mathbb{G}_n} \max_{s \in V(G)} \max_{w \in \mathbb{W}(G)} \text{Runtime}(A, G, s, w).$$

An algorithm A is worst-case optimal if there exists a constant c such that, for sufficiently large n, there exists no algorithm A' with $\text{Worst-case}(A, n) > c \cdot \text{Worst-case}(A', n)$.

Haeupler, Wajc, and Zuzic [8] introduce a more fine-grained notion of algorithmic optimality for weighted graphs called *universal optimality*. Universal optimality requires an algorithm to perform optimally for every fixed graph topology. Formally, in the single-source-shortest path problem, it removes the outer two maximizations: For a fixed algorithm A, graph topology $G = (V, E)$, and source s, the universal running time is

$$\text{Universal}(A, G, s) := \max_{w \in \mathbb{W}(G)} \text{Runtime}(A, G, s, w).$$

An algorithm A is *universally optimal* if there exists a constant c such that for every fixed topology G and every $s \in V$, no algorithm A' satisfies $\text{Universal}(A, G, s) > c \cdot \text{Universal}(A', G, s)$.

2.1 Dijkstra's algorithm

Dijkstra [2] presented an algorithm for the single-source shortest paths problem in 1959. Since the invention of Fibonnacci heaps [3], Dijkstra's algorithm can be implemented in a way that is worst-case optimal, by utilising these heaps in the implementation of Dijkstra's algorithm. A heap H maintains a set of elements where each $x \in H$ is associated with a unique *key* $\gamma(x)$. A heap H typically supports the following five operations:

Algorithm 1 Dijkstra's algorithm with a black-box heap H.

Require: (Di)graph $G = (V, E)$, weighting w, designated source $s \in V$
  `distance` $\leftarrow$ array with n entries equal to ∞
  H $\leftarrow$ empty min-heap
  `distance`$[s] = 0$
  H.PUSH$(s, 0)$
  while H is not empty **do**
    u $\leftarrow$ H.POP()
    OUTPUT.append(u)
    for Each edge $(u, v) \in E$ **do**
      if `distance`$[v]$ is ∞ **then**
        `distance`$[v] \leftarrow$ `distance`$[u] + w((u, v))$
        H.PUSH$(v,$ `distance`$[v])$
      else if v is in H and `distance`$[v] >$ `distance`$[u] + w((u, v))$ **then**
        `distance`$[v] \leftarrow$ `distance`$[u] + w((u, v))$
        H.DECREASEKEY$(v,$ `distance`$[v])$
      end if
    end for
  end while

- PUSH(x, y): Insert x into H where the key $\gamma(x)$ is y.
- DECREASEKEY(x, y): For $x \in H$ and $y \leq \gamma(x)$, update $\gamma(x)$ to y.
- POP(): Remove and return the element $x \in H$ with the smallest key.
- PEAK(): Return an element $x \in H$ with the smallest key.
- MELD(H'): Return a heap H'' on the elements in $H \cup H'$.

Fibonacci heaps support PUSH and DECREASEKEY in $O(1)$ amortized time, POP in amortized $O(\log n)$ time, and PEAK and MELD in worst-case $O(1)$ time [3]. Dijkstra's algorithm only uses the first three heap operations. Using Fibonacci heaps, it runs in $O(m + n \log n)$ time which is worst-case optimal. Dijkstra's algorithm (Algorithm 1) is conceptually simple: it initializes the heap with all vertices v that s can directly reach, where the key $\gamma(v)$ is the edge weight of (s, v). Then, it repeatedly pops the vertex u with the smallest key and it appends u to the output. Each time a vertex u is popped, its neighbors v are examined. For each neighbor v, the tentative distance d_v is computed as the distance from s to u plus the edge weight $w(u, v)$. If v is not in the heap, it is pushed with key $\gamma(v) = d_v$. If v is already in the heap and $\gamma(v) > d_v$ then the key $\gamma(v)$ is decreased to d_v.

2.2 Universally optimal Dijkstra

Iacono [9] introduced a refined measure for the running time of heap operations via the concept of *working-set size*:

▶ **Definition 1** ([9]). *Consider a heap supporting* PUSH *and* POP *and let x_i be an element in the heap. For each time step t between pushing x_i and popping x_i, we define the working set of x_i at time t as the set of elements $W(x_i, t)$ inserted after x_i and still present at time t. We include x_i itself in $W(x_i, t)$. Fix any time t_0 that maximizes the value of $|W(x_i, t)|$; we call the set $W(x_i, t_0)$* working-set *of x_i and $\phi(x_i) := |W(x_i, t_0)|$ the* working-set size.

Haeupler, Hladík, Rozhoň, Tarjan, and Tětek [6] design a heap H that supports Push and DecreaseKey in amortized $O(1)$ time, and supports Pop for an element $x \in H$ in $O(1 + \log \phi(x))$ time. The basis of their data structure are Fibonacci heaps F which support Push and DecreaseKey in amortized constant time and Pop in amortized $O(\log N)$ time where N is the number of elements in the heap.

The core idea of their data structure is to partition elements into buckets B_j of doubly exponentially increasing size. Each bucket B_j contains a Fibonacci heap F_j. Denote by m_j the minimum key of F_j. The set of $O(\log \log n)$ values m_j is stored in a fusion tree [4]. Fusion trees of size at most $O(\log \log n)$ have constant update time in the word-RAM model. By carefully merging and managing buckets, they ensure that each element x_i is stored in a Fibonacci heap F_j whose size is proportional to $\phi(x_i)$. We refer to [6] for full details; we only note here that ensuring that Push and DecreaseKey take amortized constant time whilst Pop takes $O(1 + \log \phi(x_i))$ time requires non-trivial maintenance and analysis.

Using this heap, Dijkstra's algorithm takes $O(m + \sum_{x_i \in V \setminus \{s\}} (1 + \log \phi(x_i)))$ time. They [6] also prove a universal lower bound: for any fixed (directed) graph topology $G = (V, E)$ and source $s \in V$, no algorithm A' achieves runtime $o(m + n + \sum_{x_i \in V \setminus \{s\}} \log \phi(x_i))$ for all edge weightings $w \in \mathbb{W}(G)$. Dijkstra's algorithm with this heap is therefore universally optimal.

3 Timestamp optimal heaps

Instead of designing heaps with a running time proportional to the working-set size of a heap element, we use a simple timestamp approach:

▶ **Definition 2.** *For any heap H, we define* time t *as a counter that increments after each H.Push(x, y). For any element $x_i \in H$, we denote by a_i the time when x_i was pushed onto the heap, and by b_i the time when x_i was popped from the heap. Observe that $b_i > a_i$.*

▶ **Definition 3.** *A heap H is* timestamp optimal *if it supports Push and DecreaseKey in amortized constant time, and Pop with output x_i in amortized $O(1 + \log(b_i - a_i))$ time.*

Our Data Structure. Let H denote our abstract heap data structure, supporting operations H.Push(x, y), H.DecreaseKey(x, y), and H.Pop$()$. We implement H as follows:

▶ **Invariant 1** (Figure 1). *We partition the elements of H into an array of buckets B. Each bucket $B[j]$ contains one or two Fibonacci heaps. Every element of H lies in exactly one Fibonacci heap. Along with each Fibonacci heap F we store a half-open interval $I_F \subset \mathbb{R}$. We maintain the following invariants; for all times t:*

(a) *An element $x_i \in H$ is in F if and only if $a_i \in I_F$,*
(b) *The interval I_F for $F \in B[j]$ has size 2^j,*
(c) *All intervals of $B[j-1]$ lie to the right of the intervals in $B[j]$,*
(d) *All intervals collectively partition $[0, t)$.*

By Invariant 1, for all elements $x_i \in B[j]$, $(t - a_i)$ is at least 2^{j-1}, but less than 2^{j+2}. We define M as an array of size $\lceil \log n \rceil$, where $M[j]$ stores the minimum key in $B[j]$. Furthermore, we maintain a bitstring S_M where $S_M[j] = 1$ if for all $k > j$, we have $M[j] \leq M[k]$.

▶ **Lemma 4.** *For any element $x_i \in H$ we can, given a_i and t, compute the bucket containing x_i and the Fibonacci heap that contains x_i in constant time.*

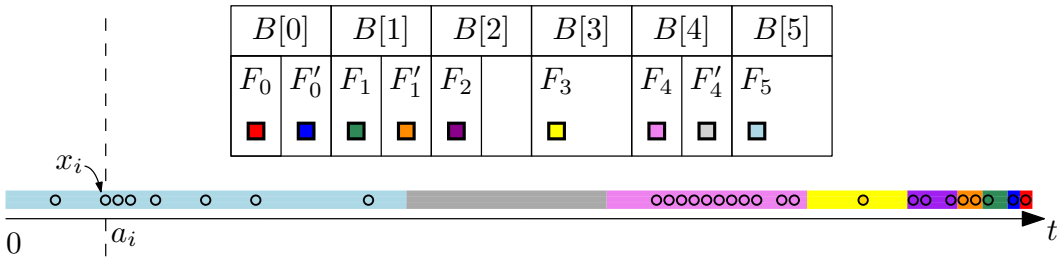


Figure 1 We illustrate our invariant. Each bucket $B[j]$ contains one or two Fibonacci heaps. Each heap F is associated with an interval on the number line in $[0, t)$. The elements x_i stored in a Fibonacci heap F are visualised by placing them at positions a_i along the number line. Importantly, when the buckets in B are indexed from left to right (i.e., from low to high indices), the corresponding intervals of the Fibonacci heaps appear in the reverse order along the number line.

Proof. Let a_i be the insertion time of x_i. We now want to determine the possible values j such that $B[j]$ contains x_i. From invariant 1.b we get: $t - a_i \geq 2^{j-1}$, or equivalently, $j \leq \log(t-a_i)+1$; and $t-a_i < 2^{j+2}$, or equivalently, $j > \log(t-a_i)-2$. In particular, for $\ell = \lfloor \log(t-a_i) \rfloor$, the Fibonacci heap containing x_i lies in one of the buckets $B[\ell-1], B[\ell], B[\ell+1]$. For each of these at most 6 corresponding intervals of the at most 6 heaps contained in those 3 buckets, we test, in constant time, whether a_i is contained in the interval. This procedure returns us the interval containing a_i, and thus, the heap (and bucket) containing x_i. ◀

▶ **Theorem 5.** *The above data structure is a timestamp optimal heap H.*

Proof. We prove that the data structure supports all three heap operations within the specified time bounds while maintaining Invariant 1.

Push(x_i, y). To push x_i onto H, we set $a_i = t$ and increment t. We then add a new Fibonacci heap F to $B[0]$ with $I_F = [a_i, a_i+1)$. Whilst a bucket $B[j]$ contains three Fibonacci heaps, we select its oldest two Fibonacci heaps F_1, F_2. We MELD F_1 and F_2 into a new Fibonacci heap F' in worst-case constant time, and we move F' to $B[j+1]$. We define the corresponding interval $I_{F'}$ as the union of the consecutive intervals I_{F_1} and I_{F_2}. Since Property (a) held for both F_1 and F_2, it now holds for F'. Moreover, $|I_{F'}| = 2^{j+1}$ and so Property (b) is maintained. Since, before the update, all intervals collectively partitioned $[0, t)$ and we only added the interval $[t, t+1)$ we maintain Properties (c) and (d). Every MELD operation decreases the number of Fibonacci heaps and every push operation creates only one heap. Hence, each push operation performs amortized $O(1)$ MELD operations.

Finally, we update M and the string S_M. After we have merged two heaps in a bucket $B[j]$, only $M[j]$ and $M[j+1]$ change. Moreover, $M[j]$ can only increase in value and $M[j+1]$ can only decrease to at least the original value of $M[j]$. Therefore, all bits $S_M[k]$ for $k \notin \{j, j+1\}$ remain unchanged and we update S_M in amortized constant time. Thus, a push operation takes amortized constant time.

DecreaseKey(x_i, y). By Lemma 4, we find the Fibonacci heap F and the bucket $B[j]$ that contain x_i in constant time. We perform F.DECREASEKEY(x_i, y) in amortized constant time. We update $M[j]$ and identify the index k of the leftmost 1-bit in S_M after index j in constant time using bitstring operations [4]. Note that $S_M[j] = 1$ if and only if $M[j] \leq M[k]$.

If $S_M[j] = 1$, then we recursively find the next $j' < j$ for which $S_M[j'] = 1$ using the same bitstring operation. If $M[j] < M[j']$, then we set $S_M[j'] = 0$ and we recurse. If $M[j] \geq M[j']$ then for all $j'' < j'$ with $S_M[j''] = 1$, $M[j] \geq M[j'']$ and our update terminates. The recursive call takes amortized constant time since it can only decrease the number of 1-bits in S_M.

Pop(). We obtain the index j of the leftmost 1-bit of S_M in constant time using bitstring operations. Per definition, the minimum key of H is in $B[j]$. We perform the Peak operation on both heaps in $B[j]$. For the heap F containing the minimum key, we do F.Pop() in $O(1 + \log 2^j) = O(j)$ time to return the element $x_i \in H$ with the minimum key. Finally, we update $M[j]$ and S_M. We update $M[j]$ in constant time and iterate from j to 1. For each iteration ℓ, let $k > \ell$ be the minimum index where $S_M[k] = 1$. We obtain k in constant time using bitstring operations and update $S_M[\ell]$ by comparing $M[\ell]$ to $M[k]$. Thus, this update takes $O(j)$ time. By Invariant 1, $\log(b_i - a_i) \in O(j)$ and thus the H.Pop() takes $O(1 + \log(b_i - a_i))$ amortized time. ◀

▶ **Corollary 6.** *Algorithm 1 can be adapted to use* $O(m + \sum_{x_i \in V \setminus \{s\}} (1 + \log(b_i - a_i)))$ *time.*

4 Universal Optimality

We show that categorizing the running time of Dijkstra's algorithm by timestamps substantially simplifies its proof for universal optimality. For a given graph $G = (V, E)$ and a designated source vertex s, we define a *linearization* L of (G, s) as any permutation of $V \setminus \{s\}$ such that there exists some choice of edge weights $w' \in \mathbb{W}(G)$ where L is the solution to the single-source-shortest path problem on (G, s, w'). Let $\ell(G, s)$ denote the number of distinct linearizations of (G, s). Observe that $\Omega(\log \ell(G, s))$ is a comparison-based universal lower bound for the single-source shortest path problem. Indeed, any comparison-based algorithm has a corresponding binary decision tree with leaves representing possible outputs. With $\Omega(\ell(G, s))$ distinct outputs, there must exist a root-to-leaf path of length $\Omega(\log \ell(G, s))$.

Our approach. We fix the input $(G = (V, E), s, w)$ and execute Dijkstra's algorithm, which defines a set of n timestamp intervals $\{[a_i, b_i]\}_{i \in [n]}$. Each interval $[a_i, b_i]$ has a minimum size of 1 and corresponds to a unique vertex $x_i \in V \setminus \{s\}$. Dijkstra's algorithm operates in $O(m + \sum_{x_i \in V \setminus \{s\}} (1 + \log(b_i - a_i)))$ time. Since any algorithm must spend $\Omega(n + m)$ time to read the input, it remains to show that $\sum_{x_i \in V \setminus \{s\}} (1 + \log(b_i - a_i)) \in \Omega(n + \log \ell(G, s))$.

▶ **Lemma 7.** *For each vertex $x_i \in V \setminus \{s\}$, choose a real value $r_i \in [a_i, b_i]$. If all r_i are distinct, then the sequence L, which orders all x_i by r_i, is a linearization of (G, s).*

Proof. Consider the initial execution of Dijkstra's algorithm on (G, s, w). For each vertex $x_i \in V \setminus \{s\}$, there is a unique edge (u, x_i) that, upon inspection, pushes the element x_i onto the heap H. The set of these edges forms a spanning tree T rooted at s.

Denote by $\text{rank}(r_i)$ the rank of r_i in the ordered sequence L. We construct a new integer edge weighting w' such that running Dijkstra's algorithm on (G, s, w') outputs L. We set $w'(e) = \infty$ for all edges $e \in E \setminus T$. We then traverse T in a breath-first manner. For each edge $(x_j, x_i) \in T$, we observe that in our original run of Dijkstra's algorithm, the algorithm popped the vertex x_j before it pushed the vertex x_i onto the heap. It follows that for the intervals $\{[a_j, b_j], [a_i, b_i]\}$, the value $b_j \leq a_i$. Since $r_j \in [a_j, b_j]$, $r_i \in [a_i, b_i]$, and $r_j \neq r_i$ we may conclude that $\text{rank}(r_i) > (r_j)$. We assign to each edge $(x_j, x_i) \in T$ the positive edge weight $w'((x_j, x_i)) = \text{rank}(r_i) - \text{rank}(r_j)$. It follows that the distance in (G, E, w') from s to any vertex x_i is equal to $\text{rank}(r_i)$ and so Dijkstra's algorithm on (G, s, w') outputs L. ◀

What remains is to show that the number of ways to select distinct $r_i \in [a_i, b_i]$, correlates with the size of the timestamp intervals. This result was first demonstrated by Cardinal, Fiorini, Joret, Jungers, and Munro [1, the first 2 paragraphs of page 8 of the ArXiV version]. It was later paraphrased in Lemma 4.3 in [5].

▶ **Lemma 8** (Originally in [1], paraphrased by Lemma 4.3 in [5]). *Let $\{[a_i, b_i]\}$ be a set of n integer intervals where each interval has size at least 1. Let Z be the set of linear orders realizable by real numbers $r_i \in [a_i, b_i]$. Then $\sum_{i\in[n]} \log(b_i - a_i) \in O(n + \log|Z|)$.*

Proof. To illustrate the simplicity of this lemma, we quote the proof in [5]: For each i between 1 and n inclusive, choose a real number r_i uniformly at random from the real interval $[0, n]$, independently for each i. With probability 1, the r_i are distinct. Let L be the permutation of $[n]$ obtained by sorting $[n]$ by r_i. Each possible permutation is equally likely. If each $r_i \in [a_i, b_i]$ then L is in Z. The probability of this happening is $\prod_{i=1}^{k}(b_i - a_i)/n$. It follows that $|Z| \geq n! \cdot \prod_{i=1}^{n}(b_i - a_i)/n$. Taking logarithms gives $\log|Z| \geq \sum_{i=1}^{n} \log(b_i - a_i) + \log n! - n\log n$. By Stirling's approximation of the factorial, $\log n! \geq n\log n - n\log e$. The lemma follows. ◀

It follows that we can recreate the main theorem from [6]:

▶ **Theorem 9** (Theorem 1.2 in [6]). *Dijkstra's algorithm, when using a sufficiently efficient heap data structure, has a universally optimal running time.*

Proof. The running time of Dijkstra's algorithm with the input (G, s, w) is $O(m)$ plus the time needed for push and pop operations. With a timestamp-optimal heap, by Corollary 6, the algorithm thus takes $O(m + \sum_{x_i \in V\setminus\{s\}} (1 + \log(b_i - a_i)))$ total time. By Lemma 8, the total running time is then $O(m + n + \log|Z|)$, where $|Z|$ denotes the number of distinct linear orders created by choosing distinct real values $r_i \in [a_i, b_i]$. By Lemma 7: $|Z| \leq \ell(G, s)$, and so Dijkstra's algorithm with any timestamp-optimal heap runs in $O(m + n + \log \ell(G, s))$ time. Conversely, any algorithm that solves SSSP needs $\Omega(m + n)$ time to read the input, and $\Omega(\log \ell(G, s))$ is an information-theoretical universal lower bound. ◀

5 Conclusion

We study Dijkstra's algorithm for the single-source-shortest path problem and give an alternative proof that Dijkstra's algorithm is universally optimal. We consider our construction to be considerably simpler than the one in [6]. We find time stamps easier to define than the working set size of an element, and regard our timestamp optimal heap to be simpler in both its construction and analysis than the heap used in [5]. As an added benefit, we only rely on Fibonacci trees and bitstring manipulation and no involved data structures (e.g., the *fusion trees* used in [6]). We regard Section 4 as our main contribution, as it gives a very concise proof of universal optimality. We understand that algorithmic simplicity is subjective, and we leave it to the reader to judge the simplicity of our approach. We note, however, that proofs are unarguably shorter since the full version of [6] counts over 50 pages. We observe that both our paper and [6] must assume a word RAM and we consider it an interesting open problem to show universal optimality on a pointer machine.

Finally, we note that our data structure is equally general to [6]. In the full version, we show that our data structure, just as the one in [6], is also universally optimal if algorithmic running time is exclusively measured by the number of edge weight comparisons that the algorithm performs. While our proof is indeed new, different, and perhaps simpler than the previous one, it is not self-contained; for this reason, we relegate it to our full version.

References

1 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M Jungers, and J Ian Munro. Sorting under partial information (without the ellipsoid algorithm). In *ACM Symposium on Theory of Computing (STOC)*, pages 359–368, 2010. doi:10.1145/1806689.1806740.

2 Edsger Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.

3 Michael Fredman and Robert Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

4 Michael Fredman and Dan Willard. Surpassing the information theoretic bound with fusion trees. *Journal of computer and system sciences*, 47(3):424–436, 1993. doi:10.1016/0022-0000(93)90040-4.

5 Bernhard Haeupler, Richard Hladík, John Iacono, Václav Rozhoň, Robert E Tarjan, and Jakub Tětek. Fast and simple sorting using partial information. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3953–3973. SIAM, 2025. doi:10.1137/1.9781611978322.134.

6 Bernhard Haeupler, Richard Hladík, Václav Rozhoň, Robert Tarjan, and Jakub Tětek. Universal optimality of dijkstra via beyond-worst-case heaps. In *Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2024. doi:10.1109/FOCS61266.2024.00125.

7 Bernhard Haeupler, Richard Hladík, Václav Rozhoň, Robert E Tarjan, and Jakub Tětek. Bidirectional dijkstra's algorithm is instance-optimal. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 202–215. SIAM, 2025. doi:10.1137/1.9781611978315.16.

8 Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *ACM Symposium on Theory of Computing (STOC)*, pages 1166–1179, 2021. doi:10.1145/3406325.3451081.

9 John Iacono. Improved upper bounds for pairing heaps. In *Scandinavian Workshop on Algorithm Theory*, pages 32–45. Springer, 2000. doi:10.5555/645900.672600.

10 John Iacono and Michael L. Fredman. *Distribution-sensitive data structures*. PhD thesis, Rutgers University, New Jersey, 2001.

11 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. Simpler optimal sorting from a directed acyclic graph. In *SIAM Symposium on Simplicity in Algorithms (SOSA)*, 2025. doi:10.1137/1.9781611978315.26.

Color Distance Oracles and Snippets: Separation Between Exact and Approximate Solutions

Noam Horowicz
Bar-Ilan University, Ramat Gan, Israel

Tsvi Kopelowitz
Bar-Ilan University, Ramat Gan, Israel

Abstract

In the *snippets* problem, the goal is to preprocess a text T so that given two pattern queries, P_1 and P_2, one can quickly locate the occurrences of the two patterns in T that are closest to each other, or report the distance between these occurrences. Kopelowitz and Krauthgamer [CPM2016] showed upper bound tradeoffs and conditional lower bounds tradeoffs for the snippets problem, by utilizing connections between the snippets problem and the problem of constructing a color distance oracle (CDO), which is a data structure that preprocess a set of points with associated colors so that given two colors c and c' one can quickly find the (distance between the) closest pair of points where one has color c and the other has color c'. However, the existing upper bound and lower bound curves are not tight.

Inspired by recent advances by Kopelowitz and Vassilevska-Williams [ICALP2020] regarding tradeoff curves for Set-disjointness data structures, in this paper we introduce new conditionally optimal algorithms for a $(1+\varepsilon)$ approximation version of the snippets problem and a $(1+\varepsilon)$ approximation version of the CDO problem, by applying fast matrix multiplication. For example, for CDO on n points in an array, if the preprocessing time is $\tilde{O}(n^a)$ and the query time is $\tilde{O}(n^b)$ then, assuming that $\omega = 2$ (where ω is the exponent of n in the runtime of the fastest matrix multiplication algorithm on two squared matrices of size $n \times n$), we show that approximate CDO can be solved with the following tradeoff

$$\begin{cases} a + 2b = 2 & \text{if } 0 \le b \le \frac{1}{3} \\ 2a + b = 3 & \text{if } \frac{1}{3} \le b \le 1. \end{cases}$$

Moreover, we prove that for exact CDO on points in an array, the algorithm of Kopelowitz and Krauthgamer [CPM2016], which obtains a tradeoff of $a + b = 2$, is essentially optimal assuming that the *strong all-pairs shortest paths* hypothesis holds for randomized algorithms. Thus, we demonstrate that the exact version of CDO is strictly harder than the approximate version. Moreover, this separation carries over to the snippets problem.

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

Keywords and phrases data structures, fast matrix multiplication, fine-grained complexity, pattern matching, distance oracles

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.72

Related Version *Full Version*: https://arxiv.org/abs/2507.04578 [12]

Funding Supported by a BSF grant 2018364, and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064).

Acknowledgements The authors thank an anonymous reviewer who suggested some ideas for tightening the lower bounds proved in Theorems 8 and 9.



© Noam Horowicz and Tsvi Kopelowitz;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

In the *snippets* problem, introduced by Kopelowitz and Krauthgamer [17], the goal is to preprocess a text T so that given two pattern queries, P_1 and P_2, one can quickly locate the locations of the two patterns in T that are closest to each other, or report the distance between these locations. The snippets problem is motivated by many common text indexing applications, such as searching a corpus of documents for two query keywords. Often, the relevance of a document to the query is measured by the proximity of the two keywords within the document. The term *snippet* is derived from search engines often providing each result with a snippet of text from the document with the two keywords close to each other.

Kopelowitz and Krauthgamer [17] designed a tradeoff algorithm for the snippets problem based on a connection to a problem on colored points in metric spaces, which we define next.

Colored points and color distances

Let $\mathbf{M}$ be a metric space with distance function $d(\cdot,\cdot)$. Let $S \subseteq \mathbf{M}$ be a set of points where $|S| = n$. Let $\mathcal{C}$ be a set of *colors*. For sake of convenience we assume that $\mathcal{C} = [|\mathcal{C}|]$. Each point $p \in S$ has an associated color $c_p \in \mathcal{C}$. For a color $c \in \mathcal{C}$, let $P_S(c) = \{p \in S | c_p = c\}$. When S is clear from context we abuse notation and denote $P(c) = P_S(c)$.

For sets of points $A, B \subset \mathbf{M}$ and a point $p \in M$, let $d(p, A) = \min_{p' \in A}\{d(p, p')\}$ and let $d(A, B) = \min_{p' \in A}\{d(p', B)\} = \min_{p' \in A, \hat{p} \in B}\{d(p', \hat{p})\}$. We extend the definition of d to inputs of colors as follows. For colors $c, c' \in \mathcal{C}$ and point $p \in \mathbf{M}$, let $d(p, c) = d(p, P(c))$ and let $\delta_{c,c'} = d(c, c') = d(P(c), P(c'))$. We say that $\delta_{c,c'}$ is the *color distance* between c and c'. A natural problem is to construct a *color distance oracle* (CDO), which is defined as follows.

▶ **Problem 1** (The Color Distance Oracle (CDO) problem [17]). *Given a set $S \subseteq \mathbf{M}$ of n colored points with colors from $\mathcal{C}$, preprocess S to support the following query: given $c, c' \in \mathcal{C}$, return $\delta_{c,c'}$.*

In this paper we consider metrics defined by locations in an array of size s, and so the metric is the set[1] $[s]$, and the distance function of two points p and q is $d(p, q) = |p - q|$.

Multi-colored points and color hierarchies

A natural generalization of a colored set is to allow each $p \in S$ to be associated with a nonempty set of colors $c(p) \subseteq \mathcal{C}$, and in such a case S is said to be *multi-colored*. This version of the CDO problem is called the *multi-color distance oracle* (MCDO) problem. In general, representing $c(p)$ costs $\Theta(|c(p)|)$ space by explicitly listing all colors in $c(p)$. Thus, the input size is $n = \sum_{p \in S} |c(p)|$.

Nevertheless, there are interesting cases in which the lists of colors for each point are not required to be given explicitly. One such example, which we focus on, is when for every two colors $c, c' \in \mathcal{C}$, either one of the sets $P(c)$ and $P(c')$ contains the other, or the two sets are disjoint. In such a case, we say that $\mathcal{C}$ has a *color hierarchy* with respect to S (a formal terminology is that $\{P(c)\}_{c \in \mathcal{C}}$ is a laminar family), represented by a rooted forest (i.e., each tree has a root) T_S of size $O(|\mathcal{C}|)$. Each color c is associated with a vertex u_c in T_S, such that the descendants of u_c are exactly all the vertices $u_{c'}$ whose color c' satisfies $P(c') \subseteq P(c)$. We convert the forest T_S to a rooted tree by adding a dummy root vertex and making the dummy root the parent of all of the roots of the trees in the forest.

[1] Throughout this paper for a positive integer k we denote $[k] = \{1, 2, , \ldots, k\}$.

With the aid of T_S, a multi-colored set S that admits a color hierarchy can be represented using only $O(n + |\mathcal{C}|)$ machine words, because it suffices to store T_S and associate with each point p just one color c (the color with the lowest corresponding vertex in T_S); the other colors of p are implicit from T_S (the colors on the path from u_c to the root of T_S). Notice that $|\mathcal{C}| > n$ implies that there are at least two colors with the same point set. Thus, we make the simplifying assumption that all colors have different point sets[2], and so the input size is $O(n)$.

Thus, a natural extension of Problem 1 is the following.

▶ **Problem 2** (The Multi-Color Distance Oracle with a Color Hierarchy (MCDOCH) problem [17]). *Given a set $S \subseteq \mathbf{M}$ of n multi-colored points with colors from $\mathcal{C}$ which form a color hierarchy with respect to S, preprocess S to support the following query: given $c, c' \in \mathcal{C}$, return $\delta_{c,c'}$.*

Let a and b be real numbers such that the preprocessing and query times of a MCDOCH algorithm are $\tilde{O}(n^a)$[3] and $\tilde{O}(n^b)$, respectively. A tradeoff algorithm for the MCDOCH problem was presented in [17], where given parameter $1 \leq \tau \leq n$, the query time is $\tilde{O}(n^b) = \tilde{O}(\tau)$ and the preprocessing time is $\tilde{O}(n^a) = \tilde{O}(n^2/\tau) = \tilde{O}(n^{2-b})$. When ignoring sub-polynomial factors, this upper bound tradeoff is given by $a + b = 2$ where $0 \leq b \leq 1$ and $1 \leq a \leq 2$. Notice that the same tradeoff applies to the CDO problem.

In addition to the tradeoff algorithms, [17] proved a 3SUM based conditional lower bound (CLB) tradeoff[4] of $a + 2b \geq 2$ for the CDO problem (and hence for the MCDOCH problem), even when $\mathbf{M}$ is defined by locations in an array. Their CLB holds also for approximate versions of the CDO problem (see Theorem 8 in [17]), where for a fixed $\alpha > 1$, the answer to a color distance query between c and c' is required to be between $\delta_{c,c'}$ and $\alpha \cdot \delta_{c,c'}$.

Solving the Snippets problem

By designing a reduction from the snippets problem to the MCDOCH problem, [17] were able to design an algorithm for the snippets problem with essentially the same tradeoff: for a chosen parameter $1 \leq \tau \leq |T|$ their snippets algorithm uses $\tilde{O}(|T|^2/\tau)$ preprocessing time and answers queries in $\tilde{O}(|P_1| + |P_2| + \tau)$ time. The main idea in the reduction is to construct the suffix tree of T, assign a color to each vertex in the suffix tree, and for each leaf ℓ in the suffix tree, the corresponding location in T is assigned all of the colors on the path from ℓ to the suffix tree root. It is straightforward to see that the colors define a hierarchy and that T_S is exactly the suffix tree.

A complexity gap

The upper bound curve of $a+b = 2$ and the CLB tradeoff of $a+2b \geq 2$ form a complexity gap for the snippets and (approximate) CDO problems, which was left open by [17]. Moreover, based on the combinatorial Boolean Matrix Multiplication (BMM) hypothesis, [17, Theorem 8] implies that any algorithm that beats the $a + b = 2$ curve, even for approximate versions, must use non-combinatorial techniques, such as fast matrix multiplication (FMM) algorithms.

We remark that the CLB tradeoff given in [17] is based on a reduction from a SetDisjointness problem to the (approximate) CDO problem, and at the time [17] was published, the SetDisjointness problem was known to exhibit the same upper bound tradeoff and 3SUM

[2] If we happen to have several colors with the same point set, we ignore all of them but one during preprocessing, and use a straightforward mapping from the original coloring to the reduced coloring during query time.

[3] Throughout this paper we use the notation $\tilde{O}()$ to suppress sub-polynomial factors.

[4] The CLB tradeoff stated here is straightforward to derive from the statement of Theorem 4 in [17].

based CLB tradeoff [18]. However, recently Kopelowitz and Vassilevska-Williams [19] closed this complexity gap by, among other ideas, using FMM techniques. Thus, a natural question to ask is whether FMM can assist in obtaining improved algorithms for CDO problems.

Our results

We close the complexity gap for the $(1+\varepsilon)$ approximation versions of CDO and $MCDOCH$, where $\mathbf{M}$ is defined by locations in an array. Formally, the problems are defined as follows.

▶ **Problem 3** (The $(1+\varepsilon)$-Approximate Color Distance Oracle (ACDO) problem on an array)**.** *Let $\mathbf{M}$ be a metric defined by locations in an array of size n. Given a set $S \subset \mathbf{M}$ of n colored points with colors from $\mathcal{C}$ and a fixed real $0 \leq \varepsilon \leq 1$, preprocess S to support the following query: given two colors $c, c' \in \mathcal{C}$, return a value $\hat{\delta}$ such that $\delta_{c,c'} \leq \hat{\delta} \leq (1+\varepsilon)\delta_{c,c'}$.*

▶ **Problem 4** (The $(1+\varepsilon)$-Approximate Multi-Color Distance Oracle problem with a Color Hierarchy (AMCDOCH) on an array)**.** *Let $\mathbf{M}$ be a metric defined by locations in an array of size $\Theta(n)$. Given a set $S \subseteq \mathbf{M}$ of n multi-colored points with colors from $\mathcal{C}$ which form a color hierarchy with respect to S, and a fixed real $0 \leq \varepsilon \leq 1$, preprocess S to support the following query: given two colors $c, c' \in \mathcal{C}$, return a value $\hat{\delta}$ such that $\delta_{c,c'} \leq \hat{\delta} \leq (1+\varepsilon)\delta_{c,c'}$.*

Our first main result is a new tradeoff algorithm for ACDO, which is stated by the following theorem. Notice that the time complexities presented are dependent on $\omega \geq 2$, which is the exponent of n in the runtime of the fastest FMM algorithm on two squared matrices of size $n \times n$. The currently best upper bound on ω is $\omega < 2.371339$ given by [2]. However, for clarity, we choose to express our tradeoffs in terms of ω. We remark that ACDO is a special case of AMCDOCH, and so Theorem 6 implies Theorem 5. However, Theorem 5 is used in the proof of Theorem 6, so we explicitly state both theorems.

▶ **Theorem 5.** *For any real $0 \leq b \leq 1$, there exists an ACDO algorithm with preprocessing time $\tilde{O}(n^a)$ and query time $\tilde{O}(n^b)$ where*

$$\begin{cases} a + \dfrac{2}{\omega-1}b = 2 & \text{if } 0 \leq b \leq \dfrac{\omega-1}{\omega+1} \\ \dfrac{2}{\omega-1}a + b = \dfrac{\omega+1}{\omega-1} & \text{if } \dfrac{\omega-1}{\omega+1} \leq b \leq 1. \end{cases}$$

▶ **Theorem 6.** *For any real $0 \leq b \leq 1$, there exists an AMCDOCH algorithm with preprocessing time $\tilde{O}(n^a)$ and query time $\tilde{O}(n^b)$ where*

$$\begin{cases} a + \dfrac{2}{\omega-1}b = 2 & \text{if } 0 \leq b \leq \dfrac{\omega-1}{\omega+1} \\ \dfrac{2}{\omega-1}a + b = \dfrac{\omega+1}{\omega-1} & \text{if } \dfrac{\omega-1}{\omega+1} \leq b \leq 1. \end{cases}$$

Combining Theorem 6 with the reduction of the snippets problem to the MCDOCH problem given in [17], we obtain the following tradeoff for a $1+\varepsilon$ approximation version of the snippets problem.

▶ **Theorem 7.** *For any fixed $0 < \varepsilon \leq 1$, and $1 \leq a \leq 2$, there exists an algorithm that preprocesses a text T in $\tilde{O}(|T|^a)$ time such that given two pattern strings P_1 and P_2 where the distance between the closest occurrences of the two patterns in T is δ, the algorithm returns a value $\hat{\delta}$ such that $\delta \leq \hat{\delta} \leq (1+\varepsilon)\delta$ in $\tilde{O}(|P_1| + |P_2| + |T|^b)$ time, where*

$$\begin{cases} a + \dfrac{2}{\omega-1}b = 2 & \text{if } 0 \leq b \leq \dfrac{\omega-1}{\omega+1} \\ \dfrac{2}{\omega-1}a + b = \dfrac{\omega+1}{\omega-1} & \text{if } \dfrac{\omega-1}{\omega+1} \leq b \leq 1. \end{cases}$$

We remark that it is straightforward to adapt our ACDO and AMCDOCH algorithms (and hence our new approximate snippets algorithm) to return the two points (one of each color) that define the distance being returned (or the locations of the two patterns in the approximate snippets problem). For details, see the full version [12].

A complexity separation between exact and approximate solutions

We remark that by combining the reduction given in [17, Theorem 8] from SetDisjointness to the CDO problem with the CLBs for SetDisjointness given in [19, Theorem 6] for the case of $\omega = 2$ (which some researchers believe should be obtainable), Theorems 5, 6 and 7 are all conditionally optimal[5] (up to subpolynomial factors). A natural question to ask is whether one can design exact algorithms with the same tradeoff bounds. We provide evidence that this is not possible, based on a *strong* version of the popular all-pairs shortest paths (APSP) conjecture [7], thereby demonstrating a separation between CDO and ACDO.

The Strong-APSP hypothesis, introduced by Chan, Vassilevska-Williams and Xu [7], states that for a graph with $\hat{n}$ vertices[6], even if all of the edge weights are in[7] $[\hat{n}]$, APSP still does not have a truly subcubic algorithm. A closely related problem to APSP is the $(min, +)$-*matrix product* problem, where the input is two $\hat{n}$ by $\hat{n}$ matrices $A = \{a_{i,j}\}$ and $B = \{b_{i,j}\}$, and the output is the matrix $D = \{d_{i,j}\}$ where $d_{i,j} = \min_{1 \leq k \leq \hat{n}}(a_{i,k} + b_{k,j})$. Shoshan and Zwick [24] showed that solving APSP with weights $[M]$ is equivalent (in the sense of having the same runtime) to solving $(min, +)$-matrix product with entries in $[O(M)]$. Thus, the Strong-APSP hypothesis implies that there is no truly subcubic time $(min, +)$-matrix product algorithm even when all of the entries are in $[\hat{n}]$.

By reducing $(min, +)$-matrix product with entries in $[\hat{n}]$ to MCDO, we are able to prove the following CLB in Section 6, which matches the upper bound given in [17].

▶ **Theorem 8.** *Assuming the Strong-APSP conjecture, any algorithm for MCDO on n points in an array of size $O(n)$ with $\tilde{O}(n^a)$ preprocessing time and $\tilde{O}(n^b)$ query time, respectively, must obey $a + b \geq 2$.*

Moreover, we are able to leverage the ideas used in the proof of Theorem 8 to reduce $(min, +)$-matrix product with entries in $[\hat{n}]$ to CDO via a randomized reduction, thereby obtaining a CLB under the assumption that the Strong-APSP hypothesis holds even for randomized algorithms (either in expectation or with high probability).

▶ **Theorem 9.** *Assuming the Strong-APSP hypothesis for randomized algorithms, any algorithm for CDO on n points in an array of size $O(n)$ with $\tilde{O}(n^a)$ preprocessing time and $\tilde{O}(n^b)$ query time, respectively, must obey $a + b \geq 2$.*

The proof of Theorem 9 is based on the proof of Theorem 8, combined with probabilistic techniques in order to create instances that are not multi-colored. Due to space considerations, the proof is given in the full version [12]

[5] For Theorems 5 and 6, the optimality follows directly from applying [17, Theorem 8] and [19, Theorem 6]. For Theorem 7, the optimality follows since an approximate solution for the snippets problem can be used to solve ACDO on an array as follows: treat each color as a character, and then the array becomes a string. Preprocess the string using the snippets algorithm, so that given an ACDO query on two colors $c, \hat{c}$, query the snippets algorithm with patterns $P = c$ and $\hat{P} = \hat{c}$. Thus, the ACDO lower bound applies to approximate snippets.

[6] We use $\hat{n}$ to differentiate from $n = |S|$ in the various CDO problems.

[7] Actually, in [7] the weights are bounded by $\hat{n}^{3-\omega}$, but since $\omega \geq 2$ we can bound the weights by $\hat{n}$.

Theorem 9 combined with Theorem 5 implies that the exact version of CDO is strictly harder than the approximate version. Moreover, we remark that the CLBs for CDO apply also to the snippets problem, since the special case of $\mathbf{M}$ defined by locations in an array is captured by the snippets problem when P_1 and P_2 are each a single character. Thus, the CLB of Theorem 9 applies also to the snippets problem, and so the exact version of the snippets problem is also strictly harder than the approximate version.

Additional Related Work

A problem closely related to the CDO is the (approximate) vertex-labeled distance oracles for graphs (VLDO) problem, where the goal is to preprocess a colored graph G with n vertices, so that given a vertex query v and a color c, one can quickly return (an approximation of) $d(v, c)$. Hermelin, Levy, Weimann, and Yuster [11] introduced the VLDO problem and designed a solution using $O(kn^{1+1/k})$ expected space, with stretch factor $4k-5$ and $O(k)$ query time. They also showed how to reduce the space usage to $O(kN^{1/k})$ but the stretch factor is exponential in $2k-21$. Chechik [8] later showed how to lower the stretch back to $4k-5$. For planar graphs, Evald, Fredslund-Hansen, and Wulff-Nilsen [10] designed a near-optimal exact tradeoff using $n^{1+o(1)}$ space with $\tilde{O}(1)$ query time, or $\tilde{O}(n)$ space with $n^{o(1)}$ query time. Li, Ma, and Ning [21] designed a $1+\varepsilon$ approximation for planar graphs.

2 Preliminaries and Algorithmic Overview

Let $[n] = \{1, 2, \ldots, n\}$. Let $S \subset \mathbb{Z}$ be a set of n integers. Let $\max(S)$ be the largest integer in S and let $\min(S)$ be the smallest integer in S. For integers $i, j \in [n]$ where $i < j$, let $S[i, j] = \{p \in S : i \leq p \leq j\}$.

Our algorithms make use of the following Nearest Neighbor Search (NNS) data structures.

▶ **Problem 10** (The Nearest Neighbour Search problem [1, 4, 16, 25]). *Given a set $S \subseteq \mathbf{M}$ of size n, preprocess S to support the following query: given an integer p, return* $\operatorname{argmin}_{p' \in S}\{d(p, p')\}$.

▶ **Problem 11** (The Range Nearest Neighbour Search (RNNS) problem [5, 6, 9, 14, 15, 20, 22, 23]). *Given a set S of n integers, preprocess S to support the following query: given $i, j \in [n]$ and an integer p, return* $\operatorname{argmin}_{p' \in S[i,j]}\{d(p, p')\}$.

2.1 Algorithmic Overview

Generic (Approximate) CDO Algorithm

Our algorithm for ACDO is based on a generic algorithm whose structure follows the structure of the algorithm in [17] for the exact CDO problem. The generic algorithm is a classic heavy-light algorithm; our algorithms are a new *implementation* of the generic algorithm.

For an integer parameter $0 \leq \tau \leq n$, a color $c \in \mathcal{C}$ is said to be *heavy* if $\left|P(c)\right| \geq \tau$ and *light* otherwise. Let $\mathcal{H} = \{h_1, h_2, \ldots, h_{|\mathcal{H}|}\}$ be the set of heavy colors, and let $\mathcal{L}$ be the set of light colors. Notice that $|\mathcal{H}| \leq \frac{n}{\tau}$. In the preprocessing phase, for each color $c \in \mathcal{C}$, the algorithm stores $P(c)$ in an NNS data structure, denoted by NNS_c. In addition, the algorithm pre-computes a matrix $E^* = \{e^*_{i,j}\}$ of size $|\mathcal{H}| \times |\mathcal{H}|$, such that $\delta_{h_i,h_j} \leq e^*_{i,j} \leq (1+\varepsilon)\delta_{h_i,h_j}$. An ACDO query on $c, c' \in \mathcal{C}$ is processed as follows. If both $c, c' \in \mathcal{H}$, then, without loss of generality, $c = h_i$ and $c' = h_j$. In such a case the algorithm returns $e^*_{i,j}$, which is a $(1+\varepsilon)$ approximation of δ_{h_i,h_j}. Otherwise, without loss of generality, C is a light color, and the algorithm returns $\min_{\hat{p} \in P(c)}\{d(\hat{p}, c')\} = \delta_{c,c'}$, by performing $\left|P(c)\right| \leq \tau$ NNS queries on $\mathrm{NNS}_{c'}$, one for each point in C.

Time complexity

The preprocessing and query time costs of the generic algorithm depend on the implementation of the NNS data structure, and the time used for computing matrix E^*. Thus, we express the time complexity of the generic algorithm as a function of $T_{p,\mathrm{NNS}}(t)$, $T_{q,\mathrm{NNS}}(t)$, and $T_{E^*}(\mathcal{H})$, which are the preprocessing time and query time of the NNS data structure on a set of size t, and the time used to compute E^*, respectively. In the preprocessing phase, the algorithm computes E^* and creates an NNS data structure for each color in $\mathcal{C}$. Thus, the preprocessing time cost is

$$O(T_{E^*}(\mathcal{H}) + \sum_{c\in\mathcal{C}} T_{p,\mathrm{NNS}}(P(c))).$$

In the query phase, the time cost is dominated by executing at most τ NNS queries on a set of size at most n, and so the time cost is $O(\tau \cdot T_{q,\mathrm{NNS}}(n))$.

Constructing E^* and solving ACDO on an array

In Section 3 we describe an efficient algorithm for constructing E^* when $\mathbf{M}$ is the metric of integers. Our algorithm is structured as follows. For pairs of heavy colors whose color distance is smaller than some threshold, the algorithm computes their distance via a straightforward brute-force computation. The more challenging part is dealing with pairs of heavy colors with larger color distance. Our approach is to compute a series of Boolean matrices with specially designed properties, so that after the computation of the matrices we are able scan the matrices and deduce a $(1+\varepsilon)$ approximation for every pair of colors with a rather large distance color. We remark that to compute the matrices used for the approximations, our algorithm makes use of FMM, which, as noted in Section 1, is required for obtaining improvements in runtime compared to the exact solution of [17].

To complete the proof of Theorem 5, in Section 4 we analyze the runtime of the generic algorithm with the runtime of constructing E^* when $\mathbf{M}$ is defined by locations in an array, and by plugging in known NNS data structures.

Solving AMCDOCH on an array

In Section 5 we prove Theorem 6. Our algorithm is a reduction from AMCDOCH on an array to ACDO on an array. The structure of our reduction somewhat resembles the structure of the MCDOCH algorithm of [17]. In particular, in the algorithm of [17] they partition the points into sets of size τ, preprocess each set into an RNNS data structure, and for each pair of sets apply τ RNNS queries to compute the distance between the sets. To obtain a faster runtime, we compute the distance between every pair of sets by recoloring the input points using the sets as a new color scheme and then applying our ACDO algorithm with the new color set. One important property of our ACDO algorithm is that when a query takes place between two heavy colors, the query time is constant since the algorithm just looks up a single value in E^*. It turns out that essentially all of the colors in the new color set are heavy, and so after the preprocessing phase of our ACDO algorithm, we are able to compute the distance between a pair of sets in $O(1)$ time. This property turns out to be helpful for our AMCDOCH algorithm, which is described in Section 5.

3 Computing E^* when $\mathbf{M}$ is integers

In this section, we show how to compute matrix E^* when our metric $\mathbf{M}$ is one dimensional, and for $p, p' \in \mathbf{M}$ we have $d(p, p') = |p - p'|$. We make the simplifying assumption that $\min(S) = 1$ since, otherwise, one could shift S by subtracting $\min(S) - 1$ from each point.

Let W be a positive integer parameter that will be determined later. The construction algorithm for E^* has two conceptual parts. The first part deals with pairs of heavy colors with distance at most $\frac{1+2\varepsilon}{\varepsilon}W$, by computing the distances exactly using a brute-force method. The second part deals with pairs of heavy colors with distance greater than $\frac{1+2\varepsilon}{\varepsilon}W$. In this case, the algorithm computes an approximation of the distances by utilizing a series of matrices defined as follows.

Let $\ell_0 = \lfloor \log_{1+\varepsilon}(\frac{1}{\varepsilon}) \rfloor + 1$ and let $\ell_{max} = \lceil \log_{1+\varepsilon}(\max(S)) \rceil$. Let $A = \{a_{i,j}\}$ be a Boolean $|\mathcal{H}| \times \lceil \frac{\max(S)}{W} \rceil$ matrix where

$$a_{i,j} = \begin{cases} 1 & P(h_i) \cap S[(j-1)W + 1, jW] \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Thus, matrix A indicates for each heavy color and each block of size W in S which starts at a location following an integer multiple of W, whether the heavy color appears in the block or not.

For $\ell_0 \leq \ell \leq \ell_{max}$, let $B^{(\ell)} = \{b^{(\ell)}_{i,j}\}$ be a Boolean $\lceil \frac{\max(S)}{W} \rceil \times |\mathcal{H}|$ matrix where

$$b^{(\ell)}_{i,j} = \begin{cases} 1 & P(h_j) \cap S[(i-1)W + 1, iW + (1+\varepsilon)^\ell W] \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Matrix $B^{(\ell)}$ indicates for each block of size $W + (1+\varepsilon)^\ell W$ in S which starts at a location following an integer multiple of W, and for each heavy color, whether the heavy color appears in the block or not.

Let $E^{(\ell)} = A \cdot B^{(\ell)} = \{e^{(\ell)}_{i,j}\}$, where the product is a Boolean matrix product. Thus, $E^{(\ell)}$ roughly indicates for each pair of heavy colors whether their color distance is at most $W + (1+\varepsilon)^\ell W$.

In Section 3.1 we state and prove lemmas regarding the connections between the matrices $E^{(\ell)}$ and approximating color distances. In Section 3.2 we describe the algorithm for constructing E^*, and prove its correctness based on the lemmas of Section 3.1.

3.1 Properties of The $E^{(\ell)}$ Matrices

The following lemma describes a connection between entries of 0 in E^ℓ and lower bounds on color distances.

▶ **Lemma 12.** *For $h_i, h_j \in \mathcal{H}$ and integer $\ell_0 \leq \ell \leq \ell_{max}$, if $e^{(\ell)}_{i,j} = 0$ and $e^{(\ell)}_{j,i} = 0$, then $\delta_{h_i,h_j} > (1+\varepsilon)^\ell W$.*

Proof. Let $p \in P(h_i)$ and $p' \in P(h_j)$ be chosen such that $\delta_{h_i,h_j} = d(p, p')$. Assume towards a contradiction that $\delta_{h_i,h_j} \leq (1+\varepsilon)^\ell W$. We focus on the case of $p' \geq p$ and so $\delta_{h_i,h_j} = p' - p$; the proof for the case $p > p'$ is symmetrical. Thus, there exists an integer $1 \leq k \leq \lceil \frac{\max(S)}{W} \rceil$ such that $p \in S[(k-1)W + 1, kW]$ and so $a_{i,k} = 1$. Therefore,

$$p' = p + \delta_{h_i,h_j} \leq p + (1+\varepsilon)^\ell W \leq kW + (1+\varepsilon)^\ell W.$$

We conclude that $p' \in S[(k-1)W + 1, kW + (1+\varepsilon)^\ell W]$, implying that $b^{(\ell)}_{k,j} = 1$, and so $e^{(\ell)}_{i,j} = 1$, which is a contradiction. ◀

The following lemma shows a connection between colors with distance greater than $(\frac{1+2\varepsilon}{\varepsilon})W$ and entries in $E^{(\ell_0)}$ whose value is 0.

▶ **Lemma 13.** *For $h_i, h_j \in \mathcal{H}$, if $\delta_{h_i,h_j} > (\frac{1+2\varepsilon}{\varepsilon})W$ then $e^{(\ell_0)}_{i,j} = 0$ and $e^{(\ell_0)}_{j,i} = 0$.*

Proof. Assume by contradiction that $e^{(\ell_0)}_{i,j} = 1$ or $e^{(\ell_0)}_{j,i} = 1$. We focus on the case of $e^{(\ell_0)}_{i,j} = 1$; the proof for the case $e^{(\ell_0)}_{j,i} = 1$ is symmetrical. Since $e^{(\ell_0)}_{i,j} = 1$, there exists an integer $1 \leq k \leq \lceil \frac{\max(S)}{W} \rceil$ such that $a_{i,k} = 1$ and $b^{(\ell_0)}_{k,j} = 1$. Therefore, there exist points $p \in P(h_i)$ and $p' \in P(h_j)$ such that $p \in S[(k-1)W+1, kW]$ and $p' \in S[(k-1)W+1, kW+(1+\varepsilon)^{(\ell_0)}]$. Thus,

$$\begin{aligned}
\delta_{h_i,h_j} &= \min_{\hat{p}\in P(c), \tilde{p}\in P(c')} \{d(\hat{p},\tilde{p})\} \leq d(p,p') = \max\left(p-p', p'-p\right) \\
&= \max\left(W-1, (1+\varepsilon)^{(\ell_0)}W + W - 1\right) = (1+\varepsilon)^{(\ell_0)}W + W - 1 \\
&= (1+\varepsilon)^{\lfloor \log_{1+\varepsilon}(\frac{1}{\varepsilon})\rfloor+1}W + W - 1 \leq ((1+\varepsilon)^{\log_{1+\varepsilon}(\frac{1}{\varepsilon})+1} + 1)W - 1 \\
&< (\frac{1+2\varepsilon}{\varepsilon})W.
\end{aligned}$$

◀

The following lemma describes a connection between entries of 1 in E^ℓ and upper bounds on color distances.

▶ **Lemma 14.** *For $h_i, h_j \in \mathcal{H}$ and integer $\ell_0 \leq \ell \leq \ell_{\max}$, if $e^{(\ell)}_{i,j} = 1$, then $\delta_{h_i,h_j} < (1+\varepsilon)^{\ell+1}W$.*

Proof. If $e^{(\ell)}_{i,j} = 1$, then there exists an integer $1 \leq k \leq \lceil \frac{\max(S)}{W} \rceil$ such that $a_{i,k} = 1$ and $b^{(\ell)}_{k,j} = 1$. Hence, there exist $p \in P(h_i)$ and $p' \in P(h_j)$ such that $p \in S[(k-1)W+1, kW]$ and $p' \in S[(k-1)W+1, kW+(1+\varepsilon)^{\ell}W]$. Thus,

$$\begin{aligned}
\delta_{h_i,h_j} &\leq \left|p-p'\right| = \max(p-p', p'-p) \leq \max\left(W-1, (1+\varepsilon)^{\ell}W + W - 1\right) \\
&= (1+\varepsilon)^{\ell}W + W - 1 < ((1+\varepsilon)^{\ell} + 1)W.
\end{aligned}$$

Notice that $\lfloor \log_{1+\varepsilon}(\frac{1}{\varepsilon})\rfloor + 1 > \log_{1+\varepsilon}(\frac{1}{\varepsilon})$. Finally, since $\lfloor \log_{1+\varepsilon}(\frac{1}{\varepsilon})\rfloor + 1 = \ell_0 \leq \ell$, we have

$$(1+\varepsilon)^{\ell} \geq (1+\varepsilon)^{\lfloor \log_{1+\varepsilon}(\frac{1}{\varepsilon})\rfloor+1} > (1+\varepsilon)^{\log_{1+\varepsilon}(\frac{1}{\varepsilon})} = \frac{1}{\varepsilon},$$

and so $1 < \varepsilon(1+\varepsilon)^{\ell}$. Thus,

$$((1+\varepsilon)^{\ell} + 1)W < ((1+\varepsilon)^{\ell} + \varepsilon(1+\varepsilon)^{\ell})W = ((1+\varepsilon)^{\ell+1})W,$$

completing the proof. ◀

The following lemma shows that $e^{(\ell)}_{i,j}$ is weakly monotone increasing with respect to ℓ.

▶ **Lemma 15.** *For $h_i, h_j \in \mathcal{H}$, and $\ell_0 \leq \ell \leq \ell_{max}$, if $e^{(\ell)}_{i,j} = 1$, then for any $\ell < \ell' \leq \ell_{max}$ we have $e^{(\ell')}_{i,j} = 1$.*

Proof. If $e^{(\ell)}_{i,j} = 1$, then there exists an integer $1 \leq k \leq \lceil \frac{\max(S)}{W} \rceil$ such that $a_{i,k} = 1$ and $b^{(\ell)}_{k,j} = 1$. Thus, there exists $p \in P(h_i)$ such that $p \in S[(k-1)W+1, kW+(1+\varepsilon)^{\ell}W]$. Moreover, since $\ell < \ell'$, we have $S[(k-1)W+1, kW+(1+\varepsilon)^{\ell}W] \subseteq S[(k-1)W+1, kW+(1+\varepsilon)^{\ell'}W]$, and so $b^{(\ell')}_{k,j} = 1$. Finally, since $a_{i,k} = 1$, we conclude that $e^{(\ell')}_{i,j} = 1$. ◀

The following lemma shows that in $E^{(\ell_{max})}$, for each pair of heavy colors, there exists at least one corresponding 1 entry.

▶ **Lemma 16.** *For $h_i, h_j \in \mathcal{H}$, either $e_{i,j}^{(\ell_{max})} = 1$ or $e_{j,i}^{(\ell_{max})} = 1$.*

Proof. Assume by contradiction that $e_{i,j}^{(\ell_{max})} = 0$ and $e_{j,i}^{(\ell_{max})} = 0$. Let $p \in P(h_i)$ and $p' \in P(h_j)$ be chosen such that $\delta_{h_i,h_j} = d(p,p')$. Hence by Lemma 12, we have $\delta_{h_i,h_j} > (1+\varepsilon)^{\ell_{max}} W = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} \frac{\max(S)}{W} \rceil} W$. Notice that $\max(S) \geq \delta_{h_i,h_j}$. Therefore,

$$\max(S) \geq \delta_{h_i,h_j} > (1+\varepsilon)^{(\lceil \log_{1+\varepsilon} \frac{\max(S)}{W} \rceil)} W \geq (1+\varepsilon)^{\log_{1+\varepsilon} \frac{\max(S)}{W}} W \geq \max(S),$$

which is a contradiction. ◀

In the following lemma we show the main property of the matrices $E^{(\ell)}$ used in the algorithm of Section 3.2 when dealing with pairs of heavy colors with color distance greater than $\frac{1+2\varepsilon}{\varepsilon} W$.

▶ **Lemma 17.** *For $h_i, h_j \in \mathcal{H}$, if $\delta_{h_i,h_j} > \frac{1+2\varepsilon}{\varepsilon} W$, then there exists a unique integer $\ell_0 \leq \ell \leq \ell_{max} - 1$ such that $e_{i,j}^{(\ell)} = e_{j,i}^{(\ell)} = 0$ and either $e_{i,j}^{(\ell+1)} = 1$ or $e_{j,i}^{(\ell+1)} = 1$.*

Proof. By Lemma 16, at least one of $e_{i,j}^{(\ell_{max})}$ and $e_{j,i}^{(\ell_{max})}$ must be 1. So assume without loss of generality that $e_{i,j}^{(\ell_{max})} = 1$. Since $\delta_{h_i,h_j} > \frac{1+2\varepsilon}{\varepsilon} W$, by Lemma 13 $e_{i,j}^{(\ell_0)} = 0$. Thus, by Lemma 15, there must exist exactly one $\ell_0 \leq \ell < \ell_{max}$ such that $e_{i,j}^{(\ell)} = 0$ and $e_{i,j}^{(\ell+1)} = 1$.

To complete the proof we show uniqueness. If $e_{i,j}^{(\ell_{max})} = e_{j,i}^{(\ell_{max})} = 1$, then assume towards a contradiction that there exist two different integers $\ell' \neq \hat{\ell}$ such that $e_{i,j}^{(\ell')} = 0$, $e_{j,i}^{(\ell')} = 0$, $e_{i,j}^{(\ell'+1)} = 1$, $e_{i,j}^{(\hat{\ell})} = 0$, $e_{j,i}^{(\hat{\ell})} = 0$, and $e_{j,i}^{(\hat{\ell}+1)} = 1$. Without loss of generality, $\ell' < \hat{\ell}$, and so by Lemma 15, $e_{i,j}^{(\ell'+1)} = 1$ and $\ell' < \hat{\ell}$ implies $e_{i,j}^{(\hat{\ell})} = 1$ which is a contradiction.

Otherwise, if at least one of $e_{i,j}^{(\ell_{max})}$ or $e_{j,i}^{(\ell_{max})}$ is 0, then since we assumed without loss of generality that $e_{i,j}^{(\ell_{max})} = 1$, we have $e_{j,i}^{(\ell_{max})} = 0$, which together with Lemma 15 implies that for all $\ell_0 \leq \ell' < \ell_{max}$, $e_{j,i}^{(\ell')} = 0$. Thus, ℓ is unique. ◀

3.2 Algorithm for Computing E^*

In this section, we describe the algorithm for computing E^*, prove its correctness and analyze its time cost. Pseudocode for the algorithm is given in Algorithm 1. The algorithm begins by a brute-force exact computation of color distances for all heavy colors h_i and h_i where $\delta_{h_i,h_j} \leq \frac{1+2\varepsilon}{\varepsilon} W$. The brute-force computation costs $O((\frac{1+2\varepsilon}{\varepsilon})Wn) = O(\frac{Wn}{\varepsilon})$ time.

The rest of the algorithm deals with the case of $\delta_{h_i,h_j} > \frac{1+2\varepsilon}{\varepsilon} W$. First, the algorithm computes the matrices $E^{(\ell)}$ from matrices A and $B^{(\ell)}$. The time cost of computing all $E^{(\ell)}$ is dominated by the cost of $\log_{1+\varepsilon}(\max(S)) = \Theta(\frac{\log \max(S)}{\varepsilon})$ matrix multiplications. Each multiplication is between a matrix of size $\frac{n}{\tau} \times \frac{\max(S)}{W}$ and a matrix of size $\frac{\max(S)}{W} \times \frac{n}{\tau}$. It is folklore knowledge that the matrix multiplication of two matrices of size $x \times y$ and $y \times z$ can be computed in $O\left(\frac{x \cdot y \cdot z}{\min(x,y,z)^{3-\omega}}\right)$. Thus, the time cost of computing the matrix multiplications is $O\left(\frac{n^2 \cdot \max(S)}{\tau^2 W \cdot \min(\frac{n}{\tau}, \frac{\max(S)}{W})^{3-\omega}} \frac{\log \max(S)}{\varepsilon}\right)$ time.

Finally, the algorithm utilizes the matrices $E^{(\ell)}$ to complete entries in E^* which correspond to heavy pairs of colors that were not covered by the brute-force computation. To do so, for each such pair $(h_i, h_j) \in \mathcal{H} \times \mathcal{H}$, the algorithm scans all $e_{i,j}^{(\ell)}$ to find the unique ℓ from

Algorithm 1 CONSTRUCT$E^*(S, \mathcal{H}, W, \varepsilon)$.

```
E* ← ∞
ℓ0 ← ⌊log_{1+ε}(1/ε)⌋ + 1; ℓmax ← ⌈log_{1+ε}(max(S))⌉
for p ∈ S and p̂ ∈ S[p, p + ((1+2ε)/ε)W] do
    if C_p = h_i and C_p̂ = h_i and e*_{i,j} > p̂ − p then
        e*_{i,j} ← p̂ − p; e*_{j,i} ← p̂ − p
    end if
end for
Construct matrix A
for ℓ ← ℓ0 to ℓmax do
    Construct matrix B^(ℓ)
    E^(ℓ) ← A · B^(ℓ)
end for
for ℓ ← ℓ0 to ℓmax do
    for (h_i, h_j) ∈ H × H do
        if e^(ℓ)_{i,j} = e^(ℓ)_{j,i} = 0 and (e^(ℓ+1)_{i,j} = 1 or e^(ℓ+1)_{j,i} = 1) then
            e*_{i,j} ← (1 + ε)^(ℓ+2) W
        end if
    end for
end for
```

Lemma 17 such that $e^{(\ell)}_{i,j} = e^{(\ell)}_{j,i} = 0$ and either $e^{(\ell+1)}_{i,j} = 1$ or $e^{(\ell+1)}_{j,i} = 1$, and sets $e^*_{i,j}$ to be $(1+\varepsilon)^{\ell+2}W$. Computing the entries in E^* for all such pairs costs $O(|\mathcal{H}|^2 \frac{\log \max(s)}{\varepsilon})) = O((\frac{n}{\tau})^2 \frac{\log \max(s)}{\varepsilon}))$, which is dominated by the cost of the matrix multiplications performed during the computation of all of the $E^{(\ell)}$ matrices.

▶ **Lemma 18.** *There exists an algorithm that computes E^* in $\tilde{O}\left(\frac{Wn}{\varepsilon} + \frac{n^2 \cdot \max(S)}{\tau^2 W \cdot \min(\frac{n}{\tau}, \frac{\max(S)}{W})^{3-\omega}}\right)$ time, such that for $h_i, h_j \in \mathcal{H}$ we have $\delta_{h_i,h_j} \leq e^*_{i,j} \leq (1+\varepsilon)\delta_{h_i,h_j}$.*

Proof. The runtime of the algorithm follows from the discussion above.

If $\delta_{h_i,h_j} \leq \frac{1+2\varepsilon}{\varepsilon}W$, then there exist $p \in P(h_1)$ and $\hat{p} \in P(h_i)$ such that $\delta_{h_i,h_j} = |\hat{p} - p| \leq \frac{1+2\varepsilon}{\varepsilon}W$, and so, after the brute-force computation, we have $e^*_{i,j} = \delta_{h_i,h_j}$.

Otherwise, $\delta_{h_i,h_j} > \frac{1+2\varepsilon}{\varepsilon}W$, and so the algorithm sets $e^*_{i,j}$ to be $(1+\varepsilon)^{\ell+2}W$, where ℓ is the unique integer from Lemma 17, and in particular, $e^{(\ell)}_{i,j} = e^{(\ell)}_{j,i} = 0$ and either $e^{(\ell+1)}_{i,j} = 1$ or $e^{(\ell+1)}_{j,i} = 1$. By Lemmas 14 and 12, $\delta_{h_i,h_j} < (1+\varepsilon)^{\ell+2}W < (1+\varepsilon)^2\delta_{h_i,h_j}$.

Finally, to obtain a $(1+\varepsilon)$ approximation one can run the algorithm with approximation parameter $\varepsilon' = \frac{\varepsilon}{3}$, which does not affect the asymptotic time complexity. ◀

4 Proof of Theorem 5

In this section, we analyze the combination of the construction algorithm for E^* given in Section 3.2 together with the generic ACDO algorithm described in Section 2.1. However, we focus on the type of instances of ACDO which are used in Section 5 for solving the approximate snippets problem. In particular, our metric is defined by locations in an array of size n, and so we have $\max(S) = n$. Thus, in our case,

$$T_{E^*}(\mathcal{H}) = O\left(\frac{Wn}{\varepsilon} + \frac{n^\omega \max(\tau, W)^{3-\omega}}{\tau^2 W} \cdot \frac{\log n}{\varepsilon}\right).$$

If $n^{\frac{\omega-1}{\omega+1}} \log^{\frac{1}{2}} n \leq \tau \leq n$, then we choose $W = (\frac{n}{\tau})^{\frac{\omega-1}{2}} \log^{\frac{1}{2}} n$. In such a case, we have $W = (\frac{n}{\tau})^{\frac{\omega-1}{2}} \log^{\frac{1}{2}} n \leq \left(\frac{n}{n^{\frac{\omega-1}{\omega+1}}}\right)^{\frac{\omega-1}{2}} \log^{\frac{1}{2}} n = n^{\frac{\omega-1}{\omega+1}} \log^{\frac{1}{2}} n \leq \tau$, and so

$$T_{E^*}(\mathcal{H}) = O\left(\frac{(\frac{n}{\tau})^{\frac{\omega-1}{2}} n}{\varepsilon} \log^{\frac{1}{2}} n + \frac{n^\omega \tau^{3-\omega}}{\tau^2 (\frac{n}{\tau})^{\frac{\omega-1}{2}} \cdot \log^{\frac{1}{2}} n} \cdot \frac{\log n}{\varepsilon}\right) = O\left(\frac{n^{\frac{\omega+1}{2}}}{\varepsilon \tau^{\frac{\omega-1}{2}}} \log^{\frac{1}{2}} n\right).$$

If $1 \leq \tau \leq n^{\frac{\omega-1}{\omega+1}} \log^{\frac{1}{2}} n$, then we choose $W = \frac{n}{\tau^{\frac{2}{\omega-1}}} \cdot \log^{\frac{1}{\omega-1}} n$. In such a case we have $W = \frac{n}{\tau^{\frac{2}{\omega-1}}} \log^{\frac{1}{\omega-1}} n \geq \frac{n}{n^{\frac{\omega-1}{\omega+1} \cdot \frac{2}{\omega-1}}} \log^{\frac{1}{\omega-1}} n = n^{\frac{\omega-1}{\omega+1}} \log^{\frac{1}{\omega-1}} n \geq \tau$, and so

$$\begin{aligned} T_{E^*}(\mathcal{H}) &= O\left(\frac{n^2}{\varepsilon \tau^{\frac{2}{\omega-1}}} \log^{\frac{1}{\omega-1}} n + \frac{n^\omega (\frac{n}{\tau^{\frac{2}{\omega-1}}})^{2-\omega} \cdot \log(n)^{\frac{2-\omega}{\omega-1}}}{\tau^2} \cdot \frac{\log n}{\varepsilon}\right) \\ &= O\left(\frac{n^2}{\varepsilon \tau^{\frac{2}{\omega-1}}} \log^{\frac{1}{\omega-1}} n\right). \end{aligned}$$

Thus, to summarize we have shown

$$T_{E^*}(\mathcal{H}) = \begin{cases} \tilde{O}\left(\frac{n^2}{\varepsilon \tau^{\frac{2}{\omega-1}}}\right) & \text{for } 1 \leq \tau \leq n^{\frac{\omega-1}{\omega+1}} \\ \tilde{O}\left(\frac{n^{\frac{\omega+1}{2}}}{\varepsilon \tau^{\frac{\omega-1}{2}}}\right) & \text{for } n^{\frac{\omega-1}{\omega+1}} \leq \tau \leq n. \end{cases}$$

Notice that in either case, $T_{E*}(\mathcal{H}) = \Omega(\max(n^2/\tau^2), n)$.

For the NNS data structure we use the van Emde Boas [25] data structure which for a set of m points from integer universe $\{1, 2, \ldots, u\}$ has a preprocessing cost of $O(m \cdot \log\log u)$ and query time $O(\log\log u)$. In our setting, $u = n$, and so $T_{p,NNS}(P(c)) = O(|P(c)| \log\log n)$ and $T_{q,NNS}(n) = O(\log\log n)$.

Thus, the construction time of the ACDO algorithm is

$$\begin{aligned} O(T_{E^*}(\mathcal{H}) + \sum_{c \in \mathcal{C}} T_{p,\mathrm{NNS}}(P(c))) &= O(T_{E^*}(\mathcal{H}) + \sum_{c \in \mathcal{C}} |P(c)| \log\log n) \\ &= O(T_{E^*}(\mathcal{H}) + n \log\log n) \\ &= \tilde{O}(T_{E^*}), \end{aligned}$$

and the query time is $O(\tau \cdot T_{q,\mathrm{NNS}}(n)) = O(\tau \log\log n) = \tilde{O}(\tau)$. To complete the proof of Theorem 5, we plug $\tau = \tilde{O}(n^b)$ into $n^a = T_{E^*}(\mathcal{H})$, to obtain the following tradeoff curve for a fixed ε.

$$\begin{cases} a + \frac{2}{\omega-1} b = 2 & \text{if } 0 \leq b \leq \frac{\omega-1}{\omega+1} \\ \frac{2}{\omega-1} a + b = \frac{\omega+1}{\omega-1} & \text{if } \frac{\omega-1}{\omega+1} \leq b \leq 1. \end{cases}$$

▶ Remark 19. One feature of Theorem 5 algorithm, which is used in the proof of Theorem 6, is that the query cost for ACDO queries when both C and C' are heavy is $O(1)$ time since all the algorithm does is looking up the answer in E^*.

5 Proof Sketch of Theorem 6

The proof of Theorem 6 is inspired by the exact MCDOCH algorithm of [17], combined with a new observation which allows to leverage Theorem 5. Due to space considerations, in this section, we only highlight the differences between our algorithm for AMCDOCH and the exact MCDOCH algorithm of [17]. The complete proof of Theorem 6 is given in the full version [12].

The relevant part of the algorithm of [17] creates an array A of size n which is a permutation of the points in S, and preprocesses A into an RNNS data structure. Then, the algorithm partitions A into blocks of size τ, for an integer parameter $1 \leq \tau \leq n$. Let $\mathcal{I} = \{I_1, I_2 \dots I_{\lceil \frac{n}{\tau} \rceil}\}$ denote the set of blocks.

Following [17], the algorithm constructs a matrix $B = \{b_{i,j}\}$ of size $\lceil \frac{n}{\tau} \rceil \times \lceil \frac{n}{\tau} \rceil$, such that $b_{i,j} = d(I_i, I_j)$. To construct B, the algorithm of [17] performs τ RNNS queries for the computation of each entry in B. Computing B turns out to be the dominating component in the preprocessing runtime of [17].

To obtain a faster preprocessing time for the approximate setting, instead of constructing B, our algorithm constructs an *approximate* matrix $\hat{B} = \{\hat{b}_{i,j}\}$ where $d(I_i, I_j) \leq \hat{b}_{i,j} \leq (1+\varepsilon)d(I_i, I_j)$. The construction of $\hat{B}$ utilizes the algorithm of Theorem 5 as follows. Define a new coloring set $\hat{\mathcal{C}} = \{\hat{c_1}, \hat{c_2}, \dots \hat{c_{|\mathcal{I}|}}\}$ over the points in S, such that for each $\hat{c_i} \in \hat{\mathcal{C}}$ we have $P(\hat{c_i}) = I_i$. Notice that for every $i \neq j$, we have $I_i \bigcap I_j = \emptyset$, and so using the colors of $\hat{\mathcal{C}}$ on S, every point in S has only one color. Thus, the algorithm uses the algorithm of Theorem 5 on S, but with the colors of $\hat{\mathcal{C}}$, and the query time designed to be $\tilde{O}(n^b) = \tilde{O}(\tau)$. Now, $\hat{b}_{i,j}$ is set to be the answer of the ACDO query on colors $\hat{c}_i$ and $\hat{c}_j$, and so $d(I_i, I_j) = \delta_{\hat{c}_i, \hat{c}_j} \leq \hat{b}_{i,j} \leq (1+\varepsilon)\delta_{\hat{c}_i, \hat{c}_j} = (1+\varepsilon)d(I_i, I_j)$.

In the last step of the preprocessing phase, the algorithm preprocesses $\hat{B}$ (instead of B in [13]) using a $2D$ Range Minimum Query (2DRMQ) data structure [3] so that given a rectangle in $\hat{B}$, defined by its corners, the algorithm returns in $O(1)$ time the smallest value entry in the rectangle.

Answering queries and correctness

The process for answering a query is the same as in [17], but using $\hat{B}$ instead of B. Thus, the correctness of the algorithm needs to be proven given the use of $\hat{B}$.

As shown in [17], for each $c \in \mathcal{C}$, $P(c)$ is exactly the points in some interval $A[x_c, y_c]$. The challenging part is answering an AMCDOCH query between C and C', when $P(c)$ and $P(c')$ are disjoint. Thus, assume without loss of generality that $x_c \leq y_c < x_{c'} \leq y_{c'}$. For the description here, we assume that x_c (and $x_{c'}$) is the first location of some block, and y_c (and $y_{c'}$) is the last location of some block. The more general cases are covered in the full version of this paper [12], with an additional execution of $\tilde{O}(\tau)$ RNNS queries.

The query algorithm executes a 2DRMQ data structure query on the rectangle in $\hat{B}$ defined by corners $(\frac{x_c - 1}{\tau} + 1, \frac{x_{c'} - 1}{\tau} + 1)$ and $(\frac{y_c}{\tau}, \frac{y_{c'}}{\tau})$. Let $\hat{b}_{i,j}$ be the answer returned by the 2DRMQ query. Notice that there exist $p \in A[x_c, y_c]$ and $p' \in A[x_{c'}, y_{c'}]$ such that $\delta_{c,c'} = d(p, p')$. Thus, there exist integers $\frac{x_c - 1}{\tau} + 1 \leq i^* \leq \frac{y_c}{\tau}$, $\frac{x_{c'} - 1}{\tau} + 1 \leq j^* \leq \frac{y_{c'}}{\tau}$ such that $p \in I_{i^*}$ and $p' \in I_{j^*}$. Thus, $\delta_{c,c'} = d(p, p') = d(I_{i^*}, I_{j^*})$, and so

$$\begin{aligned}\hat{b}_{i,j} &= \min_{\frac{x_c-1}{\tau}+1 \leq \hat{i} \leq \frac{y_c}{\tau}, \frac{x_{c'}-1}{\tau}+1 \leq \hat{j} \leq \frac{y_{c'}}{\tau}} \{\hat{b}_{\hat{i},\hat{j}}\} \leq \min_{\frac{x_c-1}{\tau}+1 \leq \hat{i} \leq \frac{y_c}{\tau}, \frac{x_{c'}-1}{\tau}+1 \leq \hat{j} \leq \frac{y_{c'}}{\tau}} \{(1+\varepsilon)d(I_{\hat{i}}, I_{\hat{j}})\} \\ &\leq (1+\varepsilon)d(I_{i^*}, I_{j^*}) = (1+\varepsilon)\delta_{c,c'}.\end{aligned}$$

Time Complexity

For the RNNS data structure we use the solution of [23] which on n points (which is the size of A) has preprocessing time $O(n \log n)$ and query time $O(\log^{\varepsilon} n) = \tilde{O}(1)$.

By Remark 19 and the fact that each block has size $\tau = \tilde{O}(n^b)$, each ACDO query used for constructing $\hat{B}$ costs $O(1)$ time. So, after preprocessing the ACDO data structure, constructing $\hat{B}$ costs $O((\frac{n}{\tau})^2)$ time. The preprocessing time of the 2DRMQ data structure of [3] when applied to $\hat{B}$ is $\tilde{O}((\frac{n}{\tau})^2)$. Thus, since $T_{E^*}(n) = \Omega(\max(n^2/\tau^2), n))$, the total time cost of the preprocessing phase, which is composed of constructing the ACDO and RNNS data structures on A, computing $\hat{B}$, and preprocessing a 2DRMQ data structure, is

$$\tilde{O}\left(T_{E^*}(n) + (n/\tau)^2 + n\right) = \tilde{O}\left(T_{E^*}(n)\right) = \begin{cases} \tilde{O}\left(\dfrac{n^2}{\varepsilon \tau^{\frac{2}{\omega-1}}}\right) & \text{for } 1 \leq \tau \leq n^{\frac{\omega-1}{\omega+1}} \\ \tilde{O}\left(\dfrac{n^{\frac{\omega+1}{2}}}{\varepsilon \tau^{\frac{\omega-1}{2}}}\right) & \text{for } n^{\frac{\omega-1}{\omega+1}} \leq \tau \leq n \end{cases}$$

The query process consists of $O(\tau)$ RNNS queries, and a single 2DRMQ query. Thus, the query time cost is $O(\tau)$. Finally, similar to the proof of Theorem 5, we obtained the following tradeoff curve for a fixed ε.

$$\begin{cases} a + \dfrac{2}{\omega-1} b = 2 & \text{if } 0 \leq b \leq \dfrac{\omega-1}{\omega+1} \\ \dfrac{2}{\omega-1} a + b = \dfrac{\omega+1}{\omega-1} & \text{if } \dfrac{\omega-1}{\omega+1} \leq b \leq 1. \end{cases}$$

6 Proof of Theorem 8: CLB for Exact MCDO on an Array

To prove Theorem 8, we design a reduction from $(min, +)$-matrix product on values in $[\hat{n}]$ to MCDO. To do so, consider an *unbalanced* version of $(min, +)$-matrix product where A is of size $\hat{n} \times \hat{m}$, B is of size $\hat{m} \times \hat{n}$, and all of the values are bounded by some positive integer M. It is straightforward to show that for any $x \geq 0$ such that $\hat{m} = \hat{n}^x$, solving unbalanced $(min, +)$-matrix product with any polynomial time improvement over $(\hat{n}^2\hat{m})^{1-o(1)}$ time is equivalent to solving $(min, +)$-matrix product on balanced matrices with $\hat{m} = \hat{n}$ in $\hat{n}^{3-\Omega(1)}$ time. Thus, an algorithm for unbalanced $(min, +)$-matrix product with values bounded by $M = [\min(\hat{n}, \hat{m})] = \hat{n}$ in time less than $(\hat{n}^2\hat{m})^{1-o(1)}$ would refute the Strong-APSP hypothesis.

The reduction

Our goal is to design a reduction from the unbalanced $(min, +)$-matrix product problem with values bounded by $\hat{n} \geq \min(\hat{n}, \hat{m})$ to MCDO. In preparation for the proof of Theorem 9, we first describe the reduction using positive integer parameter M as the upper bound on the values in the input matrices, and during the analysis we set $M = \hat{n}$.

For each $a_{i,j}$, the algorithm defines a point $a'_{i,j} = (M - a_{i,j}) + 9Mj$ with color i. For each $b_{i,j}$, the algorithm defines a point $b'_{i,j} = b_{i,j} + 3M + 9Mi$ with color $\hat{n} + j$. Thus, the largest point defined is at most $O(M \cdot \hat{m})$, and there are $2\hat{n}$ colors. Notice that there cannot be two entries in the same row (column) of A (B) that define the same point. However, it is possible that $a_{i,j} = a_{i',j}$ for $i \neq i'$, and so $a'_{i,j} = a'_{i',j}$, but the two points have different colors. A similar phenomenon can happen to points defined by entries of B which share the same row. Thus, we merge all occurrences of the same point into a single occurrence but colored

with all of the colors of the pre-merge occurrences. The set of points, denoted by S, contains at most $2\hat{n}\hat{m}$ multi-colored points in a metric defined by an array of size $O(M\hat{m})$, and so the algorithm uses the MCDO algorithm to preprocess S. Finally, for each entry $d_{i,j}$ in D, the algorithm sets $d_{i,j} \leftarrow \delta_{i,\hat{n}+j} - 2M$ by performing a query on the MCDO data structure.

Correctness

Suppose $d_{i,j} = a_{i,k^*} + b_{k^*,j}$ for some $k^* \in [\hat{m}]$. For every $k \in [\hat{m}]$, by definition we have $b'_{k,j} > a'_{i,k}$ and so $\left|a'_{i,k} - b'_{k,j}\right| = b'_{k,j} - a'_{i,k} = b_{k,j} + a_{i,k} + 2M$. Since a'_{i,k^*} is colored with color i and $b'_{k^*,j}$ is colored with color $\hat{n} + j$, we have

$$\delta_{i,\hat{n}+j} - 2M \leq \left|a'_{i,k^*} - b'_{k^*,j}\right| - 2M = a_{i,k^*} + b_{k^*,j} = d_{i,j}. \tag{1}$$

Let $k', k'' \in [\hat{m}]$. If $k' \neq k''$, then

$$\begin{aligned}\left|a'_{i,k'} - b'_{k'',j}\right| &= \max\left(a'_{i,k'} - b'_{k'',j}, b'_{k'',j} - a'_{i,k'}\right) \\ &= \max\left(9M(k' - k'') - 2M - a_{i,k'} - b_{k'',j}, b_{k'',j} + a_{i,k'} + 9M(k'' - k') + 2M\right) \\ &\geq 5M.\end{aligned}$$

However, if $k' = k''$ then $a_{i,k'} + b_{k',j} + 2M \leq 4M$. Since $4M < 5M$, we have

$$\delta_{i,\hat{n}+j} = \min_{p \in P(i), p' \in P(\hat{n}+j)} \left\{\left|p - p'\right|\right\} = \min_{1 \leq \hat{k} \leq \hat{n}} \left\{\left|a'_{i,\hat{k}} - b'_{\hat{k},j}\right|\right\}.$$

Thus, there exists some $\hat{k} \in [\hat{m}]$ such that $\delta_{i,\hat{n}+j} = \left|a'_{i,\hat{k}} - b'_{\hat{k},j}\right| = a_{i,\hat{k}} + b_{\hat{k},j} + 2M \geq d_{i,j} + 2M$, and combined with Equation 1 we have $d_{i,j} = \delta_{i,\hat{n}+j} - 2M$.

Lower bound

Recall that in our setting $M = \hat{n}$. The reduction preprocesses an MCDO data structure on $n \leq 2\hat{n}\hat{m}$ points in an array metric of size $O(M\hat{m}) = O(n)$, and then answers $\hat{n}^2$ MCDO queries. If $t_p(n)$ and $t_q(n)$ are preprocessing and query times, respectively, of the MCDO data structure, then the Strong-APSP conjecture implies that $t_p(\hat{n}\hat{m}) + \hat{n}^2 \cdot t_q(\hat{n}\hat{m}) = (\hat{n}^2\hat{m})^{1-o(1)}$.

Recall that our goal is to prove that $a+b \geq 2-o(1)$. Thus, assume towards a contradiction that $a + b = 2 - \Omega(1)$, which, by straightforward algebraic manipulation, implies that $\frac{2a}{a-b} = 1 + \frac{2}{a-b} - \Omega(1)$. Moreover, since the unbalanced $(min, +)$-matrix product is hard for any choice of $x \geq 0$, we can choose $x = \frac{2}{a-b} - 1$ and so $\hat{n}\hat{m} = \hat{n}^{1+x} = \hat{n}^{\frac{2}{a-b}}$. Notice that $2 + \frac{2b}{a-b} = \frac{2a}{a-b}$ Thus, we have $t_p(\hat{n}\hat{m}) + \hat{n}^2 \cdot t_q(\hat{n}\hat{m}) = O(\hat{n}^{\frac{2a}{a-b}} + \hat{n}^{2+\frac{2b}{a-b}}) = O(\hat{n}^{\frac{2a}{a-b}}) = O(\hat{n}^{1+\frac{2}{a-b}-\Omega(1)}) < (\hat{n}^2\hat{m})^{1-o(1)}$, which contradicts the Strong-APSP hypothesis.

7 Conclusions and Open Problems

We have shown the existence of FMM based algorithms for both ACDO and the $(1+\varepsilon)$-approximate snippets problem which, assuming $\omega = 2$, are essentially optimal. Moreover, we proved CLBs for exact version of CDO, implying that the exact versions of CDO and the snippets problem are strictly harder than their approximate versions.

We remark that one immediate and straightforward way to improve our algorithms if $\omega > 2$ is to apply fast rectangular matrix multiplication ([2]). However, we chose not to describe such improvements since they are mostly technical and do not add any additional

insight to the problems that we address. Moreover, we remark that it is straightforward to adapt our CDO algorithms to return the two points (one of each color) that define the distance in addition to the actual distance (see the full version [12]).

Our work leaves open the task of designing improved algorithms for more general metrics such as higher dimensional Euclidean space.

References

1 Mohammad Reza Abbasifard, Bijan Ghahremani, and Hassan Naderi. A survey on nearest neighbor search methods. *International Journal of Computer Applications*, 2014.

2 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.

3 Amihood Amir, Johannes Fischer, and Moshe Lewenstein. Two-dimensional range minimum queries. In *Combinatorial Pattern Matching: 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007. Proceedings 18*, pages 286–294. Springer, 2007. doi:10.1007/978-3-540-73437-6_29.

4 Alexandr Andoni. *Nearest neighbor search: The old, the new, and the impossible*. PhD thesis, Massachusetts Institute of Technology, 2009.

5 Maxim Babenko, Pawel Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 572–591. SIAM, 2014.

6 Djamal Belazzougui and Simon J Puglisi. Range predecessor and lempel-ziv parsing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2053–2071. SIAM, 2016. doi:10.1137/1.9781611974331.CH143.

7 Timothy M Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Fredman's trick meets dominance product: Fine-grained complexity of unweighted apsp, 3sum counting, and more. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, 2023.

8 Shiri Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In *Algorithms–ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings 20*, pages 325–336. Springer, 2012. doi:10.1007/978-3-642-33090-2_29.

9 Maxime Crochemore, Costas S Iliopoulos, Marcin Kubica, M Sohel Rahman, German Tischler, and Tomasz Waleń. Improved algorithms for the range next value problem and applications. *Theoretical Computer Science*, 434:23–34, 2012. doi:10.1016/J.TCS.2012.02.015.

10 Jacob Evald, Viktor Fredslund-Hansen, and Christian Wulff-Nilsen. Near-optimal distance oracles for vertex-labeled planar graphs. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2021.

11 Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. Distance oracles for vertex-labeled graphs. In *Automata, Languages and Programming: 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II 38*, pages 490–501. Springer, 2011. doi:10.1007/978-3-642-22012-8_39.

12 Noam Horowicz and Tsvi Kopelowitz. Color distance oracles and snippets: Separation between exact and approximate solutions. *arXiv preprint*, 2025. arXiv:2507.04578.

13 Matti Karppa and Petteri Kaski. Probabilistic tensors and opportunistic boolean matrix multiplication. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 496–515. SIAM, 2019. doi:10.1137/1.9781611975482.31.

14 Orgad Keller, Tsvi Kopelowitz, Shir Landau Feibish, and Moshe Lewenstein. Generalized substring compression. *Theoretical Computer Science*, 525:42–54, 2014. doi:10.1016/J.TCS.2013.10.010.

15 Orgad Keller, Tsvi Kopelowitz, and Moshe Lewenstein. Range non-overlapping indexing and successive list indexing. In *Algorithms and Data Structures: 10th International Workshop, WADS 2007, Halifax, Canada, August 15-17, 2007. Proceedings 10*, pages 625–636. Springer, 2007. doi:10.1007/978-3-540-73951-7_54.

16 Jon M Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997.

17 Tsvi Kopelowitz and Robert Krauthgamer. Color-distance oracles and snippets. In *27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.24.

18 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1272–1287. SIAM, 2016. doi:10.1137/1.9781611974331.CH89.

19 Tsvi Kopelowitz and Virginia Vassilevska Williams. Towards optimal set-disjointness and set-intersection data structures. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 74:1–74:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.74.

20 Robert Krauthgamer and James R Lee. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807. Citeseer, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982913.

21 Mingfei Li, Chu Chung Christopher Ma, and Li Ning. $(1+\varepsilon)$-distance oracles for vertex-labeled planar graphs. In *International Conference on Theory and Applications of Models of Computation*, pages 42–51. Springer, 2013. doi:10.1007/978-3-642-38236-9_5.

22 J Ian Munro, Yakov Nekrich, and Jeffrey S Vitter. Fast construction of wavelet trees. *Theoretical Computer Science*, 638:91–97, 2016. doi:10.1016/J.TCS.2015.11.011.

23 Yakov Nekrich and Gonzalo Navarro. Sorted range reporting. In *Algorithm Theory – SWAT 2012: 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings 13*, pages 271–282. Springer, 2012. doi:10.1007/978-3-642-31155-0_24.

24 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 605–614. IEEE, 1999. doi:10.1109/SFFCS.1999.814635.

25 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 75–84. IEEE, 1975. doi:10.1109/SFCS.1975.26.

Classical Algorithms for Constant Approximation of the Ground State Energy of Local Hamiltonians

François Le Gall
Graduate School of Mathematics, Nagoya University, Japan

Abstract

We construct classical algorithms computing an approximation of the ground state energy of an arbitrary k-local Hamiltonian acting on n qubits.

We first consider the setting where a good "guiding state" is available, which is the main setting where quantum algorithms are expected to achieve an exponential speedup over classical methods. We show that a *constant* approximation (i.e., an approximation with *constant relative accuracy*) of the ground state energy can be computed classically in $\mathrm{poly}\,(1/\chi, n)$ time and $\mathrm{poly}(n)$ space, where χ denotes the overlap between the guiding state and the ground state (as in prior works in dequantization, we assume sample-and-query access to the guiding state). This gives a significant improvement over the recent classical algorithm by Gharibian and Le Gall (SICOMP 2023), and matches (up to a polynomial overhead) both the time and space complexities of quantum algorithms for constant approximation of the ground state energy. We also obtain classical algorithms for higher-precision approximation.

For the setting where no guided state is given (i.e., the standard version of the local Hamiltonian problem), we obtain a classical algorithm computing a constant approximation of the ground state energy in $2^{O(n)}$ time and $\mathrm{poly}(n)$ space. To our knowledge, before this work it was unknown how to classically achieve these bounds simultaneously, even for constant approximation. We also discuss complexity-theoretic aspects of our results.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Quantum computation theory

Keywords and phrases approximation algorithms, quantum computing, dequantization

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.73

Related Version *Full Version*: `https://arxiv.org/abs/2410.21833` [45]

Funding *François Le Gall*: JSPS KAKENHI grant No. 24H00071, MEXT Q-LEAP grant No. JPMXS0120319794, JST ASPIRE grant No. JPMJAP2302 and JST CREST grant No. JPMJCR24I4.

1 Introduction

1.1 Statement of our main results

Estimating the ground state energy of Hamiltonians is a central problem in both many-body physics and quantum complexity theory. Consider a k-local Hamiltonian

$$H = \sum_{i=1}^{m} H_i \tag{1}$$

acting on n qubits, with $k = O(1)$. Here each term H_i acts non-trivially on only k qubits (but does not need to obey any geometric locality). Let $\mathcal{E}(H)$ denote the ground state energy of H, i.e., its smallest eigenvalue. For any $\varepsilon > 0$, we say that an estimate $\hat{E}$ is an ε-approximation of $\mathcal{E}(H)$ if

$$\left|\hat{E} - \mathcal{E}(H)\right| \leq \varepsilon \sum_{i=1}^{m} \|H_i\|.$$

© François Le Gall;
licensed under Creative Commons License CC-BY 4.0

It is well known that computing a $1/\mathrm{poly}(n)$-approximation of $\mathcal{E}(H)$ is QMA-hard, even for $k = 2$ and for geometrically local Hamiltonians [24, 37, 41, 56, 58]. The Quantum PCP conjecture [4, 5] posits that there exists a constant $\varepsilon > 0$ such that computing an ε-approximation of $\mathcal{E}(H)$ is QMA-hard as well.

Despite these hardness results, efficient quantum algorithms for ground state energy estimation can be constructed when a good "guiding state" is available, i.e., when a quantum state $|\psi\rangle$ that has a good overlap $|\langle\psi|\psi_0\rangle|$ with a ground state $|\psi_0\rangle$ of H is given as an additional input or can be constructed easily (this problem has been called the "guided local Hamiltonian problem" in the recent literature [21, 28, 68]). More precisely, quantum phase estimation [40, 55] and more advanced techniques [67, 26, 30, 38, 51, 53, 54, 59] lead to the following result:

▶ **Fact 1.** *Given a quantum state with overlap χ with a ground state of H, there exists a quantum algorithm that computes with high probability an ε-approximation of $\mathcal{E}(H)$ in $\mathrm{poly}\big(\frac{1}{\chi}, \frac{1}{\varepsilon}, n\big)$ time and $O\big(n + \log(\frac{1}{\varepsilon})\big)$ space.*

When $\chi = 1/\mathrm{poly}(n)$ and $\varepsilon = 1/\mathrm{poly}(n)$, both the running time and the space complexity (i.e., the number of bits and qubits needed for the computation) are polynomial in n. Even for larger values of χ, the performance of this quantum algorithm can be significantly better than the performance of classical algorithms (which typically have running time exponential in n – see later for a detailed discussion). Combined with the fact that for several important applications (e.g., quantum chemistry) good candidates for guiding states can be efficiently constructed, ground state energy estimation is one of the most promising and most anticipated applications of quantum computers (we refer to, e.g., [1, 3, 7, 10, 49, 50, 60] for discussions of these applications).

In this work we investigate the classical complexity of this guided local Hamiltonian problem. A first issue is how to present the guiding state (which is a quantum state, i.e., an exponential-dimension vector) to a classical computer. As in prior works in dequantization [8, 22, 23, 25, 28, 29, 35, 46, 64, 65], we consider sample-and-query access:

- **(i)** for any $j \in [2^n]$ we can efficiently compute $\langle j|\psi\rangle$;
- **(ii)** we can efficiently sample from the probability distribution $p\colon [2^n] \to [0,1]$ that outputs j with probability $|\langle j|\psi\rangle|^2$.

The motivation for (ii), which is the central assumption in dequantized algorithms, is as follows: since measuring the quantum state $|\psi\rangle$ in the computational basis gives a sample from the probability p, it is natural (or "fair") to assume that in the classical setting this distribution is efficiently samplable as well.

Recently, Gharibian and Le Gall [28] constructed a classical algorithm computing an ε-approximation of $\mathcal{E}(H)$ in $n^{O(\log(1/\chi)/\varepsilon)}$ time by dequantizing quantum algorithms based on the Quantum Singular Value Transformation. Here is our main result:

▶ **Theorem 1** (Simplified version)**.** *Given sample-and-query access to a quantum state with overlap χ with a ground state of H, there exists a classical algorithm that computes with high probability an ε-approximation of $\mathcal{E}(H)$ in $\mathrm{poly}\big(\frac{1}{\chi^{1/\varepsilon}}, n\big)$ time and $\mathrm{poly}\big(n, \frac{1}{\varepsilon}\big)$ space.*

Our result significantly improves the running time of the algorithm from [28]. For instance, if $\chi = \Omega(1)$, i.e., if we have a good guiding state, our algorithm has time complexity $\mathrm{poly}(2^{1/\varepsilon}, n)$ instead of $n^{O(1/\varepsilon)}$ in [28]. If $\varepsilon = \Omega(1)$, i.e., if we want only constant precision, our algorithm has time complexity $\mathrm{poly}(1/\chi, n)$ instead of $n^{O(\log(1/\chi))}$ in [28]. Additionally, our approach only uses polynomial space. Comparing Theorem 1 with the bounds of Fact 1 shows that for *constant* precision, there exist a classical algorithm matching (up to a polynomial overhead) the performance of quantum algorithms.

Using the same technique, we also obtain the following result for the case where no guided state is given (i.e., the standard version of the local Hamiltonian problem):

▶ **Theorem 2** (Simplified version). *For any constant $\varepsilon > 0$, there exists a classical algorithm that computes with high probability an ε-approximation of $\mathcal{E}(H)$ in $2^{O(n)}$ time and* $\mathrm{poly}(n)$ *space.*

To our knowledge, before this work it was unknown how to achieve simultaneously running time $2^{O(n)}$ and space complexity $\mathrm{poly}(n)$ for ground state energy estimation of arbitrary local Hamiltonians (even for constant ε). We will further discuss the implications of our results in Section 1.3 after reviewing known classical approaches for ground state energy estimation in the next subsection.

1.2 Background on classical approaches for ground state energy estimation

There are two main classical approaches for estimating the ground state energy of a local Hamiltonian:

- The power method or its variant the Lanczos method [42, 43], which estimates the ground state using matrix-vector multiplications. Since the Hamiltonian is a (sparse) matrix of dimension 2^n, the time complexity is $O^*(2^n)$.[1] There are two main issues with this approach. First, it requires storing explicit vectors in memory, which leads to space complexity $\Omega(2^n)$ and significantly reduces its applicability. Second, it is unclear how a guiding state would help significantly reduce the time complexity (having a good guiding state does reduce the number of iterations, but each iteration still requires matrix-vector multiplications of matrices and vectors of dimension 2^n).
- Quantum Monte Carlo methods, which use sampling arguments to estimate the ground state without having to store explicitly the quantum state. This approach is especially useful for "stoquastic" Hamiltonians, i.e., Hamiltonians for which all the off-diagonal elements are real and non-positive, and has lead to the design of classical algorithms as well as as complexity-theoretic investigations of the complexity of the local Hamiltonian problem for stochastic Hamiltonians [17, 19, 15, 52]. While some of these techniques have been extended to a few classes of non-stoquastic local Hamiltonians, such as gapped local Hamiltonians [16] or arbitrary Hamiltonians with succinct ground state [36], for ground state energy estimation the "sign-problem" significantly limits its applications to arbitrary local Hamiltonians [32, 66]. It is also unclear how the guiding state would help reduce the time complexity.

A third approach is direct classical simulation of the quantum circuit used in Fact 1. There are several techniques for simulating quantum circuits on a classical computer. If the circuit acts on n qubits and has m gates, the Schrödinger method stores the entire state vector in memory and performs successive matrix-vector multiplications, using roughly $m2^n$ time and 2^n space. While the space complexity can in several cases be significantly reduced using matrix product states or more general tensor networks [13], those representations also require exponential space in the worst case. On the other hand, the Feynman method calculates an amplitude as a sum of terms, using roughly 4^m time and $m+n$ space (this approach was used by Bernstein and Vazirani [12] to prove the inclusion $\mathsf{BQP} \subseteq \mathsf{P}^{\#\mathsf{P}}$). Aaronson and Chen [2] have introduced a recursive version of the Feynman method, inspired by the proof of Savitch theorem [61], that works in $2^{O(n \log m)}$ time and $\mathrm{poly}(n, m)$ space.

[1] In this paper the notation $O^*(\cdot)$ suppresses the $\mathrm{poly}(n)$ factors.

Other prior works. Several ground state energy estimation classical algorithms have also been developed for special classes of Hamiltonians, such as one-dimensional gapped local Hamiltonians [44], quantum analogues of Max Cut [6, 31, 39, 48, 57], or Hamiltonians defined on structured graphs [9, 11, 14]. Additionally, there are a few works [18, 27, 33] achieving weaker (but still nontrivial) approximation ratios of the ground state energy, and a recent work [20] achieving a constant approximation ratio for any local Hamiltonian in time slightly better than 2^n (but not space-efficiently).

1.3 Implication of our results

We now discuss several implications of our results.

Better understanding of the quantum advantage. As already mentioned, Theorem 1 implies that for any constant precision parameter ε, we can construct classical ground state energy estimation algorithms with performance matching (up to a polynomial overhead) the performance of the best known quantum algorithms. While Ref. [28] already showed this for $\chi = O(1)$, Theorem 1 proves this result for any value $\chi \in (0, 1]$. This implies that (under the assumption that current quantum algorithms are optimal) there is no superpolynomial quantum advantage for the constant-precision guided local Hamiltonian problem and gives another strong evidence that exponential quantum advantage for ground state energy estimation (and applications to, e.g., quantum chemistry) comes from the improved precision achievable in the quantum setting.

Space-efficient classical ground state energy estimation algorithms. The second main contribution of this work is the design of space-efficient algorithms for ground state energy estimation. In particular, for the case where no guiding state is available, Theorem 2 gives a $2^{O(n)}$-time $\mathrm{poly}(n)$-space classical algorithm. As already mentioned, to our knowledge before this work it was unknown how to achieve simultaneously running time $2^{O(n)}$ time and space complexity $\mathrm{poly}(n)$ for arbitrary local Hamiltonians: even for constant ε, the best running time was $2^{O(n \log n)}$ by the approach by Aaronson and Chen [2].

Potentially practical classical ground state energy estimation algorithms. From a practical perspective, the potential of Theorem 1 is even more striking. In particular, for $\chi = \Omega(1)$, i.e., when we have access to a fairly good guided state (which is the case in some applications to quantum chemistry), we obtain a $\mathrm{poly}(2^{1/\varepsilon}, n)$-time $\mathrm{poly}(n)$-space classical algorithm. Note that the running time is polynomial even for $\varepsilon = 1/\log n$. Additionally, the running time is better than 2^n, which is the typically running time of other classical methods, even for precision as low as $\varepsilon = c/n$, for a small enough constant $c > 0$. While evaluating the practicality of our algorithm is beyond the scope of this paper, we hope our algorithms find applications in many-body physics.

Complexity-theoretic implications. In order to formally discuss complexity theoretic aspects of our results and their relations with standard complexity classes as BPP, BQP and QMA, we first need to introduce decision versions of our problems. As standard in Hamiltonian complexity theory, we add the promise that either (i) $\mathcal{E}(H) \leq a$ or (ii) $\mathcal{E}(H) > b$ holds, for some values $a, b \in [0, 1]$ such that $b - a > \varepsilon$, and ask to decide which of (i) or (ii) holds. This leads to the following (standard) decision version of the local Hamiltonian problem:

$\mathsf{LH}(\varepsilon)$ (Local Hamiltonian problem – decision version)

Input: $*$ a $O(1)$-local Hamiltonian H as in Equation (1) acting on n qubits
$*$ two numbers $a, b \in [0,1]$ such that $b - a > \varepsilon$

Promise: either (i) $\mathcal{E}(H) \leq a$ or (ii) $\mathcal{E}(H) > b$ holds

Goal: decide which of (i) or (ii) holds

As already mentioned, the problem $\mathsf{LH}(\varepsilon)$ is QMA-complete for $\varepsilon = 1/\mathrm{poly}(n)$. On the other hand, for any $\varepsilon = O(1)$ Theorem 2 leads to the inclusion

$$\mathsf{LH}(\varepsilon) \in \mathsf{BPTimeSpace}\left(2^{O(n)}, \mathrm{poly}(n)\right),$$

where $\mathsf{BPTimeSpace}(t(n), s(n))$ denotes the class of (promise) decision problems that can be solved with probability at least $2/3$ by a probabilistic Turing machine running in $t(n)$ time and using $s(n)$ space.

For the guided local Hamiltonian problem, another subtle issue is how to access the guiding state $|\psi\rangle$. So far we have assumed sample-and-query access to $|\psi\rangle$ when considering classical algorithms. While satisfactory when discussing algorithmic aspects of the problem (as we did so far), such "oracle" access to the input is problematic if we want to discuss relations with standard complexity classes such as BPP, BQP and QMA. Instead, we make the following assumptions: in the quantum setting, the description of a quantum polynomial-size circuit creating $|\psi\rangle$ is given as input; in the classical setting, the description of a classical polynomial-size circuit implementing sample-and-query access to $|\psi\rangle$ is given as input. This leads to the following decision version of the guided local Hamiltonian problem:

$\mathsf{GLH}(\varepsilon, \chi)$ (Guided Local Hamiltonian problem – decision version)

Input: $*$ a $O(1)$-local Hamiltonian H as in Equation (1) acting on n qubits
$*$ the description of a $\mathrm{poly}(n)$-size circuit implementing access to a quantum state $|\psi\rangle$ with overlap at least χ with the ground state of H
$*$ two numbers $a, b \in [0,1]$ such that $b - a > \varepsilon$

Promise: either (i) $\mathcal{E}(H) \leq a$ or (ii) $\mathcal{E}(H) > b$ holds

Goal: decide which of (i) or (ii) holds

Prior results on the hardness of the guided local Hamiltonian problem [21, 28] combined with Fact 1 imply that $\mathsf{GLH}(\varepsilon, \chi)$ is BQP-complete for $\varepsilon = 1/\mathrm{poly}(n)$ and constant χ. Ref. [28] also showed that this problem is in the class BPP for constant ε and constant χ. Theorem 1 enables us to strengthen this result and show that for constant ε, the inclusion $\mathsf{GLH}(\varepsilon, \chi) \in \mathsf{BPP}$ holds for $\chi = 1/\mathrm{poly}(n)$ as well.

These complexity-theoretic implications are summarized in Table 1.

1.4 Overview of the techniques

The full version of the paper [45] includes a 3-page overview of the techniques, which is omitted here due to space constraints.

2 Preliminaries

In this section we introduce notations and definitions, and present some useful lemmas.

Table 1 The complexity of the problems $\mathsf{LH}(\varepsilon)$ and $\mathsf{GLH}(\varepsilon, \chi)$.

	$\mathsf{LH}(\varepsilon)$	$\mathsf{GLH}(\varepsilon, \Theta(1))$	$\mathsf{GLH}\left(\varepsilon, \frac{1}{\mathrm{poly}(n)}\right)$
$\varepsilon = \Theta(1)$	in $\mathsf{BPTimeSpace}(2^{O(n)}, \mathrm{poly}(n))$ (Th. 2)	in BPP (Ref. [28])	in BPP (Th. 1)
$\varepsilon = \frac{1}{\mathrm{poly}(n)}$	QMA-complete (Refs. [37, 41])	BQP-complete (Refs. [21, 28])	

2.1 Notations

General notations. For any integer N we write $[N] = \{1, \ldots, N\}$. Define $\mathbb{R}[\mathrm{poly}(n)]$ as the set of all real numbers with binary expansion of polynomial length. More precisely, for any function $f\colon \mathbb{N} \to \mathbb{N}$, we define the set

$$\mathbb{R}[f(n)] = \left\{ \pm \left(a_0 + \sum_{i=1}^{f(n)} a_i 2^i + \sum_{i=1}^{f(n)} b_i 2^{-i} \right) \;\middle|\; a_0, \ldots, a_{f(n)}, b_1, \ldots, b_{f(n)} \in \{0,1\} \right\} \subseteq \mathbb{R}$$

of all real numbers with binary expansion of polynomial length $2f(n) + 2$ (including one bit for encoding the sign). Then $\mathbb{R}[\mathrm{poly}(n)]$ is the union of the $\mathbb{R}[f(n)]$ for all polynomial functions $f(n)$. We define $\mathbb{C}[f(n)]$ and $\mathbb{C}[\mathrm{poly}(n)]$ similarly, by requiring that both the real part and the imaginary part are in $\mathbb{R}[f(n)]$ and $\mathbb{R}[\mathrm{poly}(n)]$, respectively.

Vectors and matrices. In this paper we consider vectors in $\mathbb{C}^N$ and matrices in $\mathbb{C}^{N \times N}$, for some integer N, and write $n = \lceil \log_2(N) \rceil$. Note that this notation is consistent with the notation of Section 1, where we considered the special case $N = 2^n$. We usually write quantum states (i.e., unit-norm vectors) using Greek letters and Dirac notation, e.g., we use $|\psi\rangle$ or $|\varphi\rangle$. We write arbitrary vectors (i.e., vectors of arbitrary norm) using Roman letters, e.g., we use v or w.

For a matrix $A \in \mathbb{C}^{N \times N}$ and any $\ell \in [N]$, we denote the ℓ-th row of A by $A[\ell, \cdot]$. The matrix A is normal if it can be written $A = UDU^{-1}$ where D is a diagonal matrix with real entries and U is a unitary matrix. We use $\|A\|$ to denote the spectral norm of A, which is defined for a normal matrix as the maximum magnitude of the eigenvalues of A (and defined as square root of the maximum eigenvalue of A^*A, where A^* denotes the conjugate transpose of A, in general). This norm is submultiplicative, i.e., the inequality $\|AB\| \leq \|A\|\|B\|$ holds for any matrices $A, B \in \mathbb{C}^{N \times N}$. We also have $|u^* A v| \leq \|A\|\|u\|\|v\|$ for any $u, v \in \mathbb{C}^N$, where $\|u\|$ and $\|v\|$ denote the Euclidean norms of u and v, respectively.

Eigenvalues and overlap. Consider a normal matrix $A \in \mathbb{C}^{N \times N}$. Let $A = \sum_{i=1}^{2^n} \lambda_i |u_i\rangle\langle u_i|$ be its eigenvalue decomposition, with eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{2^n}$ and corresponding orthonormal eigenvectors $|u_1\rangle, \ldots, |u_{2^n}\rangle$. We denote by $\mathcal{E}(A) = \lambda_1$ the smallest eigenvalue of A. For any $\sigma \geq 0$, let us write $S(A, \sigma) = \{i \in [N] \mid \lambda_i(A) \leq \mathcal{E}(A) + \sigma\}$. For any vector $w \in \mathbb{C}^N$, let

$$\Gamma_\sigma(A, w) = \sqrt{\sum_{i \in S(A,\, \sigma)} |\langle u_i | w \rangle|^2}$$

denote the overlap of w with the eigenspace corresponding to eigenvalues in $[\mathcal{E}(A)\,, \mathcal{E}(A) + \sigma]$. Note that the standard definition of the overlap (used in Section 1) corresponds to the case $\sigma = 0$.

2.2 Access to vectors and matrices

We now define the notions of access to vectors and matrices needed for this work. These notions are similar to prior works on dequantization [23, 46, 64], but we need to precisely discuss the encoding length and the space complexity.

We start with query access to a vector.

▶ **Definition 3.** *We have query access to a vector $w \in \mathbb{C}^N$ with encoding length $\mathsf{len}(w)$ and costs $\mathsf{qt}(w)$ and $\mathsf{qs}(w)$ if*

1. *for each $i \in [N]$, we have $w_i \in \mathbb{C}[\mathsf{len}(w)]$;*
2. *for any $i \in [N]$, the coordinate w_i can be obtained in $\mathsf{qt}(w)$ time and $\mathsf{qs}(w)$ space.*

If $\mathsf{len}(w), \mathsf{qt}(w), \mathsf{qs}(w) \leq \mathrm{poly}(n)$, we simply say that we have query access to w.

Next, we introduce the stronger notion of sample-and-query access to a vector.

▶ **Definition 4.** *We have sample-and-query access to a vector $w \in \mathbb{C}^N$ if*

1. *we have query access to w;*
2. *we can compute in $\mathrm{poly}(n)$ time[2] a sample from the distribution $p\colon [N] \to [0,1]$ such that $p(i) = \frac{|w_i|^2}{\|w\|^2}$ for each $i \in [N]$.*

When $\|w\| = 1$, Item 2 in Definition 4 states that we can efficiently sample from the same distribution as the distribution obtained when measuring the quantum state $\sum_{i=1}^N w_i|i\rangle$ in the computational basis.

We extend the notion of query access to matrices as follows:

▶ **Definition 5.** *We have query access to a matrix $B \in \mathbb{C}^{N\times N}$ if*

1. *for each $(i,j) \in [N]\times[N]$, we have $B[i,j] \in \mathbb{C}[\mathrm{poly}(n)]$;*
2. *for any $(i,j) \in [N]\times[N]$, the entry $B[i,j]$ can be obtained in $\mathrm{poly}(n)$ time;*
3. *for any $i \in [N]$, the number s_i of nonzero entries in $B[i,\cdot]$ can be obtained in $\mathrm{poly}(n)$ time;*
4. *for any $i \in [N]$ and any $\ell \in [s_i]$, the ℓ-th nonzero entry of $B[i,\cdot]$ can be obtained in $\mathrm{poly}(n)$ time.*

Items 3 and 4 in Definition 5 are needed to deal with sparse matrices.

2.3 Local Hamiltonians and matrix decompositions

We give below technical details about the description of local Hamiltonians and the matrix decompositions introduced in this work.

Description of local Hamiltonians. A k-local Hamiltonian acting on n qubits is a Hermitian matrix $H \in \mathbb{C}^{N\times N}$ with $N = 2^n$ that can be written as $H = \sum_{i=1}^m H_i$ with $m = \mathrm{poly}(n)$, where each term H_i is an Hermitian matrix acting non-trivially on at most k qubits. Each H_i can be described by a $2^k \times 2^k$ matrix representing its action on the k qubits on which it acts non-trivially. We assume that each entry of this description is in $\mathbb{C}[\mathrm{poly}(n)]$. This description is given as input. For convenience we also assume that we know $\|H_i\|$ for each $i \in [N]$.[3]

[2] Since a polynomial upper bound on the time complexity implies a polynomial upper bound on the space complexity, hereafter we omit to explicitly mention that the space complexity is $\mathrm{poly}(n)$ as well.

[3] Note that each $\|H_i\|$ can be computed from its description as a $2^k \times 2^k$ matrix. The computation is efficient when k is small, e.g., for $k = O(\log n)$, which is the most interesting regime for the local Hamiltonian problem.

Matrix decomposition. Here is the complete definition of the matrix decomposition we consider.

▶ **Definition 6.** *For a matrix A, an integer $s \geq 0$ and a real number $\kappa \in \mathbb{R}[\mathrm{poly}(n)]$, an (s, κ)-decomposition of A is a decomposition*

$$A = \sum_{i=1}^{m} A_i \quad \textit{with} \quad \sum_{i=1}^{m} \|A_i\| \leq \kappa$$

in which A_i is an s-sparse matrix for each $i \in [m]$. We always (implicitly) assume the following:

- *for each $i \in [m]$, we have query access to the matrix A_i;*
- *we know bounds $\kappa_1, \ldots, \kappa_m \in \mathbb{R}[\mathrm{poly}(n)]$ such that $\|A_i\| \leq \kappa_i$ for $i \in [m]$ and $\sum_{i=1}^{m} \kappa_i = \kappa$.*

If $\kappa = 1$, we simply call the decomposition an s-decomposition.

2.4 Lemmas

We present four lemmas that are needed to prove our results.

The first lemma is the "powering lemma" from [34] to amplify the success probability of probabilistic estimators (the formulation below for complex numbers is from [46, Lemma 3]):

▶ **Lemma 7** (Powering lemma)**.** *Consider a randomized algorithm that produces an estimate $\tilde{\mu}$ of a complex-valued quantity μ such that $|\tilde{\mu} - \mu| \leq \varepsilon$ holds with probability at least $3/4$. Then, for any $\delta > 0$, it suffices to repeat $O(\log(1/\delta))$ times the algorithm and take both the median of the real parts and the median of the imaginary parts to obtain an estimate $\hat{\mu}$ such that $|\hat{\mu} - \mu| \leq \sqrt{2}\varepsilon$ holds with probability at least $1 - \delta$.*

To perform eigenvalue estimation we will need a low-degree polynomial that approximates well the "rectangle" function. We will use the following result from [30].[4]

▶ **Lemma 8** (Lemma 29 in [30])**.** *For any $\xi \in (0, 1]$, any $\tau \in [0, 1)$ and any $\theta \in (0, 1 - \tau]$, there exists an efficiently computable polynomial $P \in \mathbb{R}[x]$ of degree $O\big(\frac{1}{\theta}\log(1/\xi)\big)$ such that $|P(x)| \in [0, 1]$ for all $x \in [-1, 1]$ and*

$$\begin{cases} P(x) \in [1 - \xi, 1] & \text{if } x \in [0, \tau], \\ P(x) \in [0, \xi] & \text{if } x \in [\tau + \theta, 1]. \end{cases} \tag{2}$$

We will use the following result from [63] that gives an upper bound on the coefficients of polynomials bounded in the interval $[-1, 1]$ (such as the polynomial from Lemma 8).

▶ **Lemma 9** (Lemma 4.1 in [63])**.** *Let $P(x) = \sum_{i=0}^{d} a_i x^i$ be a univariate polynomial of degree d such that $|P(x)| \leq 1$ for all $x \in [-1, 1]$. Then $\sum_{i=0}^{d} |a_i| \leq 4^d$.*

Finally, we discuss how to classically estimate the inner product $\langle \psi | w \rangle$ given sample-and-query access to a quantum state $|\psi\rangle$ and query access to a vector w. More precisely, we are considering the following problem:

[4] Lemma 8 follows by taking $t = \tau + \theta/2$ and $\delta' = \theta/2$ in Lemma 29 of [30]. The computability of the polynomial is discussed explicitly in [47, Appendix A.3].

$\mathsf{IP}(\varepsilon, \delta)$ (Estimation of Inner Product)

Input: $*$ sample-and-query access to a quantum state $|\psi\rangle \in \mathbb{C}^N$
$*$ query access to a vector $w \in \mathbb{C}^N$ with encoding length $\mathsf{len}(w)$ and costs $\mathsf{qt}(w)$ and $\mathsf{qs}(w)$

Output: an estimate $a \in \mathbb{C}$ such that

$$|a - \langle\psi|w\rangle| \leq \varepsilon \|w\|$$

holds with probability at least $1 - \delta$

Prior works on dequantization [23, 46, 64] have shown how to solve this problem efficiently. It can be easily checked that these approaches are space-efficient as well, leading to the following statement. For completeness we give a proof in the full version of the paper [45].

▶ **Lemma 10.** *For any $\varepsilon \in (0, 1]$ and any $\delta \in (0, 1]$, the problem $\mathsf{IP}(\varepsilon, \delta)$ can be solved classically in time $O^*\big(\mathsf{qt}(w)\,\varepsilon^{-2}\log(1/\delta)\big)$ and space $\mathsf{qs}(w) + O^*\Big(\Big(\mathsf{len}(w) + \log(1/\varepsilon)\Big)\log(1/\delta)\Big)$.*

3 Iterated Matrix Multiplication

In this section we show how to classically estimate the inner product $\langle\psi|B_r \cdots B_1|\varphi\rangle$ for sparse matrices $B_1, \ldots, B_r$ and two quantum states $|\psi\rangle$ and $|\varphi\rangle$ to which we have classical access. More precisely, we consider the following problem.

$\mathsf{IMM}(s, r, \varepsilon, \delta)$ (Estimation of Iterated Matrix Multiplication)

Input: $*$ query access to s-sparse matrices $B_1, \ldots, B_r \in \mathbb{C}^{N\times N}$
$*$ query access to a quantum state $|\varphi\rangle \in \mathbb{C}^N$
$*$ sample-and-query access to a quantum state $|\psi\rangle \in \mathbb{C}^N$

Output: an estimate $\hat{E} \in \mathbb{C}$ such that

$$\left|\hat{E} - \langle\psi|B_r \cdots B_1|\varphi\rangle\right| \leq \varepsilon \|B_1\| \cdots \|B_r\|$$

holds with probability at least $1 - \delta$

Here is the main result of this section.

▶ **Proposition 11.** *For any $s \geq 1$, any $r \geq 1$, any $\varepsilon \in (0, 1]$ and any $\delta \in (0, 1]$, the problem $\mathsf{IMM}(s, r, \varepsilon, \delta)$ can be solved classically in time*

$$O^*\big(s^r \varepsilon^{-2}\log(1/\delta)\big)$$

and space

$$O^*\big(r^2 + \big(r + \log(1/\varepsilon)\big)\log(1/\delta)\big).$$

The proof of Proposition 11 is based on the following lemma, which can be seen as a space-efficient version of the approach for iterated matrix multiplication used in [28, 62].

▶ **Lemma 12.** *There is a classical algorithm that implements query access to $B_r \cdots B_1|\varphi\rangle$ with encoding length $\mathsf{len}(B_r \cdots B_1|\varphi\rangle) = O^*(r)$ and costs $\mathsf{qt}(B_r \cdots B_1|\varphi\rangle) = O^*(s^r)$ and $\mathsf{qs}(B_r \cdots B_1|\varphi\rangle) = O^*\big(r^2\big)$.*

Proof. Here is the main idea: to obtain the ℓ-th entry of $B_r \cdots B_1|\varphi\rangle$, we only need to know the s nonzero entries of the ℓ-th row of B_r, which can be queried directly, together with the corresponding entries in the vector $B_{r-1} \cdots B_1|\varphi\rangle$, which can be computed recursively. The algorithm is described in pseudocode below.

```
Algorithm 𝒜(ℓ, r)      // computes ⟨ℓ|B_r ··· B_1|φ⟩
1  if r = 0 then
2  |  return ⟨ℓ|φ⟩ ;
3  else
4  |  z ← 0 ;
5  |  get the number of nonzero entries of the row B_r[ℓ, ·] and write it s' ;
6  |  for t from 1 to s' do
7  |  |  get the index of the t-th nonzero entry of B_r[ℓ, ·] and write it j ;
8  |  |  x ← B_r[ℓ, j] ;          // queries ⟨ℓ|B_r|j⟩
9  |  |  y ← 𝒜(j, r − 1) ;     // computes recursively ⟨j|B_{r−1} ··· B_1|φ⟩
10 |  |  z ← z + x · y ;
11 |  return z ;
```

We first analyze the correctness of the algorithm. Let $j_1, \ldots, j_{s'}$ represent the indices of the nonzero entries of the row $B_r[\ell, \cdot]$. Since

$$\langle \ell | B_r \cdots B_1 | \varphi \rangle = \sum_{t=1}^{s'} \langle \ell | B_r | j_t \rangle \langle j_t | B_{r-1} \cdots B_1 | \varphi \rangle,$$

Algorithm $\mathcal{A}(\ell, r)$ outputs $\langle \ell | B_r \cdots B_1 | \varphi \rangle$.

Let $\mathcal{T}(r)$ denote the running time of this procedure. We have $\mathcal{T}(r) \leq s\mathcal{T}(r-1) + O^*(s)$, and thus $\mathcal{T}(r) = O^*(s^r)$. For each $j \in [N]$, the entry $\langle j|\psi\rangle$ has a poly(n)-bit binary expansion. Each entry of the matrices $B_1, \ldots, B_r$ also has a poly(n)-bit binary expansion. This implies that $\mathsf{len}(B_r \cdots B_1|\varphi\rangle) = O^*(r)$. We finally consider the space complexity. The recursion tree has depth r. At each level of the recursion, the values x, y and z at Steps 8, 9 and 10 can be stored in $O^*(r)$ bits, and we need one $O(\log s)$-bit counter for storing the current value of t. The overall space complexity of the algorithm is thus $O^*(r(r + \log s)) = O^*\big(r^2\big)$. ◀

Proposition 11 is obtained by applying Lemma 10 to the vector $w = B_r \cdots B_1|\varphi\rangle$, for which we can implement query access from Lemma 12:

Proof of Proposition 11. From Lemma 12 we have query access to $w = B_r \cdots B_1|\varphi\rangle$ with encoding length $\mathsf{len}(w) = O^*(r)$ and costs $\mathsf{qt}(w) = O^*(s^r)$ and $\mathsf{qs}(w) = O^*\big(r^2\big)$. Using Lemma 10, we can then compute an estimate $a \in \mathbb{C}$ such that

$$|a - \langle \psi | B_r \cdots B_1 | \varphi \rangle| \leq \varepsilon \, \|B_r \cdots B_1 |\varphi\rangle\| \leq \varepsilon \, \|B_1\| \cdots \|B_r\|$$

holds with probability at least $1 - \delta$. The time and space complexity are

$$O^*\big(s^r \varepsilon^{-2} \log(1/\delta)\big)$$

and

$$O^*\big(r^2 + \big(r + \log(1/\varepsilon)\big) \log(1/\delta)\big),$$

respectively. ◀

4 Polynomial Transformations of Decomposable Matrices

In this section we show how to classically estimate the inner product $\langle\psi|P(A)|\varphi\rangle$ for a matrix A with an s-decomposition, a polynomial P, and two quantum states $|\psi\rangle$ and $|\varphi\rangle$ to which we have classical access. More precisely, we consider the following problem.

$\mathsf{PT}(s,d,\eta)$ (Estimation of Polynomial Transformation)

Input: $*$ a matrix $A \in \mathbb{C}^{N\times N}$ with an s-decomposition
$*$ a polynomial $P \in \mathbb{R}[x]$ of degree d with $|P(x)| \leq 1 \quad \forall x \in [-1,1]$
$*$ query access to quantum state $|\varphi\rangle \in \mathbb{C}^N$
$*$ sample-and-query access to a quantum state $|\psi\rangle \in \mathbb{C}^N$

Output: an estimate $\hat{E} \in \mathbb{C}$ such that

$$\left|\hat{E} - \langle\psi|P(A)|\varphi\rangle\right| \leq \eta \tag{3}$$

holds with probability at least $1 - 1/\exp(n)$

Here is the main result of this section.

▶ **Proposition 13.** *For any $s \geq 2$, any $d \geq 1$ and any $\eta \in (0,1] \cap \mathbb{R}[\mathrm{poly}(n)]$, the problem $\mathsf{PT}(s,d,\eta)$ can be solved classically in time $O^*\left(s^{c\cdot d}\eta^{-4}\right)$ time, for some universal constant $c > 0$, and space $O^*\left(d^2\right)$.*

The proof of Proposition 13 is based on the following lemma, whose proof is given after the proof of the proposition.

▶ **Lemma 14.** *For any $r \in \{0,\ldots,d\}$, any $\eta \in (0,1] \cap \mathbb{R}[\mathrm{poly}(n)]$ and any $\delta \in (0,1]$, there is a classical algorithm that computes an estimate $\hat{E}_r \in \mathbb{C}$ such that $\left|\hat{E}_r - \langle\psi|A^r|\varphi\rangle\right| \leq \frac{\eta}{4^d}$ holds with probability at least $1-\delta$ in time $O^*\left(s^r 2^{8d}\eta^{-4} d\log(1/\delta)\right)$ and space $O^*\left(d^2 + d\log(1/\delta)\right)$.*

Proof of Proposition 13. Let us write the polynomial P as $P(x) = \sum_{r=0}^{d} a_r x^d$. For any $\delta' \in (0,1]$, we describe how to compute an estimate $\hat{E}$ such that Equation (3) holds with probability at least $1-\delta'$. Taking $\delta' = 1/\exp(n)$ then proves the proposition.

For each $r \in \{0,\ldots,d\}$ such that $a_r \neq 0$, we apply Lemma 14 with $\delta = \frac{\delta'}{d+1}$ to obtain an approximation $\hat{E}_r$ of $\langle\psi|A^r|\varphi\rangle$ such that

$$\Pr\left[\left|\hat{E}_r - \langle\psi|A^r|\varphi\rangle\right| \leq \frac{\eta}{4^d}\right] \geq 1 - \frac{\delta'}{d+1}.$$

For each $r \in \{0,\ldots,d\}$ such that $a_r = 0$, we set $\hat{E}_r = 0$. We then output $\hat{E} = \sum_{r=0}^{d} a_r \hat{E}_r$. From the union bound and the triangle inequality, with probability at least $1-\delta'$ we have

$$\left|\hat{E} - \langle\psi|P(A)|\varphi\rangle\right| \leq \sum_{r=0}^{d} |a_r|\,|E_r - \langle\psi|A^r|\varphi\rangle| \leq \eta,$$

where we used Lemma 9 to derive the last inequality.

The time complexity is

$$O^*\left(\left(\sum_{r=0}^{d} s^r\right) 2^{8d}\eta^{-4} d\log\left(\frac{d}{\delta'}\right)\right) = O^*\left(s^d 2^{8d}\eta^{-4} d\log\left(\frac{d}{\delta'}\right)\right) = O^*\left(s^{c\cdot d}\eta^{-4}\right),$$

for some universal constant $c > 0$. The space complexity is

$$O^*\left(d^2 + d\log(d/\delta')\right) = O^*\left(d^2\right),$$

as claimed. ◀

Proof of Lemma 14. As in Definition 6, we write the s-decomposition of A as $A = \sum_{i=1}^{m} A_i$, where each A_i is an s-sparse matrix such that $\|A_i\| \leq \kappa_i$, with $\kappa_1 + \cdots + \kappa_m = 1$. Consider the probability distribution $p\colon [m]^r \to [0,1]$ defined as $p(x) = \kappa_{x_1} \cdots \kappa_{x_r}$ for any $x \in [m]^r$ (the condition $\kappa_1 + \cdots + \kappa_m = 1$ guarantees that this is a probability distribution). Define a random variable X as follows: sample a vector x from the distribution p, and set

$$X = \frac{\langle\psi|A_{x_1} \cdots A_{x_r}|\varphi\rangle}{p(x)}.$$

Repeat the above procedure $t = \left\lceil 64 \cdot 4^{2^d}/\eta^2 \right\rceil$ times and output the mean. Let Y denote the corresponding complex random variable. We have

$$\begin{aligned}
\mathbb{E}[Y] = \mathbb{E}[X] &= \sum_{x\in[m]^r} \langle\psi|A_{x_1} \cdots A_{x_r}|\varphi\rangle = \langle\psi|A^r|\varphi\rangle \\
\mathbb{V}[Y] &\leq \frac{1}{t}\mathbb{E}\left[|X|^2\right] \\
&= \frac{1}{t} \sum_{x\in[m]^r} \frac{|\langle\psi|A_{x_1} \cdots A_{x_r}|\varphi\rangle|^2}{\kappa_{x_1} \cdots \kappa_{x_r}} \\
&\leq \frac{1}{t} \sum_{x\in[m]^r} \frac{\|A_{x_1} \cdots A_{x_r}\|^2}{\kappa_{x_1} \cdots \kappa_{x_r}} \\
&\leq \frac{1}{t} \sum_{x\in[m]^r} \kappa_{x_1} \cdots \kappa_{x_r} \\
&= \frac{1}{t} \left(\kappa_1 + \cdots + \kappa_m\right)^r \\
&= \frac{1}{t}.
\end{aligned}$$

From Chebyshev's inequality, we thus obtain:

$$\Pr\left[|Y - \langle\psi|A^r|\varphi\rangle| \geq \frac{\eta}{2\sqrt{2}\cdot 4^d}\right] \leq \frac{8\cdot 4^{2d}}{\eta^2 t} \leq \frac{1}{8}. \tag{4}$$

We cannot directly use this strategy since we do not know $\langle\psi|A_{x_1} \cdots A_{x_r}|\varphi\rangle$. Instead, we estimate this quantity using Proposition 11. This leads to the following algorithm.

Algorithm $\mathcal{B}(\eta)$ `// estimates` $\langle\psi|A^r|\varphi\rangle$ `with precision` $\frac{\eta}{\sqrt{2}\cdot 4^d}$

$t \leftarrow \left\lceil 64 \cdot 4^{2d}/\eta^2 \right\rceil$;
$z \leftarrow 0$;
for i **from** 1 **to** t **do**
  Take a vector x according to the distribution p.
  Use Proposition 11 for the problem $\mathsf{IMM}\left(s, r, \frac{\eta}{2\sqrt{2}\cdot 4^d}, \frac{1}{8t}\right)$ to compute an estimate $\alpha \in \mathbb{C}$ of $\langle\psi|A_{x_1} \cdots A_{x_r}|\varphi\rangle$;
  $z \leftarrow z + \frac{\alpha}{t\cdot p(x)}$;
return z;

The complexity of Algorithm $\mathcal{B}(\eta)$ is dominated by the computation at Step 5, which is done t times. From Proposition 11, we obtain the upper bounds

$$O^*\big(t \cdot s^r 2^{4d} \eta^{-2} \log(8t)\big) = O^*\big(s^r 2^{8d} \eta^{-4} d\big)$$

and

$$O^*\left(r^2 + \left(r + \log\left(\frac{2\sqrt{2}\cdot 4^d}{\eta}\right)\right)\log(8t)\right) = O^*\big(r^2 + d^2\big) = O^*\big(d^2\big)$$

on the time and space complexities, respectively.

We now analyze the correctness of Algorithm $\mathcal{B}(\eta)$. Let Z be the random variable corresponding to the output of Step 7 when at Step 4 the vectors x's are the same vectors as in the random variable Y. For any choice of x at Step 4, the estimate α of Step 5 satisfies

$$|\alpha - \langle\psi|A_{x_1}\cdots A_{x_r}|\varphi\rangle| \leq \frac{\eta\,\kappa_{x_1}\cdots\kappa_{x_r}}{2\sqrt{2}\cdot 4^d} \tag{5}$$

with probability at least $1-1/(8t)$. Under the condition that Inequality (5) is always satisfied during the t repetitions, we have

$$|Z-Y| \leq \sum_x \frac{1}{t\cdot p(x)} \frac{\eta\,\kappa_{x_1}\cdots\kappa_{x_r}}{2\sqrt{2}\cdot 4^d} = \frac{\eta}{2\sqrt{2}\cdot 4^d},$$

where the sum is over the t vectors x chosen at Step 4. By the union bound, we thus have

$$\Pr\left[|Z-Y| > \frac{\eta}{2\sqrt{2}\cdot 4^d}\right] \leq \frac{1}{8}. \tag{6}$$

Combining Equation (4) and Equation (6) gives

$$\begin{aligned}\Pr\left[|Z - \langle\psi|A^r|\varphi\rangle| \leq \frac{\eta}{\sqrt{2}\cdot 4^d}\right] &\geq \Pr\left[|Z-Y| \leq \frac{\eta}{2\sqrt{2}\cdot 4^d} \text{ and } |Y - \langle\psi|A^r|\varphi\rangle| \leq \frac{\eta}{2\sqrt{2}\cdot 4^d}\right] \\ &\geq 1 - \Pr\left[|Z-Y| > \frac{\eta}{2\sqrt{2}\cdot 4^d}\right] - \Pr\left[|Y - \langle\psi|A^r|\varphi\rangle| > \frac{\eta}{2\sqrt{2}\cdot 4^d}\right] \\ &\geq \frac{3}{4}.\end{aligned}$$

We can then use Lemma 7 to obtain an estimate $\hat{E}_r \in \mathbb{C}$ such that $\left|\hat{E}_r - \langle\psi|A^r|\varphi\rangle\right| \leq \frac{\eta}{4^d}$ holds with probability at least $1-\delta$. Lemma 7 introduces a $\log(1/\delta)$ factor in the time complexity and an additive $O^*(d\log(1/\delta))$ term in the space complexity since for the computation of the medians we need to store $O(\log(1/\delta))$ values, each requiring $O^*(d)$ bits.[5] ◀

5 Eigenvalue Estimation

In this section we use the results proved in Section 4 to estimate the smallest eigenvalue of a normal matrix. We describe the most general problem we are solving in Section 5.1 and then prove Theorems 1 and 2 in Section 5.2.

[5] Here we are using the assumption $\kappa_i \in \mathbb{C}[\text{poly}(n)]$ for all $i \in [m]$, see Definition 6. This implies that $p(x)$ can be encoded in $O^*(r)$ bits and the output of Procedure $\mathcal{B}(\eta)$ in $O^*(d)$ bits.

5.1 General result

We consider the problem of estimating the smallest eigenvalue of a normal matrix with an (s, κ)-decomposition, given classical access to a guiding state. Here is the formal description of the problem:

$\mathsf{SE}(s, \chi, \varepsilon)$ `(Estimation of the Smallest Eigenvalue)`

Input: $*$ a normal matrix $A \in \mathbb{C}^{N \times N}$ with an (s, κ)-decomposition (for any κ)

$*$ sample-and-query access to a quantum state $|\psi\rangle \in \mathbb{C}^N$ with $\Gamma_{\frac{\varepsilon}{2}\kappa}(A, |\psi\rangle) \geq \chi$

Output: an estimate $E^* \in \mathbb{R}$ such that

$$|E^* - \mathcal{E}(A)| \leq \varepsilon\,\kappa \tag{7}$$

holds with probability at least $1 - 1/\exp(n)$

We prove the following theorem:

▶ **Theorem 15.** *For any $s \geq 2$ and any $\varepsilon, \chi \in (0, 1] \cap \mathbb{R}[\mathrm{poly}(n)]$, the problem $\mathsf{SE}(s, \chi, \varepsilon)$ can be solved classically in time $O^*\left(s^{\frac{c' \log(1/\chi)}{\varepsilon}}\right)$ time, for some universal constant $c' > 0$, and space $O^*\left(1/\varepsilon^2\right)$.*

Proof of Theorem 15. Let us define $A' = \frac{1}{2}(I + \frac{A}{\kappa})$ and write $T = \lceil 4/\varepsilon \rceil$. The (s, κ)-decomposition of A gives an $(s+1)$-decomposition of A'. Observe that A' has eigenvalues in the interval $[0, 1]$. The main idea is to divide this interval into T subintervals of length at most $\varepsilon/4$ and find in which subinterval $\mathcal{E}(A')$ lies in. Since $\mathcal{E}(A) = 2\kappa\mathcal{E}(A') - \kappa$, this will give an estimate of $\mathcal{E}(A)$.

Concretely, for any $t \in \{0, \ldots, T-1\}$, we consider the following test that checks if $\mathcal{E}(A')$ is "approximately" smaller than $t\frac{\varepsilon}{4}$. The approximation comes from the use of an estimator at Step 2 – details of the implementation of this step are discussed later.

$\mathsf{Test}(t)$ `// checks if` $\mathcal{E}(A')$ `is (approximately) smaller than` $t\frac{\varepsilon}{4}$

1 Let P be the polynomial of Lemma 8 with $\tau = t\frac{\varepsilon}{4}$, $\theta = \frac{\varepsilon}{4}$ and $\xi = \frac{\chi^2}{12}$;
2 Compute an estimate $\hat{E} \in \mathbb{C}$ such that $\left|\hat{E} - \langle\psi|P(A')|\psi\rangle\right| \leq \frac{\chi^2}{4}$;
3 **if** $\left|\hat{E}\right| \geq \frac{\chi^2}{2}$ **then** output "yes";
4 **else** output "no";

Let t^* be the smallest value of $t \in \{0, \ldots, T-1\}$ such that $\mathsf{Test}(t)$ outputs "yes". Define $E^* = t^*\frac{\varepsilon}{2}\kappa - \kappa$. The following claim, whose proof is given in the full version of the paper [45], guarantees that E^* is a correct estimate of $\mathcal{E}(A)$.

▷ Claim 16. $|E^* - \mathcal{E}(A)| \leq \varepsilon\,\kappa$.

We now discuss the implementation of Step 2. For any $\delta \in (0, 1]$, we describe how to compute an estimate E^* such that Equation (7) holds with probability at least $1 - \delta$. Taking $\delta = 1/\exp(n)$ then proves the theorem.

We use the algorithm of Proposition 13 for the problem $\mathsf{PT}(s, \deg(P), \chi^2/4)$ in order to obtain an estimator $\hat{E}$ such that $\left|\hat{E} - \langle\psi|P(A')|\psi\rangle\right| \leq \chi^2/4$ holds with probability at least $1 - \delta/T$. Since $\mathsf{Test}(t)$ is called at most T times, the union bound guarantees that with probability at least $1 - \delta$ no error occur during these tests. This implies that the output E^* satisfies the bound of Claim 16 with probability at least $1 - \delta$.

The overall time complexity is $O^*\big(T \cdot (s+1)^{c \cdot \deg(P)} \chi^{-8}\big) = O^*\Big(s^{\frac{c' \log(1/\chi)}{\varepsilon}}\Big)$, for some universal constant $c' > 0$. Since $\deg(P) = O^*(1/\varepsilon)$, the space complexity is $O^*\big(1/\varepsilon^2\big)$. ◀

5.2 Consequences: Theorems 1 and 2

We are now ready to give the full statements of Theorems 1 and 2 and prove them.

▶ **Theorem 14** (Full version). *Consider any $\varepsilon, \chi \in (0,1] \cap \mathbb{R}[\mathrm{poly}(n)]$. For any k-local Hamiltonian H on n qubits, given sample-and-query access to a quantum state $|\psi\rangle$ with*

$$\Gamma_{\frac{\varepsilon}{2}\kappa}(H, |\psi\rangle) \geq \chi,$$

where $\kappa = \sum_{i=1}^m \|H_i\|$, there is a classical algorithm that computes in $\mathrm{poly}\big(\frac{1}{\chi^{k/\varepsilon}}, n\big)$ time and $O^*\big(\frac{1}{\varepsilon^2}\big)$ space an estimate $\hat{E}$ such that

$$\left|\hat{E} - \mathcal{E}(H)\right| \leq \varepsilon \sum_{i=1}^m \|H_i\|$$

holds with probability at least $1 - 1/\exp(n)$.

Proof of Theorem 1. We apply Theorem 15 with $A = H$, $s = 2^k$ and $\kappa = \sum_{i=1}^m \|H_i\|$. ◀

▶ **Theorem 15** (Full version). *Consider any $\varepsilon \in (0,1] \cap \mathbb{R}[\mathrm{poly}(n)]$. For any k-local Hamiltonian H on n qubits, there is a classical algorithm that computes in $2^{O(kn/\varepsilon)}$ time and $O^*\big(\frac{1}{\varepsilon^2}\big)$ space an estimate $\hat{E}$ such that*

$$\left|\hat{E} - \mathcal{E}(H)\right| \leq \varepsilon \sum_{i=1}^m \|H_i\|$$

holds with probability at least $1 - 1/\exp(n)$.

Proof of Theorem 2. Let us write $H = \sum_{i=1}^{2^n} \lambda_i |u_i\rangle\langle u_i|$ the spectral decomposition of H, with $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{2^n}$ and corresponding orthonormal eigenvectors $|u_1\rangle, \ldots, |u_{2^n}\rangle$, where $\lambda_1 = \mathcal{E}(H)$. We apply Theorem 1 with the Hamiltonian $H' = H \otimes I$ acting on $2n$ qubits (here I is the identity matrix acting on n qubits) and guiding state $|\Phi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=1}^{2^n} |i\rangle|i\rangle$, for which it is trivial to implement sample-and-query access. This is a maximally entangled state, which can also be written as $|\Phi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=1}^{2^n} |u_i\rangle|v_i\rangle$, for another orthonormal basis $\{|v_1\rangle, \ldots, |v_{2^n}\rangle\}$.

Let $t \in [2^n]$ denote the multiplicity of the ground state energy of H. The eigenspace corresponding to the ground state energy of H' is thus $\mathrm{span}\,\{|u_i\rangle|j\rangle \mid i \in [t], j \in [2^n]\}$. We have

$$\Gamma_0(H, |\Phi\rangle) = \sqrt{\sum_{i=1}^{t} \sum_{j=1}^{2^n} |(\langle u_i|\langle j|)|\Phi\rangle|^2} \geq \frac{1}{\sqrt{2^n}} \sqrt{\sum_{j=1}^{2^n} |\langle j|v_1\rangle|^2} = \frac{1}{\sqrt{2^n}}.$$

The conclusion follows from Theorem 1 with $\chi = 2^{-n/2}$. ◀

References

1 Scott Aaronson. Why quantum chemistry is hard. *Nature Physics*, 5:707–708, 2009. doi:10.1038/nphys1415.

2 Scott Aaronson and Lijie Chen. Complexity-theoretic foundations of quantum supremacy experiments. In *Proceedings of the 32nd Computational Complexity Conference (CCC 2017)*, pages 22:1–22:67, 2017. doi:10.5555/3135595.3135617.

3 Daniel S. Abrams and Seth Lloyd. Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors. *Physical Review Letters*, 83:5162–5165, 1999. doi:10.1103/PhysRevLett.83.5162.

4 Dorit Aharonov, Itai Arad, and Thomas Vidick. Guest column: the quantum PCP conjecture. *SIGACT News*, 44(2):47–79, 2013. doi:10.1145/2491533.2491549.

5 Dorit Aharonov and Tomer Naveh. Quantum NP - a survey, 2002. arXiv:quant-ph/0210077.

6 Anurag Anshu, David Gosset, and Karen Morenz. Beyond product state approximations for a quantum analogue of Max Cut. In *Proceedings of the 15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, pages 7:1–7:15, 2020. doi:10.4230/LIPIcs.TQC.2020.7.

7 Alán Aspuru-Guzik, Anthony D. Dutoi, Peter J. Love, and Martin Head-Gordon. Simulated quantum computation of molecular energies. *Science*, 309(5741):1704–1707, 2005. doi:10.1126/science.1113479.

8 Ainesh Bakshi and Ewin Tang. An improved classical singular value transformation for quantum machine learning. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 2398–2453. SIAM, 2024. doi:10.1137/1.9781611977912.86.

9 Nikhil Bansal, Sergey Bravyi, and Barbara M. Terhal. Classical approximation schemes for the ground-state energy of quantum and classical Ising spin Hamiltonians on planar graphs. *Quantum Information and Computation*, 9(7):701–720, 2009. doi:10.26421/QIC9.7-8-12.

10 Bela Bauer, Sergey Bravyi, Mario Motta, and Garnet Kin-Lic Chan. Quantum algorithms for quantum chemistry and quantum materials science. *Chemical Reviews*, 120(22):12685–12717, 2020. doi:10.1021/acs.chemrev.9b00829.

11 Thiago Bergamaschi. Improved product-state approximation algorithms for quantum local Hamiltonians. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, pages 20:1–20:18, 2023. doi:10.4230/LIPICS.ICALP.2023.20.

12 Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997. doi:10.1137/S0097539796300921.

13 Jacob Biamonte and Ville Bergholm. Tensor networks in a nutshell, 2017. arXiv:1708.00006.

14 Fernando G.S.L. Brandao and Aram W. Harrow. Product-state approximations to quantum ground states. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 871–880, 2013. doi:10.1145/2488608.2488719.

15 Sergey Bravyi. Monte Carlo simulation of stoquastic Hamiltonians. *Quantum Information and Computation*, 15(13–14):1122–1140, 2015. doi:10.26421/QIC15.13-14-3.

16 Sergey Bravyi, Giuseppe Carleo, David Gosset, and Yinchen Liu. A rapidly mixing Markov chain from any gapped quantum many-body system. *Quantum*, 7:1173, 2023. doi:10.22331/q-2023-11-07-1173.

17 Sergey Bravyi, David P. Divincenzo, Roberto Oliveira, and Barbara M. Terhal. The complexity of stoquastic local Hamiltonian problems. *Quantum Information and Computation*, 8(5):361–385, 2008. doi:10.26421/QIC8.5-1.

18 Sergey Bravyi, David Gosset, Robert König, and Kristan Temme. Approximation algorithms for quantum many-body problems. *Journal of Mathematical Physics*, 60(3):032203, March 2019. doi:10.1063/1.5085428.

19 Sergey Bravyi and Barbara Terhal. Complexity of stoquastic frustration-free Hamiltonians. *SIAM Journal on Computing*, 39(4):1462–1485, 2010. doi:10.1137/08072689X.

20 Harry Buhrman, Sevag Gharibian, Zeph Landau, François Le Gall, Norbert Schuch, and Suguru Tamaki. Beating Grover search for low-energy estimation and state preparation, 2024. doi:10.48550/arXiv.2407.03073.

21 Chris Cade, Marten Folkertsma, Sevag Gharibian, Ryu Hayakawa, François Le Gall, Tomoyuki Morimae, and Jordi Weggemans. Improved hardness results for the guided local Hamiltonian problem. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, pages 32:1–32:19, 2023. doi:10.4230/LIPIcs.ICALP.2023.32.

22 Nai-Hui Chia, András Gilyén, Han-Hsuan Lin, Seth Lloyd, Ewin Tang, and Chunhao Wang. Quantum-inspired algorithms for solving low-rank linear equation systems with logarithmic dependence on the dimension. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC 2020)*, pages 47:1–47:17, 2020. doi:10.4230/LIPIcs.ISAAC.2020.47.

23 Nai-Hui Chia, András Pal Gilyén, Tongyang Li, Han-Hsuan Lin, Ewin Tang, and Chunhao Wang. Sampling-based sublinear low-rank matrix arithmetic framework for dequantizing quantum machine learning. *Journal of the ACM*, 69(5):33:1–33:72, 2022. doi:10.1145/3549524.

24 Toby Cubitt and Ashley Montanaro. Complexity classification of local Hamiltonian problems. *SIAM Journal on Computing*, 45(2):268–316, 2016. doi:10.1137/140998287.

25 Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, and Dacheng Tao. Quantum-inspired algorithm for general minimum conical hull problems. *Physical Review Research*, 2:033199, 2020. doi:10.1103/PhysRevResearch.2.033199.

26 Yimin Ge, Jordi Tura, and J. Ignacio Cirac. Faster ground state preparation and high-precision ground energy estimation with fewer qubits. *Journal of Mathematical Physics*, 60(2):022202, 2019. doi:10.1063/1.5027484.

27 Sevag Gharibian and Julia Kempe. Approximation algorithms for QMA-complete problems. *SIAM Journal on Computing*, 41(4):1028–1050, 2012. doi:10.1137/110842272.

28 Sevag Gharibian and François Le Gall. Dequantizing the quantum singular value transformation: Hardness and applications to quantum chemistry and the quantum PCP conjecture. *SIAM Journal on Computing*, 52(4):1009–1038, 2023. doi:10.1137/22M1513721.

29 András Gilyén, Zhao Song, and Ewin Tang. An improved quantum-inspired algorithm for linear regression. *Quantum*, 6:754, 2022. doi:10.22331/q-2022-06-30-754.

30 András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 193–204, 2019. doi:10.1145/3313276.3316366.

31 Sean Hallgren, Eunou Lee, and Ojas Parekh. An approximation algorithm for the MAX-2-local Hamiltonian problem. In *Proceedings of the International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2020)*, pages 59:1–59:18, 2020. doi:10.4230/LIPICS.APPROX/RANDOM.2020.59.

32 Dominik Hangleiter, Ingo Roth, Daniel Nagaj, and Jens Eisert. Easing the Monte Carlo sign problem. *Science Advances*, 6(33):2375–2548, 2020. doi:10.1126/sciadv.abb8341.

33 Aram W. Harrow and Ashley Montanaro. Extremal eigenvalues of local Hamiltonians. *Quantum*, 1:6, 2017. doi:10.22331/q-2017-04-25-6.

34 Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986. doi:10.1016/0304-3975(86)90174-X.

35 Dhawal Jethwani, François Le Gall, and Sanjay K. Singh. Quantum-inspired classical algorithms for singular value transformation. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, pages 53:1–53:14, 2020. doi:10.4230/LIPIcs.MFCS.2020.53.

36 Jiaqing Jiang. Local Hamiltonian problem with succinct ground state is MA-complete, 2024. arXiv:2309.10155.

37 Julia Kempe, Alexei Y. Kitaev, and Oded Regev. The complexity of the local Hamiltonian problem. *SIAM Journal on Computing*, 35(5):1070–1097, 2006. doi:10.1137/S0097539704445226.

38 Alex Kerzner, Vlad Gheorghiu, Michele Mosca, Thomas Guilbaud, Federico Carminati, Fabio Fracas, and Luca Dellantonio. A square-root speedup for finding the smallest eigenvalue. *Quantum Science and Technology*, 6(4):045025, 2023. doi:10.1088/2058-9565/ad6a36.

39 Robbie King. An improved approximation algorithm for quantum Max-Cut on triangle-free graphs. *Quantum*, 7:1180, 2023. doi:10.22331/q-2023-11-09-1180.

40 Alexei Yu. Kitaev. Quantum measurements and the Abelian stabilizer problem, 1995. arXiv:quant-ph/9511026.

41 Alexei Yu. Kitaev, Alexander H. Shen, and Mikhail N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, 2002.

42 Jacek Kuczyński and Henryk Woźniakowski. Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. *SIAM Journal on Matrix Analysis and Applications*, 13(4):1094–1122, 1992. doi:10.1137/0613066.

43 Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45(4):255–282, 1950. doi:10.6028/jres.045.026.

44 Zeph Landau, Umesh Vazirani, and Thomas Vidick. A polynomial time algorithm for the ground state of one-dimensional gapped local Hamiltonians. *Nature Physics*, 11:566–569, 2015. doi:10.1038/nphys3345.

45 François Le Gall. Classical algorithms for approximating the ground energy of local Hamiltonians (full version of this paper), 2025. arXiv:2410.21833.

46 François Le Gall. Robust dequantization of the quantum singular value transformation and quantum machine learning algorithms. *computational complexity*, 34:2, 2025. doi:10.1007/s00037-024-00262-3.

47 François Le Gall, Yupan Liu, and Qisheng Wang. Space-bounded quantum state testing via space-efficient quantum singular value transformation, 2023. doi:10.48550/arXiv.2308.05079.

48 Eunou Lee and Ojas Parekh. An improved quantum Max Cut approximation via maximum matching. In *Proceedings of the 51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, pages 105:1–105:11, 2024. doi:10.4230/LIPIcs.ICALP.2024.105.

49 Joonho Lee, Dominic W. Berry, Craig Gidney, William J. Huggins, Jarrod R. McClean, Nathan Wiebe, and Ryan Babbush. Even more efficient quantum computations of chemistry through tensor hypercontraction. *PRX Quantum*, 2:030305, 2021. doi:10.1103/PRXQuantum.2.030305.

50 Seunghoon Lee, Joonho Lee, Huanchen Zhai, Yu Tong, Alexander M. Dalzell, Ashutosh Kumar, Phillip Helms, Johnnie Gray, Zhi-Hao Cui, Wenyuan Liu, Michael Kastoryano, Ryan Babbush, John Preskill, David R. Reichman, Earl T. Campbell, Edward F. Valeev, Lin Lin, and Garnet Kin-Lic Chan. Evaluating the evidence for exponential quantum advantage in ground-state quantum chemistry. *Nature Communications*, 14:1952, 23. doi:10.1038/s41467-023-37587-6.

51 Lin Lin and Yu Tong. Near-optimal ground state preparation. *Quantum*, 4:372, 2020. doi:10.22331/q-2020-12-14-372.

52 Yupan Liu. StoqMA meets distribution testing. In *Proceedings of the 16th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2021)*, pages 4:1–4:22, 2021. doi:10.4230/LIPIcs.TQC.2021.4.

53 Guang Hao Low and Yuan Su. Quantum eigenvalue processing. In *Proceedings of the 2024 Symposium on Foundations of Computer Science (FOCS 2024)*, pages 1051–1062, 2024. doi:10.1109/FOCS61266.2024.00070.

54 John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. Grand unification of quantum algorithms. *PRX Quantum*, 2:040203, 2021. doi:10.1103/PRXQuantum.2.040203.

55 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

56 Roberto Oliveira and Barbara M. Terhal. The complexity of quantum spin systems on a two-dimensional square lattice. *Quantum Information and Computation*, 8(10):900–924, 2008. doi:10.26421/QIC8.10-2.

57 Ojas Parekh and Kevin Thompson. Application of the level-2 quantum Lasserre hierarchy in quantum approximation algorithms. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, pages 102:1–102:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.102.

58 Stephen Piddock and Ashley Montanaro. The complexity of antiferromagnetic interactions and 2D lattices. *Quantum Information and Computation*, 17(7-8):636–672, 2017. doi:10.26421/QIC17.7-8-6.

59 David Poulin and Pawel Wocjan. Preparing ground states of quantum many-body systems on a quantum computer. *Physical Review Letters*, 102(13):130503, 2009. doi:10.1103/PHYSREVLETT.102.130503.

60 Markus Reiher, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences*, 114(29):7555–7560, 2017. doi:10.1073/pnas.1619152114.

61 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

62 Martin Schwarz and Maarten Van den Nest. Simulating quantum circuits with sparse output distributions, 2013. arXiv:1310.6749.

63 Alexander A. Sherstov. Making polynomials robust to noise. *Theory of Computing*, 9:593–615, 2013. doi:10.4086/TOC.2013.V009A018.

64 Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019)*, pages 217–228, 2019. doi:10.1145/3313276.3316310.

65 Ewin Tang. Quantum principal component analysis only achieves an exponential speedup because of its state preparation assumptions. *Physical Review Letters*, 127:060503, 2021. doi:10.1103/PhysRevLett.127.060503.

66 Matthias Troyer and Uwe-Jens Wiese. Computational complexity and fundamental limitations to Fermionic quantum Monte Carlo simulations. *Physical Review Letters*, 94:170201, 2005. doi:10.1103/PhysRevLett.94.170201.

67 Joran van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. Quantum SDP-Solvers: Better upper and lower bounds. *Quantum*, 4:230, 2020. doi:10.22331/q-2020-02-14-230.

68 Jordi Weggemans, Marten Folkertsma, and Chris Cade. Guidable local Hamiltonian problems with implications to heuristic ansatz state preparation and the quantum PCP conjecture. In *Proceedings of the 19th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2024)*, pages 10:1–10:24, 2024. doi:10.4230/LIPIcs.TQC.2024.10.

Core-Sparse Monge Matrix Multiplication

Improved Algorithm and Applications

Paweł Gawrychowski
University of Wrocław, Poland

Egor Gorbachev
Saarland University and Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Tomasz Kociumaka
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

Min-plus matrix multiplication is a fundamental tool for designing algorithms operating on distances in graphs and different problems solvable by dynamic programming. We know that, assuming the APSP hypothesis, no subcubic-time algorithm exists for the case of general matrices. However, in many applications the matrices admit certain structural properties that can be used to design faster algorithms. For example, when considering a planar graph, one often works with a *Monge matrix* A, meaning that the *density matrix* $A^{\square}$ has non-negative entries, that is, $A^{\square}_{i,j} \coloneqq A_{i+1,j} + A_{i,j+1} - A_{i,j} - A_{i+1,j+1} \geq 0$. The min-plus product of two $n \times n$ Monge matrices can be computed in $\mathcal{O}(n^2)$ time using the famous SMAWK algorithm.

In applications such as longest common subsequence, edit distance, and longest increasing subsequence, the matrices are even more structured, as observed by Tiskin [J. Discrete Algorithms, 2008]: they are (or can be converted to) *simple unit-Monge matrices*, meaning that the density matrix is a permutation matrix and, furthermore, the first column and the last row of the matrix consist of only zeroes. Such matrices admit an implicit representation of size $\mathcal{O}(n)$ and, as shown by Tiskin [SODA 2010 & Algorithmica, 2015], their min-plus product can be computed in $\mathcal{O}(n \log n)$ time. Russo [SPIRE 2010 & Theor. Comput. Sci., 2012] identified a general structural property of matrices that admit such efficient representation and min-plus multiplication algorithms: the *core size* δ, defined as the number of non-zero entries in the density matrices of the input and output matrices. He provided an adaptive implementation of the SMAWK algorithm that runs in $\mathcal{O}((n+\delta)\log^3 n)$ or $\mathcal{O}((n+\delta)\log^2 n)$ time (depending on the representation of the input matrices).

In this work, we further investigate the core size as the parameter that enables efficient min-plus matrix multiplication. On the combinatorial side, we provide a (linear) bound on the core size of the product matrix in terms of the core sizes of the input matrices. On the algorithmic side, we generalize Tiskin's algorithm (but, arguably, with a more elementary analysis) to solve the core-sparse Monge matrix multiplication problem in $\mathcal{O}(n + \delta \log \delta) \subseteq \mathcal{O}(n + \delta \log n)$ time, matching the complexity for simple unit-Monge matrices. As witnessed by the recent work of Gorbachev and Kociumaka [STOC'25] for edit distance with integer weights, our generalization opens up the possibility of speed-ups for weighted sequence alignment problems. Furthermore, our multiplication algorithm is also capable of producing an efficient data structure for recovering the witness for any given entry of the output matrix. This allows us, for example, to preprocess an integer array of size n in $\widetilde{\mathcal{O}}(n)$ time so that the longest increasing subsequence of any sub-array can be reconstructed in $\widetilde{\mathcal{O}}(\ell)$ time, where ℓ is the length of the reported subsequence. In comparison, Karthik C. S. and Rahul [arXiv, 2024] recently achieved $\mathcal{O}(\ell + n^{1/2}\operatorname{polylog} n)$-time reporting after $\mathcal{O}(n^{3/2}\operatorname{polylog} n)$-time preprocessing.

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

Keywords and phrases Min-plus matrix multiplication, Monge matrix, longest increasing subsequence

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.74

Related Version *Full Version*: https://arxiv.org/abs/2408.04613v2 [21]



© Paweł Gawrychowski, Egor Gorbachev, and Tomasz Kociumaka;
licensed under Creative Commons License CC-BY 4.0



Funding *Egor Gorbachev*: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

1 Introduction

The *min-plus product* (also known as the distance product or the tropical product) of two matrices A and B is defined as a matrix $C = A \otimes B$ such that $C_{i,k} = \min_j (A_{i,j} + B_{j,k})$. The task of computing the min-plus product of two $n \times n$ matrices can be solved in $n^3/\exp(\Omega(\sqrt{\log n}))$ time [46], and it is fine-grained equivalent to the All-Pairs Shortest Path (APSP) problem [44], asking to compute the distances between every pair of vertices in a directed weighted graph on n vertices. While it is conjectured that APSP, and hence also the min-plus product, do not admit $n^{3-\Omega(1)}$-time solutions, faster algorithms exist for many special cases arising in numerous applications of the min-plus product; see, e.g., [2, 6, 10, 15–17, 23, 43, 47, 49]. Most of the underlying procedures rely on fast matrix multiplication for the standard $(+, \cdot)$-product.

Monge matrices constitute a notable exception: An $n \times n$ matrix A is a Monge matrix if its *density matrix* $A^{\square}$ is non-negative, that is, $A^{\square}_{i,j} \coloneqq A_{i+1,j} + A_{i,j+1} - A_{i,j} - A_{i+1,j+1} \geq 0$ holds for all $i, j \in [0\,..n-1)$. The min-plus product of two $n \times n$ Monge matrices can be computed in $\mathcal{O}(n^2)$ time using the SMAWK algorithm [1], and the resulting matrix still satisfies the Monge property. Monge matrices arise in many combinatorial optimization problems; see [7, 8] for surveys. One of the most successful applications is for planar graphs, where the distances between vertices on a single face satisfy the Monge property (see [18, Section 2.3]). This observation allowed for an $\widetilde{\mathcal{O}}(n)$-time[1] single-source shortest path algorithm for planar graphs (with negative real weights) [18], and the resulting techniques now belong to the standard toolkit for designing planar graph algorithms; see, e.g., [5, 11, 25].

Another important application of Monge matrices is in sequence alignment problems, such as *edit distance* and *longest common subsequence* (LCS), as well as in the related *longest increasing subsequence* (LIS) problem. Already in the late 1980s, Apostolico, Atallah, Larmore, and McFaddin [3] noted that the so-called *DIST matrices*, which (among others) specify the weighted edit distances between prefixes of one string and suffixes of another string, satisfy the Monge property. A modern interpretation of this observation is that these matrices store boundary-to-boundary distances in planar *alignment graphs* [37]; see also [4, 26, 32] for further early applications of DIST matrices and their Monge property.

In the late 2000s, Tiskin [39, 42] observed that the DIST matrices originating from the unweighted variants of edit distance, LCS, and LIS problems are more structured. For this, he introduced the notions of *unit-Monge matrices*, whose density matrices are permutation matrices (that is, binary matrices whose rows and columns contain exactly a single 1 entry each) and *simple Monge matrices*, whose leftmost column and bottommost row consist of zeroes. He also proved that the product of two simple unit-Monge matrices still belongs to this class and can be computed in $\mathcal{O}(n \log n)$ time provided that each matrix A is represented using the underlying permutation P_A [41]. By now, the resulting algorithm has found numerous applications, including for computing LCS and edit distance of compressed strings [19, 20, 24, 41], maintaining these similarity measures for dynamic strings [13, 22], approximate pattern matching [14, 40], parallel and distributed algorithms for similarity measures [31, 33], and oracles for substring similarity [12, 35, 36]. Furthermore, Tiskin's algorithm has been used to solve the LIS problem in various settings, such as dynamic [28],

[1] Throughout this paper, we use $\widetilde{\mathcal{O}}(\cdot)$ notation to suppress factors poly-logarithmic in the input size.

parallel [9], and distributed [30]. A disadvantage of Tiskin's original description (and even the later informal descriptions [29]) is its dependence on the algebraic structure known as the monoid of seaweed braids, which natively supports unweighted LCS only (tasks involving edit distance need to be reduced to LCS counterparts). This makes the algorithm difficult to generalize to weighted problems and extend even to seemingly simple questions such as recovering (an implicit representation of) the witness indices j such that $C_{i,k} = A_{i,j} + B_{j,k}$ [27].

Russo [34] identified the number of non-zero elements in $A^{\square}$, $B^{\square}$, and $(A \otimes B)^{\square}$, called the *core size* δ, as the parameter that enables fast min-plus matrix multiplication. It is easy to see that A, B, and $A \otimes B$ can be stored in $\mathcal{O}(n+\delta)$ space using a *condensed representation*, consisting of the *boundary entries* (the leftmost column and bottommost row, e.g., suffice) and *core elements*, i.e., the non-zero entries of the density matrix. Then, Russo's algorithm for the *core-sparse Monge matrix multiplication* problem computes the condensed representation of $A \otimes B$ in $\mathcal{O}((n+\delta)\log^2 n)$ time when provided with constant-time random access to A and B, and in $\mathcal{O}((n+\delta)\log^3 n)$ time given condensed representations of A and B.[2] Russo's algorithm has a very different structure than Tiskin's: it relies on a clever adaptation of the SMAWK algorithm [1] to provide efficient random access to C and then employs binary search to find individual non-zero entries of $C^{\square}$. This brings the question of unifying both approaches and understanding the complexity of core-sparse Monge matrix multiplication.

Our Results. We consider the core-sparse Monge matrix multiplication problem from the combinatorial and algorithmic points of view, and confirm that the core size is the right parameter that enables fast min-plus matrix multiplication. Let $\delta(A)$, or the core size of A, denote the number of non-zero elements in $A^{\square}$. We begin with analyzing, in Section 3, how the core size of $A \otimes B$ depends on the core sizes of A and B.

▶ **Theorem 1.1.** *Let A be a $p \times q$ Monge matrix, and let B be a $q \times r$ Monge matrix. We have $\delta(A \otimes B) \le 2 \cdot (\delta(A) + \delta(B))$.*

We stress that this the first bound on $\delta(A \otimes B)$ in terms of $\delta(A)$ and $\delta(B)$: the complexity analysis of Russo's algorithm [34] requires a bound on all $\delta(A)$, $\delta(B)$, and $\delta(A \otimes B)$.

Next, in Section 4, we generalize Tiskin's algorithm (but fully avoiding the formalism of the seaweed product) to solve the core-sparse Monge matrix multiplication problem. We believe that the more elementary interpretation makes our viewpoint not only more robust but also easier to understand. At the same time, the extension from simple unit-Monge matrices to general core-sparse Monge matrices introduces a few technical complications handled in the full version of the paper [21]. Notably, we need to keep track of the leftmost column and the bottommost row instead of assuming they are filled with zeroes. Further, the core does not form a permutation so splitting it into two halves of the same size requires some calculations.

▶ **Theorem 1.2.** *There is a (deterministic) algorithm that, given the condensed representations of a $p \times q$ Monge matrix A and a $q \times r$ Monge matrix B, in time $\mathcal{O}(p+q+r+(\delta(A)+\delta(B))\log(1+\delta(A)+\delta(B)))$ computes the condensed representation of $A \otimes B$.*[3]

[2] More precisely, Russo's algorithm builds a data structure that provides $\mathcal{O}(\log n)$-time random access to the entries of $A \otimes B$. Consequently, for repeated multiplication, we cannot assume constant-time access to the input matrices. Hence, $\mathcal{O}((n+\delta)\log^3 n)$ is a more realistic bound.

[3] In the technical sections of the paper we use the $\mathcal{O}(\cdot)$-notation conservatively. Specifically, we interpret $\mathcal{O}(f(x_1,\ldots,x_k))$ as the set of functions $g(x_1,\ldots,x_k)$ for which there are constants $c_g, N_g > 0$ such that $g(x_1,\ldots,x_k) \le c_g \cdot f(x_1,\ldots,x_k)$ holds for all valid tuples $(x_1,\ldots,x_k)$ satisfying $\max_i x_i \ge N_g$.

The above complexity improves upon Russo's [34] and matches Tiskin's [41] for the simple unit-Monge case. Thanks to the more direct description, we can easily extend our algorithm to build (in the same complexity) an $\mathcal{O}(n+\delta)$-size data structure that, given (i,k), in $\mathcal{O}(\log n)$ time computes the smallest *witness* j such that $C_{i,k} = A_{i,j} + B_{j,k}$.

Applications. As an application of our witness recovery functionality, we consider the problem of range LIS queries. This task asks to preprocess an integer array $s[0..n)$ so that, later, given two indices $0 \le i < j \le n$, the longest increasing subsequence (LIS) of the sub-array $s[i..j)$ can be reported efficiently. Tiskin [38, 41] showed that the LIS size ℓ can be reported in $\mathcal{O}(\log n)$ time after $\mathcal{O}(n \log^2 n)$-time preprocessing. It was unclear, though, how to efficiently report the LIS itself. The recent work of Karthik C. S. and Rahul [27] achieved $\mathcal{O}(n^{1/2} \log^3 n + \ell)$-time reporting (correct with high probability) after $\mathcal{O}(n^{3/2} \log^3 n)$-time preprocessing. It is fairly easy to use our witness recovery algorithm to deterministically support $\mathcal{O}(\ell \log^2 n)$-time reporting after $\mathcal{O}(n \log^2 n)$-time preprocessing; see Section 5 for an overview of this result. As further shown in the full version of the paper [21], the reporting time can be improved to $\mathcal{O}(\ell \log n)$ and, with the preprocessing time increased to $\mathcal{O}(n \log^3 n)$, all the way to $\mathcal{O}(\ell)$.

▶ **Theorem 1.3.** *For every parameter $\alpha \in [0,1]$, there exists an algorithm that, given an integer array $s[0..n)$, in time $\mathcal{O}(n \log^{3-\alpha} n)$ builds a data structure that can answer range LIS* reporting *queries in time $\mathcal{O}(\ell \log^{\alpha} n)$, where ℓ is the length of the reported sequence.*

In particular, there is an algorithm with $\mathcal{O}(n \log^3 n)$ preprocessing and $\mathcal{O}(\ell)$ reporting time and an algorithm with $\mathcal{O}(n \log^2 n)$ preprocessing and $\mathcal{O}(\ell \log n)$ reporting time.

In parallel to this work, Gorbachev and Kociumaka [22] used core-sparse Monge matrix multiplication for the *weighted edit distance* with integer weights. Theorem 1.2 allowed for saving two logarithmic factors in the final time complexities compared to the initial preprint using Russo's approach [34]. Weighted edit distance is known to be reducible to unweighted LCS only for a very limited class of so-called uniform weight functions [38], so this application requires the general core-sparse Monge matrix multiplication.

Open Problem. An interesting open problem is whether any non-trivial trade-off can be achieved for the *weighted* version of range LIS queries, where each element of s has a weight, and the task is to compute a maximum-weight increasing subsequence of $s[i..j)$: either the weight alone or the whole subsequence. Surprisingly, as we show in the full version of the paper [21], if our bound $\delta(A \otimes B) \le 2 \cdot (\delta(A) + \delta(B))$ of Theorem 1.1 can be improved to $\delta(A \otimes B) \le c \cdot (\delta(A) + \delta(B)) + \widetilde{\mathcal{O}}(p+q+r)$ for some $1 \le c < 2$, then our techniques automatically yield a solution with $\widetilde{\mathcal{O}}(n^{1+\log_2 c})$ preprocessing time, $\widetilde{\mathcal{O}}(1)$ query time, and $\widetilde{\mathcal{O}}(\ell)$ reporting time.

2 Preliminaries

▶ **Definition 2.1.** *A matrix A of size $p \times q$ is a* Monge matrix *if it satisfies the following* Monge property *for every $i \in [0..p-1)$ and $j \in [0..q-1)$:*

$$A_{i,j} + A_{i+1,j+1} \le A_{i,j+1} + A_{i+1,j}.$$

Furthermore, A is an anti-Monge matrix *if the matrix $-A$ (with negated entries) is a Monge matrix.*

Accordingly, whenever the expression inside $\mathcal{O}(\cdot)$ depends on multiple parameters, we sometimes add 1 or 2 to the arguments of logarithms to ensure formal correctness in corner cases.

In some sources, the following equivalent (but seemingly stronger) condition is taken as the definition of Monge matrices.

▶ **Observation 2.2.** *A matrix A of size $p \times q$ is a Monge matrix if and only if it satisfies the following inequality for all integers $0 \le a \le b < p$ and $0 \le c \le d < q$:*

$$A_{a,c} + A_{b,d} \le A_{a,d} + A_{b,c}.$$

Proof. It suffices to sum the inequality in Definition 2.1 for all $i \in [a..b)$ and $j \in [c..d)$. ◀

▶ **Definition 2.3.** *The* min-plus product *of a matrix A of size $p \times q$ and a matrix B of size $q \times r$ is a matrix $A \otimes B = C$ of size $p \times r$ satisfying $C_{i,k} = \min_{j \in [0..q)} A_{i,j} + B_{j,k}$ for all $i \in [0..p)$ and $k \in [0..r)$.*

For $i \in [0..p)$ and $k \in [0..r)$, we call $j \in [0..q)$ a witness *of $C_{i,k}$ if and only if $C_{i,k} = A_{i,j} + B_{j,k}$. We define the $p \times r$* witness matrix *$\mathcal{W}^{A,B}$ such that $\mathcal{W}^{A,B}_{i,k}$ is the smallest witness of $C_{i,k}$ for each $i \in [0..p)$ and $k \in [0..r)$.*

The min-plus product of two Monge matrices is also Monge.

▶ **Fact 2.4** ([48, Corollary A])**.** *Let A, B, and C be matrices such that $A \otimes B = C$. If A and B are Monge, then C is also Monge.*

In the context of Monge matrices, it is useful to define the core and the density matrix.

▶ **Definition 2.5.** *The* density matrix *of a matrix A of size $p \times q$ is a matrix $A^{\square}$ of size $(p-1)\times(q-1)$ satisfying $A^{\square}_{i,j} = A_{i,j+1} + A_{i+1,j} - A_{i,j} - A_{i+1,j+1}$ for $i \in [0..p-1), j \in [0..q-1)$.*

We define the core *of the matrix A as*

$$\operatorname{core}(A) \coloneqq \{(i, j, A^{\square}_{i,j}) \mid i \in [0..p-1), j \in [0..q-1), A^{\square}_{i,j} \neq 0\}$$

and denote the core size *of the matrix A by $\delta(A) \coloneqq |\operatorname{core}(A)|$. Furthermore, we define the* core sum *of the matrix A as the sum of the values of all core elements of A, that is,*

$$\delta^{\Sigma}(A) \coloneqq \sum_{i \in [0..p-1)} \sum_{j \in [0..q-1)} A^{\square}_{i,j}.$$

Note that, for a Monge matrix A, all entries of its density matrix are non-negative, and thus $\operatorname{core}(A)$ consists of triples (i, j, v) with some positive values v.

For any matrix A of size $p \times q$ and integers $0 \le a < b \le p$ and $0 \le c < d \le q$, we write $A[a..b)[c..d)$ to denote the contiguous submatrix of A consisting of all entries on the intersection of rows $[a..b)$ and columns $[c..d)$ of A. Matrices $A[a..b][c..d]$, $A(a..b][c..d)$, etc., are defined analogously.

3 Properties of δ and δ^{Σ}

In this section, we provide some useful properties of δ and δ^{Σ}. Most importantly, we show how to bound $\delta(A \otimes B)$ in terms of $\delta(A)$ and $\delta(B)$.
The following observation is a straightforward consequence of the definitions of δ^{Σ} and $A^{\square}$.

▶ **Observation 3.1.** *For a matrix A of size $p \times q$, and integers $0 \le a \le b < p$ and $0 \le c \le d < q$,*

$$A_{a,c} + A_{b,d} + \delta^{\Sigma}(A[a..b][c..d]) = A_{a,d} + A_{b,c}.$$

Proof. By the definition of $A^{\square}$, we have $A_{i,j} + A_{i+1,j+1} + A^{\square}_{i,j} = A_{i,j+1} + A_{i+1,j}$ for all $i \in [a..b)$ and $j \in [c..d)$. The desired equality follows by summing up all these equalities. ◀

An application of Observation 3.1 for $a = c = 0$ implies that every value of A can be uniquely reconstructed from A's core and the values in the topmost row and the leftmost column of A.

▶ **Definition 3.2.** *The* condensed representation *of a matrix A consists of the core of A as well as the values in the topmost row and the leftmost column of A.*

▶ **Observation 3.3.** *Any submatrix A' (not necessarily contiguous) of a Monge matrix A is Monge.*

Proof. By Observation 2.2, if A is Monge, the Monge property holds for every (not necessarily contiguous) 2×2 submatrix of A. In particular, it holds for contiguous 2×2 submatrices of A', and thus A' satisfies Definition 2.1. ◀

We next show a crucial property of the witness matrix.

▶ **Lemma 3.4** ([48, Theorem 1])**.** *For any two Monge matrices, A of size $p \times q$ and B of size $q \times r$, the witness matrix $\mathcal{W}^{A,B}$ is non-decreasing by rows and columns.*

Proof. We prove that $\mathcal{W}^{A,B}$ is non-decreasing by columns. The claim for rows is analogous. Fix some $i \in [0..p-1)$ and $k \in [0..r)$. Let $j^* \coloneqq \mathcal{W}^{A,B}_{i,k}$ and $j \in [0..j^*)$. As j^* is the smallest witness for (i,k), we have $A_{i,j} + B_{j,k} > A_{i,j^*} + B_{j^*,k}$. Using this fact and the Monge property for A, we derive

$$\begin{aligned} A_{i+1,j} + B_{j,k} &\geq (A_{i,j} + A_{i+1,j^*} - A_{i,j^*}) + B_{j,k} \\ &= (A_{i,j} + B_{j,k}) + (A_{i+1,j^*} - A_{i,j^*}) \\ &> (A_{i,j^*} + B_{j^*,k}) + (A_{i+1,j^*} - A_{i,j^*}) \\ &= A_{i+1,j^*} + B_{j^*,k}. \end{aligned}$$

Since $A_{i+1,j} + B_{j,k} > A_{i+1,j^*} + B_{j^*,k}$ for all $j \in [0..j^*)$, we conclude that $\mathcal{W}^{A,B}_{i+1,k} \geq j^* = \mathcal{W}^{A,B}_{i,k}$ holds as required. ◀

We now show how to bound $\delta^{\Sigma}(A \otimes B)$ in terms of $\delta^{\Sigma}(A)$ and $\delta^{\Sigma}(B)$.

▶ **Lemma 3.5.** *Let A be a $p \times q$ Monge matrix, and let B be a $q \times r$ Monge matrix. Then, $\delta^{\Sigma}(A \otimes B) \leq \min\{\delta^{\Sigma}(A), \delta^{\Sigma}(B)\}$.*

Proof. Let $C \coloneqq A \otimes B$. Denote $j \coloneqq \mathcal{W}^{A,B}_{0,0}$ and $j' \coloneqq \mathcal{W}^{A,B}_{p-1,r-1}$, where $j \leq j'$ due to Lemma 3.4. We have $C_{0,0} = A_{0,j} + B_{j,0}$ and $C_{p-1,r-1} = A_{p-1,j'} + B_{j',r-1}$. Furthermore, $C_{p-1,0} \leq A_{p-1,j} + B_{j,0}$ and $C_{0,r-1} \leq A_{0,j'} + B_{j',r-1}$ due to the definition of C. Hence, due to Observation 3.1, we obtain

$$\begin{aligned} \delta^{\Sigma}(C) &= C_{p-1,0} + C_{0,r-1} - C_{0,0} - C_{p-1,r-1} \\ &\leq (A_{p-1,j} + B_{j,0}) + (A_{0,j'} + B_{j',r-1}) - (A_{0,j} + B_{j,0}) - (A_{p-1,j'} + B_{j',r-1}) \\ &= A_{p-1,j} + A_{0,j'} - A_{0,j} - A_{p-1,j'} \\ &= \delta^{\Sigma}(A[0..p)[j..j']) \\ &\leq \delta^{\Sigma}(A). \end{aligned}$$

The inequality $\delta^{\Sigma}(C) \leq \delta^{\Sigma}(B)$ can be obtained using a symmetric argument. ◀

The following corollary says that every core element of $A \otimes B$ can be attributed to at least one core element of A and at least one core element of B.

▶ **Corollary 3.6.** *Let A be a $p \times q$ Monge matrix, let B be a $q \times r$ Monge matrix, and let $C := A \otimes B$. Consider integers $i \in [0..p-1)$ and $k \in [0..r-1)$ such that $C^{\square}_{i,k} \neq 0$. There exist integers $j_A, j_B \in [\mathcal{W}^{A,B}_{i,k} .. \mathcal{W}^{A,B}_{i+1,k+1})$ such that $A^{\square}_{i,j_A} \neq 0$ and $B^{\square}_{j_B,k} \neq 0$.*

Proof. Let $j = \mathcal{W}^{A,B}_{i,k}$ and $j' = \mathcal{W}^{A,B}_{i+1,k+1}$. By monotonicity of $\mathcal{W}^{A,B}$ (Lemma 3.4), we have $j \le \mathcal{W}^{A,B}_{i,k+1}, \mathcal{W}^{A,B}_{i+1,k} \le j'$. Thus, $C[i..i+1][k..k+1] = A[i..i+1][j..j'] \otimes B[j..j'][k..k+1]$. Due to Lemma 3.5, we have $0 < C^{\square}_{i,k} = \delta^{\Sigma}(C[i..i+1][k..k+1]) \le \delta^{\Sigma}(A[i..i+1][j..j'])$. Thus, there exists a core element in $A^{\square}[i..i+1)[j..j')$.

Symmetrically, $0 < C^{\square}_{i,k} = \delta^{\Sigma}(C[i..i+1][k..k+1]) \le \delta^{\Sigma}(B[j..j'][k..k+1])$, and there exists a core element in $B^{\square}[j..j')[k..k+1)$. ◀

We now use Corollary 3.6 to derive a bound on $\delta(A \otimes B)$ in terms of $\delta(A)$ and $\delta(B)$. The underlying idea is to show that every core element of C is either the first or the last one (in the lexicographic ordering) that the mapping of Corollary 3.6 attributes to the corresponding core element of A *or* B (see Figure 1 for why the opposite would lead to a contradiction).

▶ **Theorem 1.1.** *Let A be a $p \times q$ Monge matrix, and let B be a $q \times r$ Monge matrix. We have $\delta(A \otimes B) \le 2 \cdot (\delta(A) + \delta(B))$.*

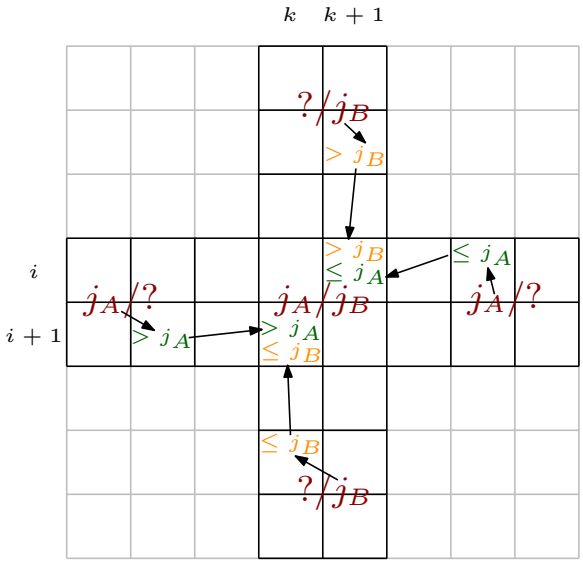


Figure 1 We consider the matrix of witnesses $\mathcal{W}^{A,B}$, drawn as an array with rows in $[0..p)$ indexed form top to bottom and columns in $[0..r)$ indexed from left to right. In the cells, we write bounds on the corresponding witnesses. In the grid nodes, we write the two values in $[0..q-1)$ derived from Corollary 3.6 for each core element of C. The center of the picture corresponds to a core value $C^{\square}_{i,k}$, attributed to $A^{\square}_{i,j_A}$ and $B^{\square}_{j_B,k}$, respectively. For a proof by contradiction, suppose that there are some core elements to the left and right of (i,k) attributed to $A^{\square}_{i,j_A}$ and some core elements above and below (i,k) attributed to $B^{\square}_{j_B,k}$. The arrows represent implications (based on Corollary 3.6 and Lemma 3.4), from which a contradiction follows: $j_A < \mathcal{W}^{A,B}_{i+1,k} \le j_B < \mathcal{W}^{A,B}_{i,k+1} \le j_A$.

Proof. Let $C \coloneqq A \otimes B$. Define a function $f_A : \mathrm{core}(C) \to \mathrm{core}(A)$ that maps every core element $(i, k, C^{\square}_{i,k}) \in \mathrm{core}(C)$ to $(i, j, A^{\square}_{i,j})$ for the smallest $j \in [\mathcal{W}^{A,B}_{i,k} .. \mathcal{W}^{A,B}_{i+1,k+1})$ with $A^{\square}_{i,j} \neq 0$; such a j exists due to Corollary 3.6. Analogously, we define a function $f_B : \mathrm{core}(C) \to \mathrm{core}(B)$ that maps every core element $(i, k, C^{\square}_{i,k}) \in \mathrm{core}(C)$ to $(j, k, B^{\square}_{j,k})$ for the smallest $j \in [\mathcal{W}^{A,B}_{i,k} .. \mathcal{W}^{A,B}_{i+1,k+1})$ with $B^{\square}_{j,k} \neq 0$; again, such a j exists due to Corollary 3.6.

▷ Claim 3.7. Every core element $c \in \mathrm{core}(C)$ is the lexicographically minimal or maximal one in its pre-image $f_A^{-1}(f_A(c))$ under f_A or its pre-image $f_B^{-1}(f_B(c))$ under f_B.

Proof. For a proof by contradiction, pick an element $c \coloneqq (i, k, C^{\square}_{i,k}) \in \mathrm{core}(C)$ violating the claim. Let $(i, j_A, A^{\square}_{i,j_A}) \coloneqq a \coloneqq f_A(c)$ and $(j_B, k, B^{\square}_{j_B,k}) \coloneqq b \coloneqq f_B(c)$. We make four symmetric arguments (all illustrated in Figure 1), with the first one presented in more detail.

1. Since c is not the minimal core element in $f_A^{-1}(a)$, we have $f_A(c') = a$ for some core element $(i', k', C^{\square}_{i',k'}) \coloneqq c' \in \mathrm{core}(C)$ that precedes c in the lexicographical order, denoted $c' \prec c$. Due to $f_A(c') = a = (i, j_A, A^{\square}_{i,j_A})$, we must have $i' = i$ and $j_A < \mathcal{W}^{A,B}_{i+1,k'+1}$. Since $i' = i$, then $c' \prec c$ implies $k' + 1 \leq k$. The monotonicity of the witness matrix (Lemma 3.4) thus yields $\mathcal{W}^{A,B}_{i+1,k'+1} \leq \mathcal{W}^{A,B}_{i+1,k}$. Overall, we conclude that $j_A < \mathcal{W}^{A,B}_{i+1,k}$.
2. Since c is not the maximal core element in $f_A^{-1}(a)$, we have $f_A(c') = a$ for some core element $c' \succ c$. In particular, $c' = (i, k', C^{\square}_{i,k'}) \in \mathrm{core}(C)$ for some $k' > k$. Then, $\mathcal{W}^{A,B}_{i,k+1} \leq \mathcal{W}^{A,B}_{i,k'} \leq j_A$ follows from Lemma 3.4 and $f_A(c') = a$, respectively.
3. Since c is not the minimal core element in $f_B^{-1}(b)$, we have $f_B(c') = b$ for some core element $c' \prec c$. In particular, $c' = (i', k, C^{\square}_{i',k}) \in \mathrm{core}(C)$ for some $i' < i$. Then, $\mathcal{W}^{A,B}_{i,k+1} \geq \mathcal{W}^{A,B}_{i'+1,k+1} > j_B$ follows from Lemma 3.4 and $f_B(c') = b$, respectively.
4. Since c is not the maximal core element in $f_B^{-1}(b)$, we have $f_B(c') = b$ for some core element $c' \succ c$. In particular, $c' = (i', k, C^{\square}_{i',k}) \in \mathrm{core}(C)$ for some $i' > i$. Then, $\mathcal{W}^{A,B}_{i+1,k} \leq \mathcal{W}^{A,B}_{i',k} \leq j_B$ follows from Lemma 3.4 and $f_B(c') = b$, respectively.

Overall, we derive a contradiction: $j_A < \mathcal{W}^{A,B}_{i+1,k} \leq j_B < \mathcal{W}^{A,B}_{i,k+1} \leq j_A$. ◁

As every core element of C is either the first or the last one in some pre-image under f_A or f_B, we get that $\delta(C) \leq 2 \cdot (\delta(A) + \delta(B))$. ◀

▶ **Example 3.8.** Note that the inequality for the core size from Theorem 1.1 is weaker than the inequality for the core sum from Lemma 3.5. We claim that this weakening is not an artifact of our proof. Consider the following Monge matrices

$$A = \begin{pmatrix} 0 & 4 & 5 & 6 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 0 \\ 0 & 1 & 2 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 0 & 2 & 4 & 6 \\ 0 & 0 & 2 & 4 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \qquad C = \begin{pmatrix} 0 & 2 & 4 & 6 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

We have $C = A \otimes B$, $\delta(A) = 2$, $\delta(B) = 3$, and $6 = \delta(C) > \delta(A) + \delta(B) > \min\{\delta(A), \delta(B)\}$.

▶ Remark 3.9. To the best of our knowledge, Theorem 1.1 shows the first bound on the core size of $A \otimes B$ in terms of the core sizes of A and B. Previously, such a bound was only known for unit-Monge matrices [41]. In particular, our bound allows simplifying some time complexities of already existing algorithms. For example, [34, Lemma 18] shows how to compute the product of two explicitly given Monge matrices A and B in time $\mathcal{O}(p \log q + q \log r + \delta(A \otimes B) \log q \log r + (\delta(A) + \delta(B)) \log^2 q)$. Assuming that $r \in \mathrm{poly}(q)$, Theorem 1.1 tells us that we can drop the third summand in the time complexity.

4 Core-Sparse Monge Matrix Min-Plus Multiplication Algorithm

In this section, we present an algorithm that, given the condensed representations of two Monge matrices, computes the condensed representation of their $(\min,+)$-product. Even though the condensed representation itself does not provide efficient random access to the values of the matrix, the following fact justifies our choice of the representation of Monge matrices.

▶ **Fact 4.1** ([22, Lemma 4.4]). *There exists an algorithm that, given the condensed representation of a matrix A of size $p \times q$, in time $\mathcal{O}(p+q+\delta(A)\log(1+\delta(A)))$ builds a* core-based matrix oracle data structure $\mathsf{CMO}(A)$ *that provides $\mathcal{O}(\log(2+\delta(A)))$-time random access to the entries of A.*

Proof Sketch. By Observation 3.1 applied for $a = c = 0$, it suffices to answer orthogonal range sum queries on top of $\operatorname{core}(A)$. According to [45, Theorem 3], it can be done in $\mathcal{O}(\delta(A)\log(1+\delta(A)))$ preprocessing time and $\mathcal{O}(\log(2+\delta(A)))$ query time. ◀

▶ **Definition 4.2.** *Let A be a $p \times q$ matrix. For every $i \in [0..p-1)$, denote* $\operatorname{core}_{i,\cdot}(A) = \{(i, j, A^{\square}_{i,j}) \mid j \in [0..q-1), A^{\square}_{i,j} \neq 0\}$ *and* $\delta_{i,\cdot}(A) = |\operatorname{core}_{i,\cdot}(A)|$. *Analogously, for every $j \in [0..q-1)$, denote* $\operatorname{core}_{\cdot,j}(A) = \{(i, j, A^{\square}_{i,j}) \mid i \in [0..p-1), A^{\square}_{i,j} \neq 0\}$ *and* $\delta_{\cdot,j}(A) = |\operatorname{core}_{\cdot,j}(A)|$.

▶ **Observation 4.3.** $\delta(A) = \sum_{i\in[0..p-1)} \delta_{i,\cdot}(A) = \sum_{j\in[0..q-1)} \delta_{\cdot,j}(A)$.

We now design another matrix oracle as an alternative to $\mathsf{CMO}(A)$ for the situation in which one wants to query an entry of A that is adjacent to some other already known entry.

▶ **Lemma 4.4.** *There is an algorithm that, given the condensed representation of a $p \times q$ matrix A, in time $\mathcal{O}(p+q+\delta(A))$ builds a* local core oracle data structure $\mathsf{lco}(A)$ *with the following interface.*

Boundary access: *given indices $i \in [0..p)$ and $j \in [0..q)$ such that $i = 0$ or $j = 0$, in time $\mathcal{O}(1)$ returns $A_{i,j}$.*

Vertically adjacent recomputation: *given indices $i \in [0..p-1)$ and $j \in [0..q)$, in time $\mathcal{O}(\delta_{i,\cdot}(A)+1)$ returns $A_{i+1,j} - A_{i,j}$.*

Horizontally adjacent recomputation: *given indices $i \in [0..p)$ and $j \in [0..q-1)$, in time $\mathcal{O}(\delta_{\cdot,j}(A)+1)$ returns $A_{i,j+1} - A_{i,j}$.*

Proof. The local core oracle data structure of A stores all the values of the topmost row and the leftmost column of A, as well as two collections of lists: $\operatorname{core}_{i,\cdot}(A)$ for all $i \in [0..p-1)$ and $\operatorname{core}_{\cdot,j}(A)$ for all $j \in [0..q-1)$. The values of the topmost row and the leftmost column of A are already given. The lists $\operatorname{core}_{i,\cdot}(A)$ and $\operatorname{core}_{\cdot,j}(A)$ can be computed in $\mathcal{O}(p+q+\delta(A))$ time from $\operatorname{core}(A)$. Hence, we can build $\mathsf{lco}(A)$ in time $\mathcal{O}(p+q+\delta(A))$.

Boundary access can be implemented trivially. We now show how to implement the vertically adjacent recomputation. Suppose that we are given $i \in [0..p-1)$ and $j \in [0..q)$. Due to Observation 3.1, we have $A_{i,j} + A_{i+1,0} = A_{i+1,j} + A_{i,0} + \delta^{\Sigma}(A[i..i+1][0..j])$. Note that the values $A_{i,0}$ and $A_{i+1,0}$ can be computed in constant time using boundary access queries, and $\delta^{\Sigma}(A[i..i+1][0..j])$ can be computed from $\operatorname{core}_{i,\cdot}(A)$ in time $\mathcal{O}(\delta_{i,\cdot}(A)+1)$. Hence, $A_{i+1,j} - A_{i,j}$ can be computed in time $\mathcal{O}(\delta_{i,\cdot}(A)+1)$.

The horizontally adjacent recomputation is implemented analogously so that $A_{i,j+1} - A_{i,j}$ can be computed in time $\mathcal{O}(\delta_{\cdot,j}(A)+1)$. ◀

Note that vertically and horizontally adjacent recomputations allow computing the neighbors of any given entry $A_{i,j}$.

▶ **Lemma 4.5.** *Given the condensed representation of a $p \times q$ matrix A, the condensed representation of any contiguous submatrix A' of A can be computed in time $\mathcal{O}(p+q+\delta(A))$.*

Proof. Say, $A' = A[i..i'][j..j']$. Note that $\mathrm{core}(A')$ can be obtained in time $\mathcal{O}(\delta(A))$ by filtering out all elements of $\mathrm{core}(A)$ that lie outside of $[i..i') \times [j..j')$. It remains to obtain the topmost row and the leftmost column of A'. In time $\mathcal{O}(p+q+\delta(A))$ we build $\mathsf{lco}(A)$ of Lemma 4.4. By starting from $A_{i,0}$ and repeatedly applying the horizontally adjacent recomputation, we can compute $A[i..i][0..q)$ (and thus $A[i..i][j..j']$) in time $\mathcal{O}(\sum_{j\in[0..q)}(\delta_{\cdot,j}(A)+1)) = \mathcal{O}(q+\delta(A))$. The leftmost column of A' can be computed in time $\mathcal{O}(p+\delta(A))$ analogously. ◀

We now show a helper lemma that compresses "ultra-sparse" Monge matrices to limit their dimensions by the size of their core.

▶ **Lemma 4.6** (Matrix compression)**.** *There are two algorithms* `Compress` *and* `Decompress` *with the following properties: The* `Compress` *algorithm, given the condensed representations of a $p^* \times q^*$ Monge matrix A^* and a $q^* \times r^*$ Monge matrix B^*, in time $\mathcal{O}(p^* + q^* + r^* + \delta(A^*) + \delta(B^*))$, builds a $p \times q$ Monge matrix A and a $q \times r$ Monge matrix B such that $p \le \delta(A^*)+1$, $q \le \delta(A^*)+\delta(B^*)+1$, $r \le \delta(B^*)+1$, $\delta(A) = \delta(A^*)$, and $\delta(B) = \delta(B^*)$. The* `Decompress` *algorithm, given the condensed representations of A^*, B^*, and $A \otimes B$, where $(A, B) =$ `Compress`(A^*, B^*), computes the condensed representation of $A^* \otimes B^*$ in time $\mathcal{O}(p^* + r^* + \delta(A^*) + \delta(B^*))$.*

Proof Sketch. The `Compress` algorithm deletes all rows of A^* that do not contain any core elements and all columns of B^* that do not contain any core elements. This way, $p \le \delta(A^*)+1$ and $r \le \delta(B^*)+1$ are guaranteed. Furthermore, all core elements of these two matrices are preserved. If there are no core elements between rows i and $i+1$ of A^*, the corresponding entries of these two rows differ by some constant c, and thus all the entries of the row $i+1$ of $A^* \otimes B^*$ can be obtained from the corresponding entries of row i of $A^* \otimes B^*$ by adding c. Since we can recover c from A^*, no information about the answer is "lost" when deleting such rows. An analogous property holds for the removed columns of B^*. The `Decompress` algorithm reverses the row and column removals performed by the `Compress` algorithm.

The `Compress` algorithm also reduces the number q^* of columns of A^* and rows of B^* to $q \le \delta(A^*) + \delta(B^*) + 1$. Observe that if there are no core elements between columns j and $j+1$ of A^* and between rows j and $j+1$ of B^*, then the values $A^*_{i,j} + B^*_{j,k}$ and $A^*_{i,j+1} + B^*_{j+1,k}$ differ by a constant independent of i and k. Depending on the sign of this constant difference, we can delete either the j-th column of A^* and the j-th row of B^* or the $(j+1)$-th column of A^* and the $(j+1)$-th row of B^* so that we do not change the min-plus product of these matrices. By repeating this process for all indices j that do not "contribute" to the cores of A^* and B^*, we get $q \le \delta(A^*) + \delta(B^*) + 1$. The formal proof can be found in the full version of the paper [21]. ◀

Finally, we show our main Monge matrix multiplication algorithm.

▶ **Theorem 1.2.** *There is a (deterministic) algorithm that, given the condensed representations of an $n_1 \times n_2$ Monge matrix M_1 and an $n_2 \times n_3$ Monge matrix M_2, in time $\mathcal{O}(n_1 + n_2 + n_3 + (\delta(M_1) + \delta(M_2)) \log(1 + \delta(M_1) + \delta(M_2)))$ computes the condensed representation of $M_1 \otimes M_2$.*

Proof Sketch. We design a recursive divide-and-conquer procedure `Multiply`(A^*, B^*) (see Algorithm 1 for the details) and solve the problem by initially running `Multiply`(M_1, M_2). Given the matrices A^* and B^*, we first apply the `Compress` algorithm of Lemma 4.6 to

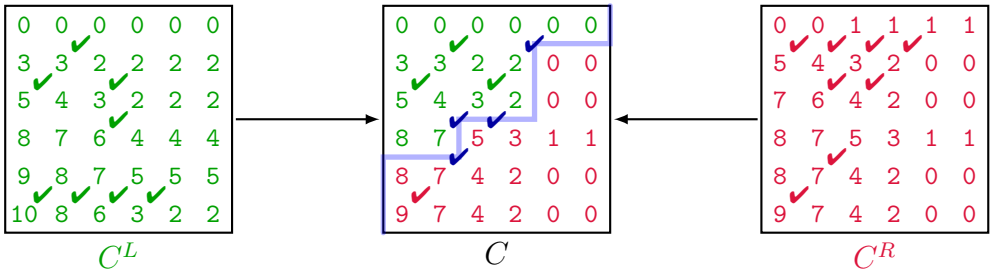


Figure 2 An example of how C is obtained from C^L and C^R. The blue ladder represents the border between the values that are inherited from C^L and the values that are inherited from C^R. Ticks correspond to the core elements: the green ticks are inherited from C^L, the red ticks are inherited from C^R, and the blue ticks represent the core element computed from scratch.

compress A^* and B^* into a $p \times q$ matrix A and a $q \times r$ matrix B respectively. We then compute the condensed representation of $C \coloneqq A \otimes B$ and finally use the `Decompress` algorithm of Lemma 4.6 to decompress C into $A^* \otimes B^*$.

If $p = r = 1$, we compute the matrix C trivially. Otherwise, we pick a splitting point $m \in (0..q)$, split the matrix A vertically into the matrix A^L of size $p \times m$ and the matrix A^R of size $p \times (q-m)$, and split the matrix B horizontally into the matrix B^L of size $m \times r$ and the matrix B^R of size $(q-m) \times r$. We pick m in such a way that it splits the cores of A and B almost equally across A^L and B^L and A^R and B^R, that is, $\delta(A^L) + \delta(B^L)$ and $\delta(A^R) + \delta(B^R)$ are at most $(\delta(A) + \delta(B))/2$. We recursively compute the condensed representations of the matrices $C^L \coloneqq A^L \otimes B^L$ and $C^R \coloneqq A^R \otimes B^R$. The resulting matrix C can be obtained as the element-wise minimum of C^L and C^R. Furthermore, one can see that, due to Lemma 3.4, in some top-left region of C, the values are equal to the corresponding values of C^L, and in the remaining bottom-right region of C, the values are equal to the corresponding values of C^R; see Figure 2. We find the boundary between these two regions by starting in the bottom-left corner of C and traversing it towards the top-right corner along this boundary. We use $\mathsf{lco}(C^L)$ and $\mathsf{lco}(C^R)$ to sequentially compute the subsequent entries along the boundary. Having determined the boundary, we construct the core of C by picking the core elements of C^L located to the top-left of this boundary, picking the core elements of C^R located to the bottom-right of this boundary, and computing the $C^\square$ values on the boundary from scratch. The values in the topmost row and the leftmost column of C can be trivially obtained as element-wise minima of the corresponding values in C^L and C^R. This concludes the recursive procedure; see Algorithm 1 for the pseudocode. This algorithm follows the classical divide-and-conquer framework, and thus its time complexity can be easily derived from Theorem 1.1. The full description of the algorithm, the proof of its correctness, and the formal analysis of its time complexity can be found in the full version of the paper [21]. ◀

If the $(\min, +)$-product of matrices represents the lengths of the shortest paths in a graph, the corresponding witness matrix allows computing the underlying shortest paths themselves. For that, we extend the presented algorithm to allow witness reconstruction.

▶ **Theorem 4.7.** *The algorithm of Theorem 1.2 can be extended so that, within the same time complexity, it also builds a data structure that takes $\mathcal{O}(n_1 + n_2 + n_3 + \delta(M_1) + \delta(M_2))$ space and provides $\mathcal{O}(\log(2 + \delta(M_1) + \delta(M_2)))$-time oracle access to $\mathcal{W}^{M_1, M_2}$.*

Algorithm 1 The algorithm from Theorem 1.2. Given the condensed representations of the Monge matrices A^* and B^*, the algorithm returns the condensed representation of $A^* \otimes B^*$.

```
1  Multiply(A*, B*) begin
2      A, B ← Compress(A*, B*);
3      p ← height of A; q ← width of A; r ← width of B;
4      if p = 1 and r = 1 then
5          C ← 1 × 1 matrix with the entry equal to min{A_{0,j} + B_{j,0} | j ∈ [0..q)};
6          return Decompress(A*, B*, C);
7      Sort core(A) by columns and core(B) by rows;                    // needed to find m
8      Pick the largest m ∈ (0..q) with
         δ(A[0..p)[0..m)) + δ(B[0..m)[0..r)) ≤ (δ(A) + δ(B))/2;
9      Build the condensed representations of A[0..p)[0..m) and B[0..m)[0..r);
10     C^L ← Multiply(A[0..p)[0..m), B[0..m)[0..r)); Build lco(C^L) using Lemma 4.4;
11     Build the condensed representations of A[0..p)[m..q) and B[m..q)[0..r);
12     C^R ← Multiply(A[0..p)[m..q), B[m..q)[0..r)); Build lco(C^R) using Lemma 4.4;
13     lst ← list of length p;                  // vertical edges of the ladder from Figure 2
14     i ← p − 1, j ← −1;      // current matrix coordinates while traversing the ladder
15     Maintain the well-defined values in C^L[i..i+1][j..j+1] and C^R[i..i+1][j..j+1]
         using lco(C^L) and lco(C^R), and the values in C[i..i+1][j..j+1] as element-wise
         minima;
16     Core ← {};
17     while i ≥ 0 do
18         if i < p − 1 and 0 ≤ j < r − 1 and C_{i,j+1} + C_{i+1,j} ≠ C_{i,j} + C_{i+1,j+1} then
19             Core ← Core ∪ {(i, j, C_{i,j+1} + C_{i+1,j} − C_{i,j} − C_{i+1,j+1})};
20         if j = r − 1 or C^L_{i,j+1} > C^R_{i,j+1} then
21             lst_i ← j;
22             i ← i − 1;
23         else
24             j ← j + 1;
25     Core ← Core ∪ {(i, j, C^{L□}_{i,j}) ∈ core(C^L) | j < lst_{i+1}} ∪ {(i, j, C^{R□}_{i,j}) ∈ core(C^R) |
         j > lst_i};
26     Compute the topmost row and the leftmost column of C as the element-wise
         minima of the topmost rows and the leftmost columns of C^L and C^R;
27     Condensed representation of C ← (C[0..0][0..r), C[0..p)[0..0], Core);
28     return Decompress(A*, B*, C);
```

Proof Sketch. We slightly modify the algorithm of Theorem 1.2. For the leaf recursive calls with $p = r = 1$, we explicitly store the minimal witness of the only entry of C. In the non-leaf recursive calls, we store the correspondence between the indices in the compressed matrices A, B, C and the decompressed matrices A^*, B^*, C^*, as well as the border separating the values of C^L and C^R in C (the blue curve of Figure 2). Naively, this data takes space proportional to the time complexity of Theorem 1.2. Nevertheless, using bit-masks equipped with rank/select functionality, the total space complexity can be brought down to be linear in terms of the size of the input. To answer a query, we descend the recursion tree. In constant time we can find an entry of C corresponding to the queried entry of C^* and decide on which side of the border separating the values of C^L and C^R the queried entry of C is.

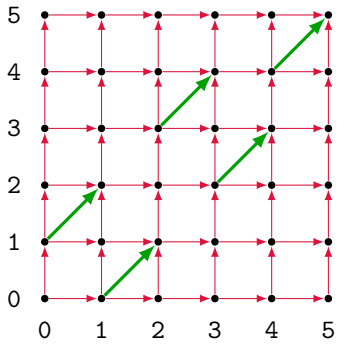


Figure 3 The alignment graph for $s = [1, 0, 3, 2, 4]$, with red edges of weight 0 and green edges of weight 1. The length of the longest path from the i-th vertex in the bottom row to the j-th vertex in the top row of this graph for $i < j$ is equal to $\mathsf{LIS}(s[i..j))$.

After that, we recurse into either $A^L \otimes B^L$ or $B^L \otimes B^R$. In the terminal recursion call with $p = r = 1$ we return the minimal witness that is stored explicitly. The time complexity of the algorithm is proportional to the depth of recursion. See the full version of the paper [21] for a formal description of the algorithm. ◀

5 Range LIS Queries: Sketch

One of the original applications of Tiskin's procedure for simple unit-Monge matrix multiplication [38, 41] is an algorithm that preprocesses a given sequence $(s[i])_{i\in[0..n)}$ of integers in $\mathcal{O}(n \log^2 n)$ time so that *Range Longest Increasing Subsequence* (Range LIS) queries can be answered in $\mathcal{O}(\log n)$ time. We show an alternative way of obtaining the same result using Theorem 1.2 and avoiding the seaweed braid formalism. The extension of Theorem 4.7 allows recovering the underlying longest increasing subsequence of length ℓ in $\mathcal{O}(\ell \log^2 n)$ time. Further $\mathcal{O}(n \log^3 n)$-time preprocessing allows for $\mathcal{O}(\ell)$-time recovery, which improves upon the result of [27].

Without loss of generality we assume that $(s[i])_{i\in[0..n)}$ is a permutation of $[0..n)$.[4] We use a popular tool for string similarity problems and interpret range LIS queries as computing the longest path between some pair of vertices of a corresponding *alignment graph* G^s; see Figure 3. The crucial property of G^s is that it is planar, and thus due to [18, Section 2.3], the matrix M^s of longest distances from the vertices in the bottom row to the vertices of the top row of this graph is anti-Monge.[5] Thus, the problem of constructing a data structure for range LIS queries is reduced to the problem of computing the condensed representation of M^s.

As $\mathsf{LIS}(s[i..j)) \le n$ for all $i, j \in [0..n]$ with $i < j$, all entries of M^s are bounded by $\mathcal{O}(n)$, and thus $\delta(M^s) = \mathcal{O}(n)$ holds due to Observation 3.1 and the fact that $\delta(M^S) \le \delta^\Sigma(M^S)$ holds for integer-valued matrices. We compute M^s in a divide-and-conquer fashion. We split the sequence s into subsequences s^{lo} and s^{hi} containing all values in $[0..\lfloor \frac{n}{2} \rfloor)$ and $[\lfloor \frac{n}{2} \rfloor..n)$ respectively, and recursively compute $M^{s^{\mathrm{lo}}}$ and $M^{s^{\mathrm{hi}}}$. After that, we note that $G^{s^{\mathrm{lo}}}$ and $G^{s^{\mathrm{hi}}}$

[4] In what follows, we reserve the word "permutation" for permutations of $[0..m)$ for some $m \in \mathbb{Z}_+$.
[5] Note that, in reality, some entries of this matrix are infinite. In the formal description of the algorithm, we augment the alignment graph so that the finite entries of this matrix are preserved and the infinite entries become finite.

are essentially compressed versions of the lower half and the upper half of G^s, respectively. Based on this, we transform $M^{s^{\text{lo}}}$ in $\mathcal{O}(n)$ time into the matrix of the longest distances from the vertices in the bottom row of G^s to the vertices in the middle row of G^s. Analogously, we transform $M^{s^{\text{hi}}}$ into the matrix of the longest distances from the middle row of G^s to the top row of G^s. To obtain M^s, it remains to $(\max, +)$-multiply these two matrices using Theorem 1.2. Every single recursive call takes time $\mathcal{O}(n \log n)$ dominated by the algorithm of Theorem 1.2. The whole divide-and-conquer procedure takes $\mathcal{O}(n \log^2 n)$ time. Given the condensed representation of M^s, we use Fact 4.1 to create an oracle for $\mathcal{O}(\log n)$-time range LIS queries, thus replicating the result of [41].

Compared to the results of [41], this algorithm operates directly on the local LIS values and thus can be easily converted into an algorithm for the reporting version of range LIS queries, where we want to not only find the length of the longest path in the alignment graph but to also compute the structure of the underlying path itself. For that, we simply use the witness reconstruction oracle of Theorem 4.7 to find the midpoint of the path and descend the recursion tree. This costs $\mathcal{O}(\log n)$ time per recursive call and allows reconstructing the entire length-ℓ LIS in time $\mathcal{O}(\ell \log^2 n)$. A similar recursive LIS reporting scheme is used in [9].

By treating increasing subsequences of length at most $\log^2 n$ separately, we further obtain an algorithm with $\mathcal{O}(n \log^3 n)$ preprocessing time and $\mathcal{O}(\ell)$ reporting time. It improves the result of [27] with $\mathcal{O}(n^{3/2} \operatorname{polylog} n)$-time preprocessing and $\mathcal{O}(\ell + n^{1/2} \operatorname{polylog} n)$-time reporting.

The formal proofs of the results in this section are given in the full version of the paper [21].

References

1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.

2 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997. doi:10.1006/JCSS.1997.1388.

3 Alberto Apostolico, Mikhail J. Atallah, Lawrence L. Larmore, and Scott McFaddin. Efficient parallel algorithms for string editing and related problems. *SIAM Journal on Computing*, 19(5):968–988, 1990. doi:10.1137/0219066.

4 Gary Benson. A space efficient algorithm for finding the best nonoverlapping alignment score. *Theoretical Computer Science*, 145(1&2):357–369, 1995. doi:10.1016/0304-3975(95)92848-R.

5 Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM Journal on Computing*, 46(4):1280–1303, 2017. doi:10.1137/15M1042929.

6 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM Journal on Computing*, 48(2):481–512, 2019. doi:10.1137/17M112720X.

7 Rainer E. Burkard. Monge properties, discrete convexity and applications. *European Journal of Operational Research*, 176(1):1–14, 2007. doi:10.1016/J.EJOR.2005.04.050.

8 Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996. doi:10.1016/0166-218X(95)00103-X.

9 Nairen Cao, Shang-En Huang, and Hsin-Hao Su. Nearly optimal parallel algorithms for longest increasing subsequence. In Kunal Agrawal and Julian Shun, editors, *35th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2023*, pages 249–259. ACM, 2023. doi:10.1145/3558481.3591078.

10 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010. doi:10.1137/08071990X.

11 Panagiotis Charalampopoulos, Paweł Gawrychowski, Yaowei Long, Shay Mozes, Seth Pettie, Oren Weimann, and Christian Wulff-Nilsen. Almost optimal exact distance oracles for planar graphs. *Journal of the ACM*, 70(2):12:1–12:50, 2023. doi:10.1145/3580474.

12 Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. An almost optimal edit distance oracle. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 48:1–48:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.48.

13 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Shay Mozes. Dynamic string alignment. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, volume 161 of *LIPIcs*, pages 9:1–9:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CPM.2020.9.

14 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster pattern matching under edit distance: A reduction to dynamic puzzle matching and the seaweed monoid of permutation matrices. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 698–707. IEEE, 2022. doi:10.1109/FOCS54457.2022.00072.

15 Shucheng Chi, Ran Duan, and Tianle Xie. Faster algorithms for bounded-difference min-plus product. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *33rd ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 1435–1447. SIAM, 2022. doi:10.1137/1.9781611977073.60.

16 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In Stefano Leonardi and Anupam Gupta, editors, *54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.

17 Anita Dürr. Improved bounds for rectangular monotone min-plus product and applications. *Information Processing Letters*, 181:106358, 2023. doi:10.1016/J.IPL.2023.106358.

18 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. doi:10.1016/j.jcss.2005.05.007.

19 Arun Ganesh, Tomasz Kociumaka, Andrea Lincoln, and Barna Saha. How compression and approximation affect efficiency in string distance measures. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *33rd ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 2867–2919. SIAM, 2022. doi:10.1137/1.9781611977073.112.

20 Paweł Gawrychowski. Faster algorithm for computing the edit distance between SLP-compressed strings. In Liliana Calderón-Benavides, Cristina N. González-Caro, Edgar Chávez, and Nivio Ziviani, editors, *19th International Symposium on String Processing and Information Retrieval, SPIRE 2012*, volume 7608 of *LNCS*, pages 229–236. Springer, 2012. doi:10.1007/978-3-642-34109-0_24.

21 Paweł Gawrychowski, Egor Gorbachev, and Tomasz Kociumaka. Core-sparse Monge matrix multiplication: Improved algorithm and applications, 2024. arXiv:2408.04613v2.

22 Egor Gorbachev and Tomasz Kociumaka. Bounded edit distance: Optimal static and dynamic algorithms for small integer weights. In *57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 2157–2166, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3717823.3718168.

23 Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhan Xu. Faster monotone min-plus product, range mode, and single source replacement paths. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 75:1–75:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.75.

24 Danny Hermelin, Gad M. Landau, Shir Landau, and Oren Weimann. Unified compression-based acceleration of edit-distance computation. *Algorithmica*, 65(2):339–353, 2013. doi:10.1007/S00453-011-9590-6.

25 Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, and Piotr Sankowski. Decremental single-source reachability in planar digraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1108–1121. ACM, 2017. doi:10.1145/3055399.3055480.

26 Sampath Kannan and Eugene W. Myers. An algorithm for locating nonoverlapping regions of maximum alignment score. *SIAM Journal on Computing*, 25(3):648–662, 1996. doi:10.1137/S0097539794262677.

27 Karthik C. S. and Saladi Rahul. Range longest increasing subsequence and its relatives: Beating quadratic barrier and approaching optimality, 2024. doi:10.48550/arXiv.2404.04795.

28 Tomasz Kociumaka and Saeed Seddighin. Improved dynamic algorithms for longest increasing subsequence. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, pages 640–653. ACM, 2021. doi:10.1145/3406325.3451026.

29 Jaehyun Koo. On range LIS queries, 2023. Accessed: 2025-04-22. URL: https://codeforces.com/blog/entry/111625.

30 Jaehyun Koo. An optimal MPC algorithm for subunit-Monge matrix multiplication, with applications to LIS. In Kunal Agrawal and Erez Petrank, editors, *36th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2024*, pages 145–154. ACM, 2024. doi:10.1145/3626183.3659974.

31 Peter Krusche and Alexander Tiskin. New algorithms for efficient parallel string comparison. In Friedhelm Meyer auf der Heide and Cynthia A. Phillips, editors, *22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2010*, pages 209–216. ACM, 2010. doi:10.1145/1810479.1810521.

32 Gad M. Landau and Michal Ziv-Ukelson. On the common substring alignment problem. *Journal of Algorithms*, 41(2):338–359, 2001. doi:10.1006/JAGM.2001.1191.

33 Nikita Mishin, Daniil Berezun, and Alexander Tiskin. Efficient parallel algorithms for string comparison. In Xian-He Sun, Sameer Shende, Laxmikant V. Kalé, and Yong Chen, editors, *50th International Conference on Parallel Processing, ICPP 2021*, pages 50:1–50:10. ACM, 2021. doi:10.1145/3472456.3472489.

34 Luís M. S. Russo. Monge properties of sequence alignment. *Theoretical Computer Science*, 423:30–49, 2012. doi:10.1016/J.TCS.2011.12.068.

35 Yoshifumi Sakai. A substring-substring LCS data structure. *Theoretical Computer Science*, 753:16–34, 2019. doi:10.1016/J.TCS.2018.06.034.

36 Yoshifumi Sakai. A data structure for substring-substring LCS length queries. *Theoretical Computer Science*, 911:41–54, 2022. doi:10.1016/J.TCS.2022.02.004.

37 Jeanette P. Schmidt. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4):972–992, 1998. doi:10.1137/S0097539795288489.

38 Alexander Tiskin. Semi-local string comparison: algorithmic techniques and applications, 2007. arXiv:0707.3619.

39 Alexander Tiskin. Semi-local longest common subsequences in subquadratic time. *Journal of Discrete Algorithms*, 6(4):570–581, 2008. doi:10.1016/J.JDA.2008.07.001.

40 Alexander Tiskin. Threshold approximate matching in grammar-compressed strings. In Jan Holub and Jan Zdárek, editors, *Prague Stringology Conference 2014*, pages 124–138. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2014. URL: http://www.stringology.org/event/2014/p12.html.

41 Alexander Tiskin. Fast distance multiplication of unit-Monge matrices. *Algorithmica*, 71(4):859–888, 2015. doi:10.1007/S00453-013-9830-Z.

42 Alexandre Tiskin. Semi-local string comparison: Algorithmic techniques and applications. *Mathematics in Computer Science*, 1(4):571–603, 2008. doi:10.1007/S11786-007-0033-3.

43 Virginia Vassilevska and Ryan Williams. Finding a maximum weight triangle in $n^{3-\Delta}$ time, with applications. In Jon M. Kleinberg, editor, *38th Annual ACM Symposium on Theory of Computing, STOC 2006*, pages 225–231. ACM, 2006. doi:10.1145/1132516.1132550.

44 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *International Congress of Mathematicians, ICM 2018*, pages 3447–3487. World Scientific, 2018. doi:10.1142/9789813272880_0188.

45 Dan E. Willard. New data structures for orthogonal range queries. *SIAM Journal on Computing*, 14(1):232–253, 1985. doi:10.1137/0214019.

46 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

47 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In Shuchi Chawla, editor, *31st ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 12–29. SIAM, 2020. doi:10.1137/1.9781611975994.2.

48 F. Frances Yao. Speed-up in dynamic programming. *SIAM Journal on Algebraic Discrete Methods*, 3(4):532–540, 1982. doi:10.1137/0603055.

49 Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In Claire Mathieu, editor, *20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 950–957. SIAM, 2009. doi:10.1137/1.9781611973068.103.

An Improved Bound for Plane Covering Paths

Hugo A. Akitaya
Miner School of Computer & Information Sciences, University of Massachusetts, Lowell, MA, USA

Greg Aloupis
Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA

Ahmad Biniaz
School of Computer Science, University of Windsor, Canada

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada

Jean-Lou De Carufel
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Cyril Gavoille
LaBRI, University of Bordeaux, France

John Iacono
Department of Computer Science, Université libre de Bruxelles, Belgium

Linda Kleist
Department of Computer Science, Universität Potsdam, Germany

Michiel Smid
School of Computer Science, Carleton University, Ottawa, Canada

Diane Souvaine
Department of Computer Science, Tufts University, Medford, MA, USA

Leonidas Theocharous
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Abstract

A *covering path* for a finite set P of points in the plane is a polygonal path such that every point of P lies on a segment of the path. The vertices of the path need not be at points of P. A covering path is *plane* if its segments do not cross each other. Let $\pi(n)$ be the minimum number such that every set of n points in the plane admits a plane covering path with at most $\pi(n)$ segments. We prove that $\pi(n) \leq \lceil 6n/7 \rceil$. This improves the previous best-known upper bound of $\lceil 21n/22 \rceil$, due to Biniaz (SoCG 2023). Our proof is constructive and yields a simple $O(n \log n)$-time algorithm for computing a plane covering path.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Covering Path, Upper Bound, Simple Algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.75

Funding *Hugo A. Akitaya*: Supported by the NSF award CCF-2348067.
Ahmad Biniaz: Supported in part by NSERC.
Prosenjit Bose: Supported in part by NSERC.
Jean-Lou De Carufel: Supported in part by NSERC.
Cyril Gavoille: Supported by the French ANR projects ANR-22-CE48-0001 (TEMPOGRAL) and ANR-24-CE48-7768-01 (ENEDISC).
John Iacono: This work was supported by the Fonds de la Recherche Scientifique-FNRS.
Michiel Smid: Supported in part by NSERC.
Leonidas Theocharous: Supported in part by NSERC.

Acknowledgements This work was initiated at the 12th Annual Workshop on Geometry and Graphs, held at the Bellairs Research Institute in Holetown, Barbados, in February 2025. The authors thank the organizers and participants.



© Hugo A. Akitaya, Greg Aloupis, Ahmad Biniaz, Prosenjit Bose, Jean-Lou De Carufel, Cyril Gavoille, John Iacono, Linda Kleist, Michiel Smid, Diane Souvaine, and Leonidas Theocharous;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

The problems of traversing points in the plane by polygonal paths are fundamental to discrete and computational geometry, and have been studied from both combinatorial and computational points of view. For instance one can refer to traversing a set of points by a non-crossing path [3, 14], or by a path that minimizes the total edge length [7, 28, 30, 31],[1] minimizes the longest edge length [10, 11], maximizes the shortest edge length [4, 5], minimizes the number of turns (or bends) [18, 32],[2] or minimizes the total or the largest turning angle [1, 2, 20].

In this paper we focus on traversing a set of points by a non-crossing polygonal path with a small number of segments. This problem has been studied before [12, 13, 18, 32], and is related to the classic nine-dot puzzle that asks to cover the vertices of a 3×3 grid by a polygonal path with no more than 4 segments [29], as in Figure 1.

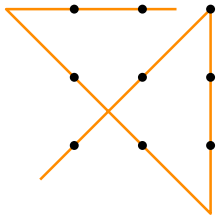


Figure 1 A non-plane covering path with four segments for the nine-dot puzzle.

Let P be a finite set of points in the plane. A *covering path* for P is a polygonal path (i.e., a path with straight-line segments) such that every point of P lies on some segment of the path. A vertex of a covering path can be any point in the plane (i.e., not necessarily a point in P). The segments of a covering path are also referred to as *links* [6] or *bends* [32] in the literature. A covering path with the smallest number of links is called a *minimum-link* or a *minimum-bend* covering path. A covering path is called *plane* or *non-crossing* if its segments do not cross each other. Figure 1 illustrates a covering path with 4 segments that is not plane.

In addition to their puzzling nature, dating back to 1914 [29], covering paths with a small number of segments are of interest in robotics and heavy machinery, where turning is an expensive operation [32]. Covering paths have also received considerable attention in recent years [6, 13, 18, 25]. Morić (in CCCG 2010) [17] and Welzl (in GWOP 2011) [17], and later Dumitrescu, Gerbner, Keszegh, and Tóth [18] raised challenging questions about covering paths, including the following.

▶ **Question 1.** *What is the minimum number $\pi(n)$ such that every set of n points in the plane can be covered by a non-crossing path with at most $\pi(n)$ straight-line segments?*

It is not hard to see that $n/2 \le \pi(n) \le n-1$, because any set of points can be covered by a plane path of $n-1$ segments (say by connecting the points from left to right), and each straight-line segment can cover at most two points (if no three points are collinear). Dumitrescu et al. [18] presented the first non-trivial lower and upper bounds:

$$\frac{5n}{9} - O(1) < \pi(n) < \left(1 - \frac{1}{601\,080\,391}\right) n.$$

[1] Known as the traveling salesperson path problem.
[2] Known as the minimum bend traveling salesperson problem.

The upper bound has been improved to $\lceil 21n/22 \rceil$ by Biniaz [12, 13]. The upper bounds are universal (i.e., any point set admits these bounds), while the lower bounds are existential (i.e., there exist point sets that achieve these bounds).

1.1 Our Contributions

We present an algorithm that constructs a plane covering path of at most $\lceil 6n/7 \rceil$ segments for any set of n points in the plane, which implies that $\pi(n) \leq \lceil 6n/7 \rceil$. The algorithm runs in $O(n \log n)$ time. This is optimal, because the problem of computing a plane covering path (even without the minimality constraint) has an $\Omega(n \log n)$ lower bound in the worst case in the algebraic decision tree model of computation – by a reduction from sorting [18].

Simplicity and differences to previous algorithms. Our algorithm shares similarities with the algorithms of Biniaz [12, 13] and Dumitrescu et al. [18] in terms of iteratively scanning the points from left to right. One limitation of the previous approaches is the use of *caps* and *cups* to save a segment in each iteration. A *k-cap* is a concave x-monotone chain of k points, and a *k-cup* is a convex x-monotone chain of k points. Dumitrescu et al. construct an 18-cap or 18-cup in each iteration of their algorithm [18], while Biniaz uses 5-caps or 5-cups [12, 13]. To ensure the existence of such caps or cups, at least 601080391 and 21 points are required, respectively, as implied by the Erdős-Szekeres theorem [19]. In contrast, our algorithm uses convex chains of points that are not necessarily x-monotone (hence neither a cap nor a cup).

Besides using a cap or a cup in each iteration, the previous algorithms [13, 18] partition the remaining scanned points (within the iteration) into certain subsets such that each subset is contained in a convex region, and the convex regions of all subsets are disjoint. The algorithms compute a covering path within each region, and then concatenate the paths and the cap or cup. The partitioning and the concatenation are nicely crafted so that the resulting path is plane. Our iterations are more straightforward and do not use complex partitioning – we consider three main cases based on the size of the convex hull of 5 points.

1.2 Related Problems and Results

If we allow crossings, one can obtain better upper bounds on the number of segments of covering paths. For example by repeatedly applying the Erdős-Szekeres theorem [19], one can obtain a covering path with at most $n/2 + O(n/\log n)$ segments [17, 18]. The optimization problem, which is to compute a minimum-link covering path for a set of points, is shown NP-hard by Arkin et al. [6]. The best known approximation algorithm, due to Stein and Wagner [32], has a ratio of $O(\log z)$ where z is the maximum number of collinear points.

A *covering tree* for a point set P is a tree with straight-line edges that cover every point of P. Covering trees are useful in red-blue separation [22], and in the construction of rainbow polygons [13, 21]. Let $\tau(n)$ be the minimum number such that every set of n points in the plane can be covered by a plane tree with at most $\tau(n)$ edges. It is known that $9n/17 - O(1) \leq \tau(n) \leq \lceil 4n/5 \rceil$ where the lower and upper bounds come from [18] and [13], respectively. The exact values of $\pi(n)$ and $\tau(n)$ for the vertices of the square grid have been determined by Keszegh [25].

Two other well-studied related problems to mention are covering points with minimum-link[3] axis-aligned paths [8, 9, 16, 24] or with a minimum number of lines [15, 23, 26, 27].

[3] For axis-aligned paths this is usually referred to as minimum-turn.

2 The Algorithm

In this section we prove the following theorem.

▶ **Theorem 1.** *Every set of n points in the plane admits a non-crossing covering path with at most $\lceil 6n/7 \rceil$ segments. Thus, $\pi(n) \leqslant \lceil 6n/7 \rceil$. Such a path can be constructed in $O(n \log n)$ time.*

Our proof is constructive. We present an algorithm that computes, for any set P of n points in the plane, a non-crossing covering path with at most $\lceil 6n/7 \rceil$ segments.

Preliminaries. We denote the convex hull of a point set S by $\mathrm{CH}(S)$. For two points p and q we denote by $\ell(p, q)$ the line through p and q, by pq the line segment with endpoints p and q, and by $\overrightarrow{pq}$ the ray that emanates from p and passes through q. The concatenation of two paths, δ_1 and δ_2, that share an endpoint is denoted by $\delta_1 \oplus \delta_2$.

Our algorithm is iterative and scans the points from left to right. Note that via a suitable rotation we may assume that no two points of P have the same x-coordinate. In the first iteration we scan only one point. In each subsequent iteration except for the last, we scan 6 or 7 new points and cover them with 5 or 6 new segments, respectively. Thus the ratio of the number of new segments to the number of covered points in these *intermediate* iterations is at most $6/7$. In the last iteration we scan the (at most 6) remaining points.

We assume that, among the scanned points in each iteration, no three are collinear. Collinear points are typically easier to handle because one can cover three points by one segment – such cases can be handled by an argument similar to that of [13]. Let m represent the number of points that have been scanned so far and let l' represent the rightmost scanned point. We maintain the following invariant at the beginning of every iteration after the first.

Invariant. All m points that have been scanned so far are covered by a non-crossing path δ with at most $\lfloor 6m/7 \rfloor$ segments. The path δ lies to the left of the vertical line through l'. The degree of l' in δ is one if $m \geqslant 2$ and zero if $m = 1$.

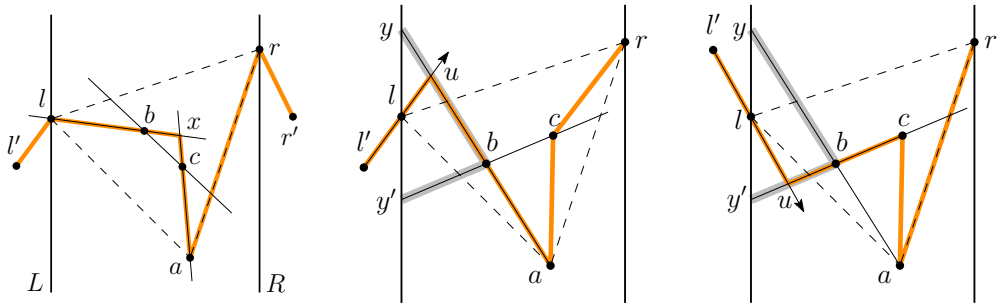
The invariant holds after the first iteration, trivially. We now focus on how to handle each intermediate iteration; this is the main part of our algorithm and proof. The last iteration is described at the end. Note that the invariant has the floor-function, whereas Theorem 1 has the ceiling-function. As we will see, at the beginning of each iteration, the m points scanned so far are covered by at most $\lfloor 6m/7 \rfloor$ segments. After the last iteration, we have $m = n$ and this upper bound becomes $\lceil 6n/7 \rceil$.

In each intermediate iteration the new points are appended to the current path δ in a suitable way, without altering the structure of δ. In our descriptions we will use Δ to denote the resulting path.

We start by scanning 6 new points. Denote these points by l, a, b, c, r, r', where l is the leftmost, r' is the rightmost and r is the second rightmost point, as in Figure 2. Let $S = \{l, a, b, c, r\}$. Thus, S has all newly scanned points except for r'. Let L and R be the vertical lines through l and r, respectively. We refer to the region between L and R as *the slab*. Observe that l and r are both vertices of the boundary of $\mathrm{CH}(S)$. We consider three cases depending on the number of vertices on $\mathrm{CH}(S)$.

Case 1. $\mathrm{CH}(S)$ has 3 vertices. Without loss of generality, assume that a is the third vertex of $\mathrm{CH}(S)$, located below $\ell(l, r)$. Thus b and c are inside triangle $\triangle(a, l, r)$. The line $\ell(b, c)$ intersects exactly two edges of $\triangle(a, l, r)$. We consider three subcases.

- $\ell(b,c)$ intersects $\ell(l,r)$ and ar, as shown in Figure 2(a). The points l, b, c, and a form the vertices of a convex quadrilateral. After a suitable relabeling, we may assume that they appear in this order along the boundary of the quadrilateral. Then the intersection x of $\ell(a,c)$ and $\ell(l,b)$ lies in $\triangle(a,l,r)$. We form the path path (l',l,x,a,r,r') and append it to δ. Thus we cover the six new vertices (l,b,c,a,r,r') with five segments. The savings come from avoiding using segment bc.



(a) (b) (c)

Figure 2 CH(S) has three vertices. (a) $\ell(b,c)$ intersects lr and ar. (b) $\ell(b,c)$ intersects al and ar, and $\overrightarrow{l'l}$ intersects by. (c) $\ell(b,c)$ intersects al and ar, and $\overrightarrow{l'l}$ intersects by'.

- $\ell(b,c)$ intersects lr and al. This is symmetric to the previous case. We let l' and l play the role of r' and r. Points a,b,c,r form a convex quadrilateral, and we create the path (l',l,a,x,r,r').
- $\ell(b,c)$ intersects al and ar. We may assume, without loss of generality, that b is to the left of c. Then $\overrightarrow{ab}$ intersects L or $\overrightarrow{ac}$ intersects R. Because of symmetry, assume that $\overrightarrow{ab}$ intersects L. Their intersection, y, must be above l. Denote by y' the intersection point of $\ell(b,c)$ with L, as in Figure 2(b). Observe that y' is below l. In this setting $\overrightarrow{l'l}$ intersects either by or by', at a point u. If $\overrightarrow{l'l}$ intersects by, we set $\Delta = \delta \oplus (l',u,a,c,r,r')$ as in Figure 2(b), otherwise we set $\Delta = \delta \oplus (l',u,c,a,r,r')$ as in Figure 2(c). Either way we cover 6 vertices with 5 segments, by avoiding lb on our path.

Case 2. CH(S) has 5 vertices (i.e., l, a, b, c, r). By reflection and relabeling, there are two subcases to consider.

- The three vertices a, b, and c lie on the same side of lr (*wlog* below). We may assume that they appear in the order a,b,c from left to right; see Figure 3(a). Then the intersection point x of $\ell(l,a)$ and $\ell(b,c)$ lies in the slab. We set $\Delta = \delta \oplus (l',l,x,c,r,r')$.
- Two vertices, say a and b, lie on one side of lr (*wlog* below), and c lies on the other side. Assume that a is to the left of b. The intersection, u, of $\ell(l,a)$ and $\ell(r,b)$ must be in the slab, by convexity. If l' is above $\ell(l,c)$ then we set $\Delta = \delta \oplus (l',c,l,u,r,r')$, as shown in Figure 3(b). Symmetrically, if r' is above $\ell(r,c)$, we set $\Delta = \delta \oplus (l',l,u,r,c,r')$.
  Finally if l' is below $\ell(l,c)$ and r' is below $\ell(r,c)$, let v be the intersection point of $\ell(l',l)$ and $\ell(r,c)$; see Figure 3(c). By convexity, v is in the slab, so we can set $\Delta = \delta \oplus (l',v,r,a,b,r')$.

Case 3. CH(S) has 4 vertices: l,r,a,b. We consider two subcases.

Case 3a. a and b lie on different sides of lr. *Wlog,* a is below. The point c lies either in triangle $\triangle(a,b,l)$ or in triangle $\triangle(a,b,r)$. The configurations are symmetric so we show how

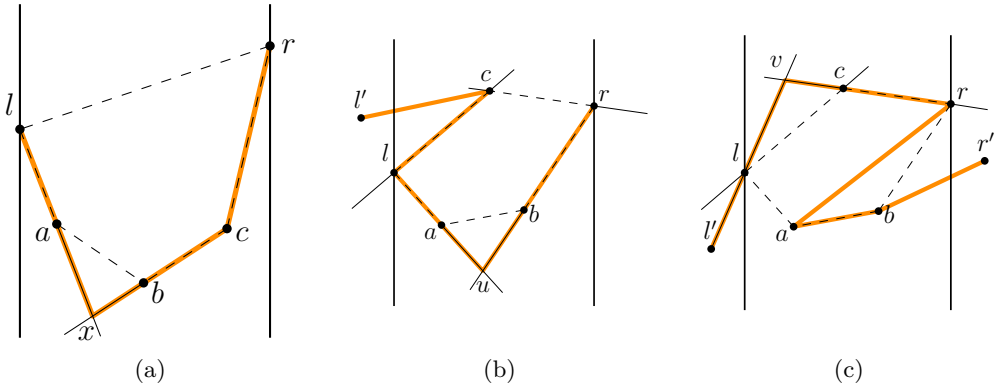


Figure 3 CH(S) has five vertices. (a) a, b, c lie on the same side of lr. (b) l' is above $\ell(l, c)$. (c) l' is below $\ell(l, c)$ and r' is below $\ell(r, c)$.

to handle the former. It is implied that c is to the left of $\overrightarrow{ab}$, which in turn implies that at least one of $\overrightarrow{ac}$ or $\overrightarrow{bc}$ intersects L. *Wlog,* assume that $\overrightarrow{ac}$ intersects L. The intersection point, y, must be above l. There are two subcases to consider:

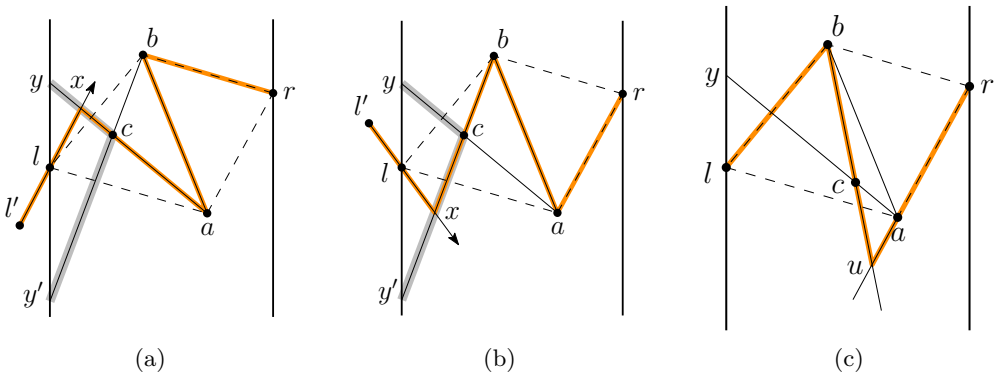


Figure 4 CH(S) has four vertices, the points a and b lie on different sides of lr, the point c is in $\triangle(a, b, l)$, and $\overrightarrow{ac}$ intersects L. (a) $\overrightarrow{bc}$ intersects L, and $\overrightarrow{l'l}$ intersects cy. (b) $\overrightarrow{bc}$ intersects L, and $\overrightarrow{l'l}$ intersects cy'. (c) $\overrightarrow{bc}$ intersects R.

- $\overrightarrow{bc}$ intersects L, at point y', which must be below l. Then either $\overrightarrow{l'l}$ intersects cy as in Figure 4(a), or it intersects cy' as in Figure 4(b). Either way, let the intersection point be x. Respectively, we set $\Delta = \delta \oplus (l', x, a, b, r, r')$ or we set $\Delta = \delta \oplus (l', x, b, a, r, r')$.
- $\overrightarrow{bc}$ intersects R, as in Figure 4(c). In this case the intersection point u of $\ell(b, c)$ and $\ell(r, a)$ lies in the slab. We set $\Delta = \delta \oplus (l', l, b, u, r, r')$.

Case 3b. a and b are on the same side of lr, (*wlog,* below; and also a is to the left of b). By convexity, the intersection point, x, of $\ell(l, a)$ and $\ell(r, b)$ is in the slab. If l' lies above $\ell(l, c)$, as in Figure 5(a), we set $\Delta = \delta \oplus (l', c, l, x, r, r')$. Symmetrically, if r' lies above $\ell(r, c)$, then we set $\Delta = \delta \oplus (l', l, x, r, c, r')$. It remains to consider the case when l' is below $\ell(l, c)$ and r' is below $\ell(r, c)$. We consider two subcases.

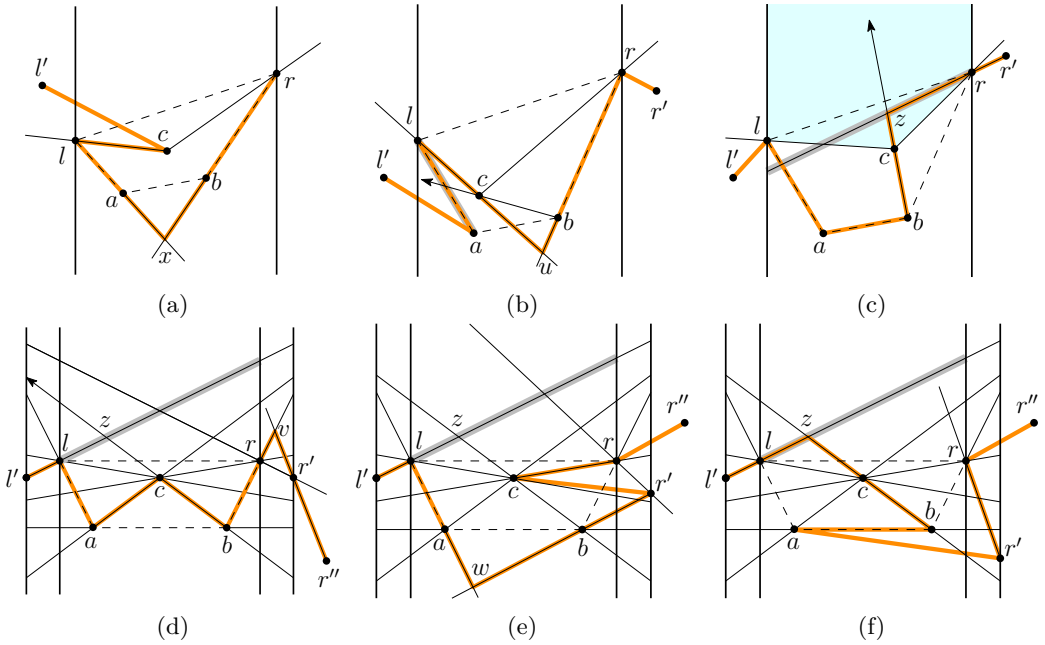


Figure 5 CH(S) has four vertices, a and b lie on the same side of lr. (a) l' is above $\ell(l,c)$. In (b)-(e) we have l' below $\ell(l,c)$ and r' below $\ell(r,c)$. (b) $\overrightarrow{bc}$ intersects al. (c) $\overrightarrow{bc}$ intersects $\overrightarrow{r'r}$ within the slab. (d) r'' is below $\ell(r,r')$. (e) r'' is above $\ell(r,r')$, and r' is above $\ell(a,b)$. (f) r'' is above $\ell(r,r')$, and r' is below $\ell(a,b)$.

- $\overrightarrow{bc}$ intersects al, as in Figure 5(b). In this case, the intersection point u of $\overrightarrow{lc}$ and $\overrightarrow{rb}$ is in the slab because c is to the right of l, and b is to the left of r. Moreover, since l is above $\overrightarrow{bc}$, u does not lie on either of the segments lc and rb. We set $\Delta = \delta \oplus (l', a, l, u, r, r')$. Symmetrically, if $\overrightarrow{ac}$ intersects br we can set $\Delta = \delta \oplus (l', l, u, r, b, r')$, but this does not seem to help or simplify the next case.
- $\overrightarrow{bc}$ does not intersect al. In this case $\overrightarrow{bc}$ lies in the wedge with counterclockwise boundary ray $\overrightarrow{bl}$ and clockwise boundary ray $\overrightarrow{br}$. Because l' is below $\ell(l,c)$ and r' is below $\ell(r,c)$, it follows that $\overrightarrow{bc}$ intersects at least one of the rays $\overrightarrow{l'l}$ and $\overrightarrow{r'r}$ within the slab. If the intersection is with $\overrightarrow{r'r}$ then the intersection point z must be inside the shaded region in Figure 5(c), i.e., above lr or in the triangle $\triangle(l,c,r)$. Accordingly, we set $\Delta = \delta \oplus (l', l, a, b, z, r')$. Otherwise, let z be the intersection of $\overrightarrow{bc}$ with $\overrightarrow{l'l}$. This is the case when we need to scan a new point.
  Let r'' be the next point, after r'. If r'' is below $\ell(r,r')$, then the intersection point v of $\ell(b,r)$ and $\ell(r'',r')$ is in the vertical slab between r and r'; we set $\Delta = \delta \oplus (l', l, a, c, b, v, r'')$ as in Figure 5(d). Otherwise, r'' is above $\ell(r,r')$. Now if r' is above $\ell(a,b)$, then the intersection point w of $\ell(l,a)$ and $\ell(r',b)$ lies in the slab, below $\ell(a,b)$; we set $\Delta = \delta \oplus (l', l, w, r', c, r, r'')$ as in Figure 5(e). Because r' is below $\ell(c,r)$, cr' does not intersect rr''. If r' is below $\ell(a,b)$, we set $\Delta = \delta \oplus (l', z, b, a, r', r, r'')$ as in Figure 5(f). In each case we cover seven new points (including r'') with six edges.

This is the end of an intermediate iteration. In each case, we append to δ a non-crossing path that lies in the vertical slab between l and the rightmost scanned point. Thus Δ is non-crossing. Moreover the degree of the rightmost point in Δ (which is r' or r'') is one.

In each case we have scanned 6 or 7 new points and covered them by 5 or 6 new segments, respectively. If we have scanned 6 points, then the total number M of scanned points is $M = m + 6$, and the number of segments of Δ is

$$|\Delta| = |\delta| + 5 \leqslant \left\lfloor \frac{6m}{7} \right\rfloor + 5 = \left\lfloor \frac{6m + 35}{7} \right\rfloor \leqslant \left\lfloor \frac{6M}{7} \right\rfloor .$$

If we have scanned 7 points, then $M = m + 7$, and we have

$$|\Delta| = |\delta| + 6 \leqslant \left\lfloor \frac{6m}{7} \right\rfloor + 6 = \left\lfloor \frac{6m + 42}{7} \right\rfloor = \left\lfloor \frac{6M}{7} \right\rfloor .$$

Therefore, the invariant holds after every intermediate iteration.

In the last iteration of the algorithm we are left with n' points where $1 \leqslant n' \leqslant 6$. We cover these points by an x-monotone path with $n' - 1$ segments and connect it to the rightmost scanned point (of the path that has been constructed so far) by an additional segment. This gives a non-crossing covering path for all points. The total number of segments in this path is at most

$$\left\lfloor \frac{6(n - n')}{7} \right\rfloor + (n' - 1) + 1 = \left\lfloor \frac{6(n - n') + 7n'}{7} \right\rfloor = \left\lfloor \frac{6n + n'}{7} \right\rfloor \leqslant \left\lfloor \frac{6n + 6}{7} \right\rfloor \leqslant \left\lceil \frac{6n}{7} \right\rceil .$$

Each iteration takes $O(1)$ time. Therefore, after rotating and sorting the points in $O(n \log n)$ time, the rest of the algorithm takes $O(n)$ time. This concludes the proof of Theorem 1.

▶ Remark 2. One could scan exactly 7 points in every intermediate iteration, and obtain the same upper bound. Our choice to scan 6 points whenever possible serves as a demonstration for the complexity of producing a non-crossing covering path with at most $5n/6$ segments.

▶ Remark 3. In contrast to the algorithms of [13] and [18], our algorithm does not rely on the existence of caps or cups to reduce the number of segments used. This has been illustrated in Figure 2(a) when c is to the right of a; in Figure 5(c) where z is to the left of c; and in Figure 5(f) when z is to the right of c.

▶ Remark 4. For the 7 points l', l, a, b, c, d, r, r' in Figure 5(d) we have not found a non-crossing path that starts at l', ends at r', covers all the points, and stays within the slab defined by l' and r'. This suggests the necessity of considering r'' in our algorithm.

3 Open Problems

A natural open problem is to narrow the gap between the lower bound $5n/9$ and the upper bound $6n/7$ for $\pi(n)$. It is not hard to see that a covering path with a minimum number of segments is a subset of the arrangement of the lines through the $\binom{n}{2}$ pairs of points. Thus one can find a covering path with a minimum number of segments in exponential time. It would be interesting to present a sub-exponential algorithm for computing such a path.

References

1 Alok Aggarwal, Don Coppersmith, Sanjeev Khanna, Rajeev Motwani, and Baruch Schieber. The angular-metric traveling salesman problem. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 221–229, 1997. URL: `http://dl.acm.org/citation.cfm?id=314161.314259`.

2 Alok Aggarwal, Don Coppersmith, Sanjeev Khanna, Rajeev Motwani, and Baruch Schieber. The angular-metric traveling salesman problem. *SIAM Journal on Computing*, 29(3):697–711, 1999. doi:10.1137/S0097539796312721.

3 Oswin Aichholzer, Sergio Cabello, Ruy Fabila Monroy, David Flores-Peñaloza, Thomas Hackl, Clemens Huemer, Ferran Hurtado, and David R. Wood. Edge-removal and non-crossing configurations in geometric graphs. *Discrete Mathematics & Theoretical Computer Science*, 12(1):75–86, 2010. doi:10.46298/DMTCS.525.

4 Esther M. Arkin, Yi-Jen Chiang, Joseph S. B. Mitchell, Steven Skiena, and Tae-Cheon Yang. On the maximum scatter TSP (extended abstract). In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 211–220, 1997. URL: http://dl.acm.org/citation.cfm?id=314161.314258.

5 Esther M. Arkin, Yi-Jen Chiang, Joseph S. B. Mitchell, Steven Skiena, and Tae-Cheon Yang. On the maximum scatter traveling salesperson problem. *SIAM Journal on Computing*, 29(2):515–544, 1999. doi:10.1137/S0097539797320281.

6 Esther M. Arkin, Joseph S. B. Mitchell, and Christine D. Piatko. Minimum-link watchman tours. *Information Processing Letters*, 86(4):203–207, 2003. doi:10.1016/S0020-0190(02)00502-1.

7 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998. doi:10.1145/290179.290180.

8 Sergey Bereg, Prosenjit Bose, Adrian Dumitrescu, Ferran Hurtado, and Pavel Valtr. Traversing a set of points with a minimum number of turns. In *Proceedings of the 23rd ACM Symposium on Computational Geometry (SCG)*, pages 46–55, 2007. doi:10.1145/1247069.1247077.

9 Sergey Bereg, Prosenjit Bose, Adrian Dumitrescu, Ferran Hurtado, and Pavel Valtr. Traversing a set of points with a minimum number of turns. *Discrete & Computational Geomometry*, 41(4):513–532, 2009. doi:10.1007/S00454-008-9127-1.

10 Ahmad Biniaz. Euclidean bottleneck bounded-degree spanning tree ratios. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 826–836, 2020. doi:10.1137/1.9781611975994.50.

11 Ahmad Biniaz. Euclidean bottleneck bounded-degree spanning tree ratios. *Discrete & Computational Geometry*, 67(1):311–327, 2022. doi:10.1007/S00454-021-00286-4.

12 Ahmad Biniaz. Improved bounds for covering paths and trees in the plane. In *39th International Symposium on Computational Geometry (SoCG)*, volume 258, pages 19:1–19:15, 2023. doi:10.4230/LIPICS.SOCG.2023.19.

13 Ahmad Biniaz. Improved bounds for covering paths and trees in the plane. *J. Comput. Geom.*, 15(1):88–108, 2024. doi:10.20382/JOCG.V15I1A4.

14 Jakub Cerný, Zdenek Dvorák, Vít Jelínek, and Jan Kára. Noncrossing Hamiltonian paths in geometric graphs. *Discrete Applied Mathematics*, 155(9):1096–1105, 2007. doi:10.1016/J.DAM.2005.12.010.

15 Jianer Chen, Qin Huang, Iyad Kanj, and Ge Xia. Near-optimal algorithms for point-line covering problems. *CoRR*, abs/2012.02363, 2020. arXiv:2012.02363.

16 Michael J. Collins. Covering a set of points with a minimum number of turns. *International Journal of Computational Geometry & Applications*, 14(1-2):105–114, 2004. doi:10.1142/S021819590400138X.

17 Erik D. Demaine and Joseph O'Rourke. Open problems from CCCG 2010. In *Proceedings of the 22nd Canadian Conference on Computational Geometry*, 2011.

18 Adrian Dumitrescu, Dániel Gerbner, Balázs Keszegh, and Csaba D. Tóth. Covering paths for planar point sets. *Discrete & Computational Geometry*, 51(2):462–484, 2014. doi:10.1007/S00454-013-9563-4.

19 P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

20 Sándor P. Fekete and Gerhard J. Woeginger. Angle-restricted tours in the plane. *Computational Geometry: Theory and Applications*, 8:195–218, 1997. doi:10.1016/S0925-7721(96)00012-0.

21 David Flores-Peñaloza, Mikio Kano, Leonardo Martínez-Sandoval, David Orden, Javier Tejel, Csaba D. Tóth, Jorge Urrutia, and Birgit Vogtenhuber. Rainbow polygons for colored point sets in the plane. *Discrete Mathematics*, 344(7):112406, 2021. doi:10.1016/J.DISC.2021.112406.

22 Radoslav Fulek, Balázs Keszegh, Filip Morić, and Igor Uljarević. On polygons excluding point sets. *Graphs and Combinatorics*, 29(6):1741–1753, 2013. doi:10.1007/S00373-012-1221-8.

23 Magdalene Grantson and Christos Levcopoulos. Covering a set of points with a minimum number of lines. In *Proceedings of the 6th International Conference on Algorithms and Complexity* (*CIAC*), pages 6–17, 2006. doi:10.1007/11758471_4.

24 Minghui Jiang. On covering points with minimum turns. *International Journal of Computational Geometry & Applications*, 25(1):1–10, 2015. doi:10.1142/S0218195915500016.

25 Balázs Keszegh. Covering paths and trees for planar grids. *Geombinatorics Quarterly*, 24, 2014.

26 Stefan Langerman and Pat Morin. Covering things with things. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, volume 2461 of *Lecture Notes in Computer Science*, pages 662–673. Springer, 2002. doi:10.1007/3-540-45749-6_58.

27 Stefan Langerman and Pat Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005. doi:10.1007/S00454-004-1108-4.

28 E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem*. John Wiley & Sons, New York, NY, 1985.

29 Samuel Loyd. *Cyclopedia of 5000 Puzzles, Tricks & Conundrums*. The Lamb Publishing Company, 1914.

30 Joseph S. B. Mitchell. Geometric shortest paths and network optimization. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. North Holland / Elsevier, 2000. doi:10.1016/B978-044482537-7/50016-4.

31 Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, 1977. doi:10.1016/0304-3975(77)90012-3.

32 Clifford Stein and David P. Wagner. Approximation algorithms for the minimum bends traveling salesman problem. In *Proceedings of the 8th International Conference on Integer Programming and Combinatorial Optimization* (*IPCO*), pages 406–422, 2001. doi:10.1007/3-540-45535-3_32.

A 3.3904-Competitive Online Algorithm for List Update with Uniform Costs

Mateusz Basiak
University of Wrocław, Poland

Marcin Bienkowski
University of Wrocław, Poland

Martin Böhm
University of Wrocław, Poland

Marek Chrobak
University of California, Riverside, CA, USA

Łukasz Jeż
University of Wrocław, Poland

Jiří Sgall
Charles University, Prague, Czech Republic

Agnieszka Tatarczuk
University of Wrocław, Poland

Abstract

We consider the List Update problem where the cost of each swap is assumed to be 1. This is in contrast to the "standard" model, in which an algorithm is allowed to swap the requested item with previous items for free. We construct an online algorithm Full-Or-Partial-Move (*Fpm*), whose competitive ratio is at most 3.3904, improving over the previous best known bound of 4.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases List update, work functions, amortized analysis, online algorithms, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.76

Related Version *Full Version*: `https://arxiv.org/abs/2503.17264` [14]

Funding Suppported by Polish National Science Centre grants 2022/45/B/ST6/00559, 2020/39/B/ST6/01679 and 2022/47/D/ST6/02864, National Science Foundation grant CCF-2153723, and by grant 24-10306S of GA ČR.

1 Introduction

The List Update problem. In the online *List Update problem* [3, 20, 21], the objective is to maintain a set of items stored in a linear list in response to a sequence of access requests. The cost of accessing a requested item is equal to its distance from the front of the list. After each request, an algorithm is allowed to rearrange the list by performing an arbitrary number of swaps of adjacent items. In the model introduced by Sleator and Tarjan in their seminal 1985 paper on competitive analysis [25], an algorithm can repeatedly swap the requested item with its preceding item at no cost. These swaps are called *free*. All other swaps are called *paid* and have cost 1 each. As in other problems involving self-organizing data structures [7], the goal is to construct an *online* algorithm, i.e., operating without the knowledge of future requests. The cost of such an algorithm is compared to the cost of the optimal *offline* algorithm; the ratio of the two costs is called the *competitive ratio* and is subject to minimization.

Sleator and Tarjan proved that the algorithm *Move-To-Front* (*Mtf*), which after each request moves the requested item to the front of the list, is 2-competitive [25]. This ratio is known to be optimal if the number of items is unbounded. Their work was the culmination of previous extensive studies of list updating, including experimental results, probabilistic approaches, and earlier attempts at amortized analysis (see [15] and the references therein).



© Mateusz Basiak, Marcin Bienkowski, Martin Böhm, Marek Chrobak, Łukasz Jeż, Jiří Sgall, and Agnieszka Tatarczuk;
licensed under Creative Commons License CC-BY 4.0



As shown in subsequent work, *Mtf* is not unique – there are other strategies that achieve ratio 2, such as *TimeStamp* [2] or algorithms based on work functions [9]. In fact, there are infinitely many algorithms that achieve ratio 2 [13, 21].

The uniform cost model. Following [24], we will refer to the cost model of [25] as *standard*, and we will denote it here by LUP_S. This model has been questioned in the literature for not accurately reflecting true costs in some implementations [24, 23, 22, 18], with the concept of free swaps being one of the main concerns.

A natural approach to address this concern, considered in some later studies (see, e.g., [24, 7, 21, 4, 12]), is simply to charge cost 1 for *any* swap. We will call it here the *uniform cost model* and denote it LUP_1. A general lower bound of 3 on the competitive ratio of deterministic algorithms for LUP_1 was given by Reingold *et al.* [24]. Changing the cost of free swaps from 0 to 1 at most doubles the cost of any algorithm, so *Mtf* is no worse than 4-competitive for LUP_1. Surprisingly, no algorithm is known to beat *Mtf*, i.e., achieve ratio lower than 4.[1].

Our main result. To address this open problem, we develop an online algorithm *Full-Or-Partial-Move* (*Fpm*) for LUP_1 with competitive ratio $\frac{1}{8}\cdot(23+\sqrt{17}) \approx 3.3904$, significantly improving the previous upper bound of 4.

Our algorithm *Fpm* remains 3.3904-competitive even for the *partial cost* function, where the cost of accessing location ℓ is $\ell - 1$, instead of the *full cost* of ℓ used in the original definition of List Update [25]. Both functions have been used in the literature, depending on context and convenience. For any online algorithm, its partial-cost competitive ratio is at least as large as its full-cost ratio, although the difference typically vanishes when the list size is unbounded. We present our analysis in terms of partial costs.

Fpm remains 3.3904-competitive also in the dynamic scenario of List Update that allows operations of insertions and deletions, as in the original definition in [25] (see the full version of the paper [14]).

Technical challenges and new ideas. The question whether ratio 3 can be achieved remains open. We also do not know if there is a *simple* algorithm with ratio below 4. We have considered some natural adaptions of *Mtf* and other algorithms that are 2-competitive for LUP_S, but for all we were able to show lower bounds higher than 3 for LUP_1.

As earlier mentioned, *Mtf* is 4-competitive for LUP_1. It is also easy to show that its ratio is not better than 4: repeatedly request the last item in the list. Ignoring additive constants, the algorithm pays twice the length of the list at each step, while any algorithm that just keeps the list in a fixed order, pays only half the length on the average.

The intuition why *Mtf* performs poorly is that it moves the requested items to front "too quickly". For the aforementioned adversarial strategy against *Mtf*, ratio 3 can be obtained by moving the items to front only *every other time* they are requested. This algorithm, called *Dbit*, is a deterministic variant of algorithm *Bit* from [24] and a special case of algorithm *Counter* in [24], and it has been also considered in [21]. In the full version of the paper [14], we show that *Dbit* is not better than 4-competitive in the partial cost model, and not better than 3.302-competitive in the full cost model.

[1] The authors of [21] claimed an upper bound of 3 for LUP_1, but later discovered that their proof was not correct (personal communication with S. Kamali).

One can generalize these approaches by considering a more general class of algorithms that either leave the requested item at its current location or move it to the front. We show that such a strategy cannot achieve ratio better than 3.25, even for just three items. A naive fix would be, for example, to always move the requested item half-way towards front. This algorithm is even worse: its competitive ratio is at least 6 (both those proofs are in the full version of the paper [14]).

We also show (see Section 5) via a computer-aided argument that, for LUP_1, the work function algorithm's competitive ratio is larger than 3, even for lists of length 5. This is in contrast to the its performance for the standard model, where it achieves optimal ratio 2 [9].

Our algorithm *Fpm* overcomes the difficulties mentioned above by combining a few new ideas. The first idea is a more sophisticated choice of the target location for the requested item. That is, aside from *full moves* that move the requested items to the list front, *Fpm* sometimes performs a *partial move* to a suitably chosen target location in the list. This location roughly corresponds to the front of the list when this item was requested earlier.

The second idea is to keep track, for each pair of items, of the work function for the two-item subsequence consisting of these items. These work functions are used in two ways. First, they roughly indicate which relative order between the items in each pair is "more likely" in an optimal solution. The algorithm uses this information to decide whether to perform a full move or a partial move. Second, the simple sum of all these pair-based work functions is a lower bound on the optimal cost, which is useful in analyzing the competitive ratio of *Fpm*.

Related work. Better bounds are known for randomized algorithms both in the standard model (LUP_S) and the uniform cost model (LUP_1). For LUP_S, a long line of research culminated in a 1.6-competitive algorithm [19, 24, 2, 6], and a 1.5-lower bound [26]. The upper bound of 1.6 is tight in the class of so-called projective algorithms, whose computation is uniquely determined by their behavior on two-item instances [8]. For LUP_1, the ratio is known to be between 1.5 [4] and 2.64 [24].

It is possible to generalize the uniform cost function by distinguishing between the cost of 1 for following a link during search and the cost of $d \geq 1$ for a swap [24, 4] This model is sometimes called the P^d model; in this terminology, our LUP_1 corresponds to P^1. While *Mtf* is 4-competitive for P^1, it does not generalize in an obvious way to $d > 1$. Other known algorithms for the P^d model (randomized and deterministic *Counter*, *RandomReset* and *TimeStamp*) have bounds on competitive ratios that monotonically decrease with growing d [24, 4]. In particular, deterministic *Counter* achieves ratio 4.56 when d tends to infinity [7] and for the same setting ($d \to \infty$) a recent result by Albers and Janke [4] shows a randomized algorithm *TimeStamp* that is 2.24-competitive.

A variety of List Update variants have been investigated in the literature over the last forty years, including models with lookahead [1], locality of reference [10, 5], parameterized approach [17], algorithms with advice [16], prediction [11], or alternative cost models [18, 22, 23]. A particularly interesting model was proposed recently by Azar *et al.* [12], where an online algorithm is allowed to postpone serving some requests, but is either required to serve them by a specified deadline or pay a delay penalty.

In summary, List Update is one of canonical problems in the area of competitive analysis, used to experiment with refined models of competitive analysis or to study the effects of additional features. This underscores the need to fully resolve the remaining open questions regarding its basic variants, including the question whether ratio 3 is attainable for LUP_1.

2 Preliminaries

Model. An algorithm has to maintain a list of items, while a sequence σ of access requests is presented in an online manner. In each step $t \geq 1$, the algorithm is presented an access request σ^t to an item in the list. If this item is in a location ℓ, the algorithm incurs cost $\ell - 1$ to access it. (The locations in the list are indexed $1, 2, ...$) Afterwards, the algorithm may change the list configuration by performing an arbitrary number of swaps of neighboring items, each of cost 1.

For any algorithm A, we denote its cost for processing a sequence σ by $A(\sigma)$. The optimal algorithm is denoted by Opt.

Notation. Let $\mathcal{P}$ be the set of all unordered pairs of items. For a pair $\{x, y\} \in \mathcal{P}$, we use the notation $x \prec y$ ($x \succ y$) to denote that x is before (after) y in the list of an online algorithm. (The relative order of x, y may change over time, but it will be always clear from context what step of the computation we are referring to.) We use $x \preceq y$ ($x \succeq y$) to denote that $x \prec y$ ($x \succ y$) or $x = y$.

For an input σ and a pair $\{x, y\} \in \mathcal{P}$, σ_{xy} is the subsequence of σ restricted to requests to items x and y only. Whenever we say that an algorithm serves input σ_{xy}, we mean that it has to maintain a list of two items, x and y.

2.1 Work Functions

Work functions on item pairs. For each prefix σ of the input sequence, an online algorithm may compute a so-called *work function* W^{xy}, where $W^{xy}(xy)$ (or $W^{xy}(yx)$) is the optimal cost of the solution that serves σ_{xy} and ends with the list in configuration xy (or yx). (Function W^{xy} also has prefix σ as an argument. Its value will be always uniquely determined from context.) The values of W for each step can be computed iteratively using straightforward dynamic programming. Note that the values of W are non-negative integers and $|W^{xy}(xy) - W^{xy}(yx)| \leq 1$.

Modes. For a pair $\{x, y\} \in \mathcal{P}$, we define its *mode* depending on the value of the work function W^{xy} in the current step and the mutual relation of x and y in the list of an online algorithm. In the following definition we assume that $y \prec x$.

- Pair $\{x, y\}$ is in mode α if $W^{xy}(yx) + 1 = W^{xy}(xy)$.
- Pair $\{x, y\}$ is in mode β if $W^{xy}(yx) = W^{xy}(xy)$.
- Pair $\{x, y\}$ is in mode γ if $W^{xy}(yx) - 1 = W^{xy}(xy)$.

For an illustration of work function evolution and associated modes, see Figure 1.

If a pair $\{x, y\}$ is in mode α, then the minimum of the work function W^{xy} is at configuration yx, i.e., the one that has y before x. That is, an online algorithm keeps these items in a way that "agrees" with the work function. Note that α is the initial mode of all pairs. Conversely, if a pair $\{x, y\}$ is in state γ, then the minimum of the work function W^{xy} is at configuration xy. In this case, an online algorithm keeps these items in a way that "disagrees" with the work function.

2.2 Lower Bound on OPT

Now we show how to use the changes in the work functions of item pairs to provide a useful lower bound on the cost of an optimal algorithm. The following lemma is a standard and straightforward result of the list partitioning technique [9].[2]

[2] There are known input sequences on which the relation of Lemma 1 is not tight [9].

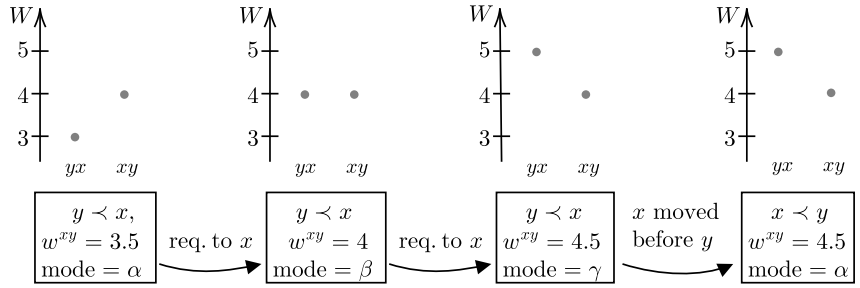


Figure 1 The evolution of the work function W^{xy}. Initially, W^{xy} has its minimum in the state yx and $y \prec x$ (in the list of an online algorithm). Thus, the mode of the pair $\{x, y\}$ is α. Next, because of the two requests to x, the value of $W^{xy}(yx)$ is incremented, while the value at xy remains intact. The mode is thus changed from α to β and then to γ. Finally, when an algorithm moves item x before y, the mode of the pair $\{x, y\}$ changes to α. The value of w^{xy} (the average of $W^{xy}(xy)$ and $W^{xy}(yx)$) increases by $\frac{1}{2}$ whenever the mode changes due to a request.

▶ **Lemma 1.** *For every input sequence σ, it holds that $\sum_{\{x,y\}\in\mathcal{P}} \textsc{Opt}(\sigma_{xy}) \le \textsc{Opt}(\sigma)$.*

Averaging work functions. We define the function w^{xy} as the average value of the work function W^{xy}, i.e.,

$$w^{xy} \triangleq \tfrac{1}{2} \cdot \big(W^{xy}(xy) + W^{xy}(yx) \big).$$

We use w^{xy}_t to denote the value of w^{xy} after serving the first t requests of σ, and define $\Delta_t w^{xy} \triangleq w^{xy}_t - w^{xy}_{t-1}$. The growth of w^{xy} can be related to $\textsc{Opt}(\sigma)$ in the following way.

▶ **Lemma 2.** *For a sequence σ consisting of T requests, it holds that $\sum_{t=1}^{T} \sum_{\{x,y\}\in\mathcal{P}} \Delta_t w^{xy} \le \textsc{Opt}(\sigma)$.*

Proof. We first fix a pair $\{x, y\} \in \mathcal{P}$. We have $w^{xy}_0 = \frac{1}{2}$ and $w^{xy}_T \le \textsc{Opt}(\sigma_{xy}) + \frac{1}{2}$. Hence, $\sum_{t=1}^{T} \Delta_t w^{xy} = w^{xy}_T - w^{xy}_0 \le \textsc{Opt}(\sigma_{xy})$. The proof follows by summing over all pairs $\{x, y\} \in \mathcal{P}$ and invoking Lemma 1. ◀

Pair-based OPT. Lemma 2 gives us a convenient tool to lower bound $\textsc{Opt}(\sigma)$. We define the cost of *pair-based* $\textsc{Opt}$ in step t as $\sum_{\{x,y\}\in\mathcal{P}} \Delta_t w^{xy}$. For a given request sequence σ, the sum of these costs over all steps is a lower bound on the actual value of $\textsc{Opt}(\sigma)$.

On the other hand, we can express $\Delta_t w^{xy}$ (and thus also the pair-based $\textsc{Opt}$) in terms of the changes of the modes of item pairs. See Figure 1 for an illustration.

▶ **Observation 3.** *If a pair $\{x, y\}$ changes its mode due to the request in step t then $\Delta_t w^{xy} = \frac{1}{2}$, otherwise $\Delta_t w^{xy} = 0$.*

3 Algorithm Full-Or-Partial-Move

For each item x, $\textsc{Fpm}$ keeps track of an item denoted θ_x and called the *target* of x. At the beginning, $\textsc{Fpm}$ sets $\theta_x = x$ for all x. Furthermore, at each time, $\textsc{Fpm}$ ensures that $\theta_x \preceq x$.

The rest of this section describes the overall strategy of algorithm $\textsc{Fpm}$. Our description is top-down, and proceeds in three steps:

- First we describe, in broad terms, what actions are involved in serving a request, including the choice of a move and the principle behind updating target items.

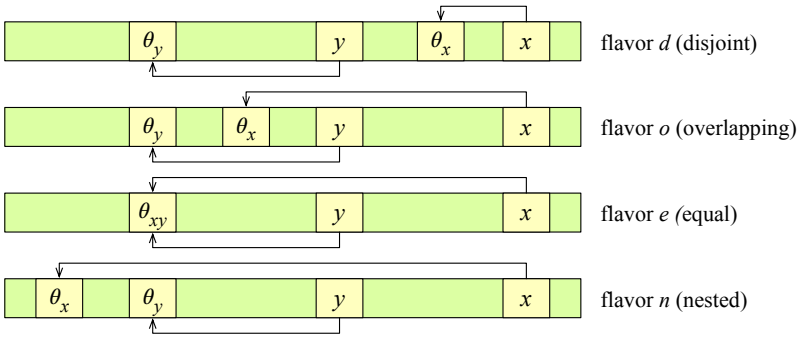


Figure 2 The flavor of a pair $\{x, y\}$ (where $y \prec x$) depends on the position of target θ_x with respect to items $\theta_y \preceq y \prec x$. In the figure for flavor e, we write θ_{xy} for the item $\theta_x = \theta_y$.

- Next, we define the concept of states associated with item pairs and their potentials.
- Finally, we explain how algorithm *FPM* uses these potential values to decide how to adjust the list after serving the request.

This description will fully specify how *FPM* works, providing that the potential function on the states is given. Thus, for any choice of the potential function the competitive ratio of *FPM* is well-defined. What remains is to choose these potential values to optimize the competitive ratio. This is accomplished by the analysis in Section 4 that follows.

Serving a request. Whenever an item z^* is requested, *FPM* performs the following three operations, in this order:

1. *Target cleanup.* If z^* was a target of another item y (i.e., $\theta_y = z^*$ for $y \neq z^*$), then θ_y is updated to the successor of z^*. This happens for all items y with this property.
2. *Movement of z^*.* *FPM* executes one of the two actions: a *partial move* or a *full move*. We will explain how to choose between them later.
   - In the partial move, item z^* is inserted right before θ_{z^*}. (If $\theta_{z^*} = z^*$, this means that z^* does not change its position.)
   - In the full move, item z^* is moved to the front of the list.
3. *Target reset.* θ_{z^*} is set to the front item of the list.

It is illustrative to note a few properties and corner cases of the algorithm.

- Target cleanup is executed only for items following z^*, and thus the successor of z^* exists then (i.e., *FPM* is well defined).
- If $\theta_{z^*} = z^*$ and a partial move is executed, then z^* is not moved, but the items that targeted z^* now target the successor of z^*.
- For an item x, the items that precede θ_x in the list were requested (each at least once) after the last time x had been requested.

Modes, flavors and states. Fix a pair $\{x, y\}$ such that $y \prec x$, and thus also $\theta_y \preceq y \prec x$. This pair is assigned one of four possible *flavors*, depending on the position of θ_x (cf. Figure 2):

- flavor d (**d**isjoint): if $\theta_y \preceq y \prec \theta_x \preceq x$,
- flavor o (**o**verlapping): if $\theta_y \prec \theta_x \preceq y \prec x$,
- flavor e (**e**qual): if $\theta_y = \theta_x \preceq y \prec x$,
- flavor n (**n**ested): if $\theta_x \prec \theta_y \preceq y \prec x$

For a pair $\{x, y\}$ that is in a mode $\xi \in \{\alpha, \beta, \gamma\}$ and has a flavor $\omega \in \{d, o, e, n\}$, we say that the *state* of $\{x, y\}$ is ξ^ω. Recall that all pairs are initially in mode α, and note that their initial flavor is d. That is, the initial state of all pairs is α^d.

We will sometimes combine the flavors o with e, stating that the pair is in state ξ^{oe} if its mode is ξ and its flavor is o or e. Similarly, we will also combine flavors n and e.

Pair potential. To each pair $\{x, y\}$ of items, we assign a non-negative *pair potential* Φ_{xy}. We abuse the notation and use ξ^ω not only to denote the pair state, but also the corresponding values of the pair potential. That is, we assign the potential $\Phi_{xy} = \xi^\omega$ if pair $\{x, y\}$ is in state ξ^ω. We pick the actual values of these potentials only later in Subsection 4.5.

We emphasize that the states of each item pair depend on work functions for this pair that are easily computable in online manner. Thus, FPM may compute the current values of Φ_{xy}, and also compute how their values would change for particular choice of a move.

Choosing the cheaper move. For a step t, let $\Delta_t \text{FPM}$ be the cost of FPM in this step, and for a pair $\{x, y\} \in \mathcal{P}$, let $\Delta_t \Phi_{xy}$ be the change of the potential of pair $\{x, y\}$ in step t. Let z^* denote the requested item. FPM chooses the move (full or partial) with the smaller value of

$$\Delta_t \text{FPM} + \sum_{y \prec z^*} \Delta_t \Phi_{z^* y},$$

breaking ties arbitrarily. Importantly, note that the value that FPM minimizes involves only pairs including the requested item z^* and items currently preceding it.

4 Analysis of Full-Or-Partial-Move

In this section, we show that for a suitable choice of pair potentials, FPM is 3.3904-competitive.

To this end, we first make a few observations concerning how the modes, flavors (and thus states) of item pairs change due to the particular actions of FPM. These are summarized in Table 1, Table 2, and Table 3, and proved in Subsection 4.2 and Subsection 4.3.

Next, in Subsection 4.4 and Subsection 4.5, we show how these changes influence the amortized cost of FPM pertaining to particular pairs. We show that for a suitable choice of pair potentials, we can directly compare the amortized cost of FPM with the cost of OPT.

4.1 Structural Properties

Note that requests to items other than x and y do not affect the mutual relation between x and y. Moreover, to some extent, they preserve the relation between targets θ_x and θ_y, as stated in the following lemma.

▶ **Lemma 4.** *Fix a pair $\{x, y\} \in \mathcal{P}$ and assume that $\theta_y \preceq \theta_x$ (resp. $\theta_y = \theta_x$). If FPM serves a request to an item z^* different than x and y, then these relations are preserved. The relation $\theta_y \prec \theta_x$ changes into $\theta_y = \theta_x$ only when θ_y and θ_x are adjacent and $z^* = \theta_y$.*

Proof. By the definition of FPM, the targets of non-requested items are updated (during the target cleanup) only if they are equal to z^*. In such a case, they are updated to the successor of z^*. We consider several cases.

- If $z^* \notin \{\theta_x, \theta_y\}$, the targets θ_x and θ_y are not updated.
- If $z^* = \theta_y = \theta_x$, both targets are updated to the successor of z^*, and thus they remain equal.

- If $z^* = \theta_y \prec \theta_x$, target θ_y is updated. If θ_y and θ_x were adjacent (θ_y was an immediate predecessor of θ_x), they become equal. In either case, $\theta_y \preceq \theta_x$.
- If $\theta_y \prec \theta_x = z^*$, target θ_x is updated, in which case the relation $\theta_y \prec \theta_x$ is preserved. ◀

4.2 Mode Transitions

Now, we focus on the changes of modes. It is convenient to look first at how they are affected by the request itself (which induces an update of the work function), and subsequently due to the actions of *FPM* (when some items are swapped). The changes are summarized in the following observation.

▶ **Observation 5.** *Let z^* be the requested item.*

- *Fix an item $y \succ z^*$. The mode transitions of the pair $\{z^*, y\}$ due to request are $\alpha \to \alpha$, $\beta \to \alpha$, and $\gamma \to \beta$. Subsequent movement of z^* does not further change the mode.*
- *Fix an item $y \prec z^*$. Due to the request, pair $\{z^*, y\}$ changes first its mode due to the request in the following way: $\alpha \to \beta$, $\beta \to \gamma$, and $\gamma \to \gamma$. Afterwards, if FPM moves z^* before y, the subsequent mode transitions for pair $\{z^*, y\}$ are $\beta \to \beta$ and $\gamma \to \alpha$.*

4.3 State Transitions

Throughout this section, we fix a step and let z^* be the requested item.

We analyze the potential changes for both types of movements. We split our considerations into three cases corresponding to three types of item pairs. The first two types involve z^* as one pair item, where the second item either initially precedes z^* (cf. Lemma 7) or follows z^* (cf. Lemma 8). The third type involves pairs that do not contain z^* at all (cf. Lemma 9).

While we defined 12 possible states (3 modes × 4 flavors), we will show that α^n, γ^d, and γ^o never occur. This clearly holds at the very beginning as all pairs are then in state α^d.

For succinctness, we also combine some of the remaining states, reducing the number of states to the following six: α^d, β^d, α^{oe}, β^o, β^{ne} and γ^{ne}. (For example, a pair is in state α^{oe} if it is in state α^o or α^e.) In the following we analyze the transitions between them. We start with a simple observation.

▶ **Lemma 6.** *Fix an item $y \neq z^*$. If FPM performs a full move, then the resulting flavor of the pair $\{z^*, y\}$ is d.*

Proof. Item z^* is moved to the front of the list, and its target θ_{z^*} is reset to this item, i.e., $z^* = \theta_{z^*}$. After the movement, we have $z^* \prec \theta_y$. This relation follows trivially if z^* is indeed moved. However, it holds also if z^* was already on the first position: even if $\theta_y = z^*$ before the move, θ_y would be updated to the successor of z^* during the target cleanup. The resulting ordering is thus $\theta_{z^*} = z^* \prec \theta_y \preceq y$, i.e., the flavor of the pair becomes d. ◀

▶ **Lemma 7.** *Fix $y \prec z^*$. The state transitions for pair $\{z^*, y\}$ are given in Table 1.*

Proof. First, suppose *FPM* performs a full move. The pair $\{z^*, y\}$ is swapped, which changes its mode according to Observation 5 ($\alpha \to \beta$, $\beta \to \alpha$, $\gamma \to \alpha$). By Lemma 6, the flavor of the pair becomes d. This proves the correctness of the transitions in row three of Table 1.

In the rest of the proof, we analyze the case when *FPM* performs a partial move. We consider three sub-cases depending on the initial flavor of the pair $\{z^*, y\}$. For the analysis of mode changes we will apply Observation 5.

Table 1 State transitions for pairs $\{z^*, y\}$ where $y \prec z^*$ right before the request to z^*.

State before move	α^d	β^d	α^{oe}	β^o	β^{ne}	γ^{ne}
State after partial move	β^{ne}	γ^{ne}	β^d or β^o or β^{ne}	α^{oe}	α^d	α^d
State after full move	β^d	α^d	β^d	α^d	α^d	α^d

Table 2 State transitions for pairs $\{z^*, y\}$ where $z^* \prec y$ right before the request to z^*.

State before move	α^d	β^d	α^{oe}	β^o	β^{ne}	γ^{ne}
State after move	α^d	α^d	α^d	α^d	α^d or α^{oe}	β^d or β^o or β^{ne}

- Before the movement, the flavor of the pair was d, i.e., $\theta_y \preceq y \prec \theta_{z^*} \preceq z^*$.
  The movement of z^* does not swap the pair, i.e., its mode transitions are $\alpha \to \beta$ and $\beta \to \gamma$. As θ_{z^*} is set to the list front, after the movement $\theta_{z^*} \preceq \theta_y \preceq y \prec z^*$, i.e., the flavor of the pair becomes either e or n. This explains the state transitions $\alpha^d \to \beta^{ne}$ and $\beta^d \to \gamma^{ne}$.
- Before the movement, the flavor of the pair was o, i.e., $\theta_y \prec \theta_{z^*} \preceq y \prec z^*$.
  Due to the movement, the pair is swapped, and its mode transitions are $\alpha \to \beta$ and $\beta \to \alpha$. As θ_{z^*} is set to the list front, we have $\theta_{z^*} \preceq \theta_y$. After the movement, $\theta_y \preceq z^* \prec y$, i.e., the flavor of the pair becomes either o or e. This explains the state transitions $\alpha^{oe} \to (\beta^o$ or $\beta^{ne})$ and $\beta^o \to \alpha^{oe}$.
- Before the movement, the flavor of the pair was e or n, i.e., $\theta_{z^*} \preceq \theta_y \preceq y \prec z^*$.
  The movement swaps the pair, and its mode transitions are $\alpha \to \beta$, $\beta \to \alpha$ and $\gamma \to \alpha$. After the movement $z^* \prec \theta_y$, and target θ_{z^*} is set to the list front, which results in $\theta_{z^*} \preceq z^* \prec \theta_y \preceq y$, i.e., the pair flavor becomes d. This explains the state transitions $\alpha^{oe} \to \beta^d$, $\beta^{ne} \to \alpha^d$, and $\gamma^{ne} \to \alpha^d$ ◀

▶ **Lemma 8.** *Fix $y \succ z^*$. The state transitions for pair $\{z^*, y\}$ are given in Table 2.*

Proof. The pair $\{z^*, y\}$ is not swapped due to the request, and thus, by Observation 5, its mode transition is $\alpha \to \alpha$, $\beta \to \alpha$, $\gamma \to \beta$.

If FPM performs a full move, the flavor of the pair becomes d by Lemma 6. This explains the state transitions $\alpha^d \to \alpha^d$, $\beta^d \to \alpha^d$, $\alpha^{oe} \to \alpha^d$, $\beta^o \to \alpha^d$, $\beta^{ne} \to \alpha^d$, and $\gamma^{ne} \to \beta^d$.

In the following, we assume that FPM performs a partial move, and we will identify cases where the resulting pair flavor is different than d. We consider two cases.

- The initial flavor is d, o or e. That is $\theta_{z^*} \preceq z^* \prec y$ and $\theta_{z^*} \preceq \theta_y \preceq y$. During the target cleanup, θ_y may be updated to its successor, but it does not affect these relations, and in particular we still have $\theta_{z^*} \preceq \theta_y$. Thus, when z^* is moved, it gets placed before θ_y. This results in the ordering $\theta_{z^*} \preceq z^* \prec \theta_y \preceq y$, i.e., the resulting flavor is d.
- The initial flavor is n, i.e., $\theta_y \prec \theta_{z^*} \preceq z^* \prec y$. As $\theta_y \neq z^*$, the target θ_y is not updated during the target cleanup. As z^* is moved right before original position of θ_{z^*}, it is placed after θ_y, and the resulting ordering is $\theta_{z^*} \preceq \theta_y \prec z^* \prec y$. That is, the flavor becomes o or e, which explains the state transitions $\beta^{ne} \to \alpha^{oe}$ and $\gamma^{ne} \to (\beta^o$ or $\beta^{ne})$. ◀

▶ **Lemma 9.** *Fix $y \prec x$, such that $x \neq z^*$ and $y \neq z^*$. State transitions for pair $\{x, y\}$ are given in Table 3.*

Table 3 State transitions for pairs $\{x, y\}$ where $x \neq z^*$ and $y \neq z^*$ right before the request to z^*.

State before move	α^d	β^d	α^{oe}	β^o	β^{ne}	γ^{ne}
State after move	α^d	β^d	α^{oe}	β^o or β^{ne}	β^{ne}	γ^{ne}

Proof. The mode of the pair $\{x, y\}$ is not affected by the request to z^*.

The flavor of the pair $\{x, y\}$ depends on mutual relations between x, y, θ_x and θ_y. By Lemma 4, the only possible change is that θ_x and θ_y were different but may become equal: this happens when they were adjacent and the earlier of them is equal to z^*. We consider four cases depending on the initial flavor of the pair.

- The initial flavor was e ($\theta_y = \theta_x \preceq y \prec x$). As θ_y and θ_x are not adjacent, the flavor remains e.
- The initial flavor was d ($\theta_y \preceq y \prec \theta_x \preceq x$). Suppose $\theta_y = z^*$. As $y \neq z^*$, we have $\theta_y = z^* \prec y \prec \theta_x$. That is, θ_x and θ_y are not adjacent, and thus the flavor remains d.
- The initial flavor was o ($\theta_y \prec \theta_x \preceq y \prec x$). In this case it is possible that θ_y and θ_x are adjacent and $z^* = \theta_y$. The flavor may thus change to e or remain o.
- The initial flavor was n ($\theta_x \prec \theta_y \preceq y \prec x$). Similarly to the previous case, it is possible that θ_x and θ_y are adjacent and $z^* = \theta_x$. The flavor may thus change to e or remain n. ◀

4.4 Amortized Analysis

We set $R = \frac{1}{8}(23 + \sqrt{17}) \leq 3.3904$ as our desired competitive ratio.

In the following, we fix a step t in which item z^* is requested. We partition $\mathcal{P}$ into three sets corresponding to the three types of pairs:

- $\mathcal{P}_1^t \triangleq \{\{y, z^*\} : y \prec z^*\}$ (pairs where z^* is the second item, analyzed in Table 1),
- $\mathcal{P}_2^t \triangleq \{\{y, z^*\} : z^* \prec y\}$ (pairs where z^* is the first item, analyzed in Table 2),
- $\mathcal{P}_3^t \triangleq \{\{x, y\} : x \neq z^* \wedge y \neq z^*\}$ (pairs where z^* is not involved, analyzed in Table 3).

For succinctness, wherever it does not lead to ambiguity, we omit subscripts t, i.e., write $\Delta\Phi_{xy}$ and Δw^{xy} instead of $\Delta_t\Phi_{xy}$ and $\Delta_t w^{xy}$. We also omit superscripts t in $\mathcal{P}_1^t$, $\mathcal{P}_2^t$, and $\mathcal{P}_3^t$.

Our goal is to show the following three bounds:

- $\Delta F_{PM} + \sum_{\{x,y\}\in\mathcal{P}_1} \Delta\Phi_{xy} \leq R \cdot \sum_{\{x,y\}\in\mathcal{P}_1} \Delta w^{xy}$,
- $\sum_{\{x,y\}\in\mathcal{P}_2} \Delta\Phi_{xy} \leq R \cdot \sum_{\{x,y\}\in\mathcal{P}_2} \Delta w^{xy}$,
- $\sum_{\{x,y\}\in\mathcal{P}_3} \Delta\Phi_{xy} \leq R \cdot \sum_{\{x,y\}\in\mathcal{P}_3} \Delta w^{xy}$.

Note that the left hand sides of these inequalities correspond to the portions of amortized cost of F_{PM} corresponding to sets $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, while the right hand sides are equal to R times the corresponding portion of the cost of pair-based O_{PT}. Hence, if we can show the above inequalities for every step t, the competitive ratio of R will follow by simply adding them up.

As we show in the sections that follow, these bounds reduce to some constraints involving state potentials α^d, β^d, α^{oe}, β^o, β^{ne} and γ^{ne}. The bounds for $\mathcal{P}_2$ and $\mathcal{P}_3$, while they involve summations over pairs, can be justified by considering individual pairs and the needed constraints are simple inequalities between state potentials, summarized in the following assumption:

▶ **Assumption 10.** *We assume that*

- $\alpha^d = 0$,
- *all constants* β^d, α^{oe}, β^o, β^{ne} *and* γ^{ne} *are non-negative,*
- $\alpha^{oe} \leq \beta^{ne} + \frac{1}{2}R \leq \beta^o + \frac{1}{2}R$,
- $\max\{\beta^d, \beta^o\} \leq \gamma^{ne} + \frac{1}{2}R$.

The bound for $\mathcal{P}_1$ is most critical (not surprisingly, as it corresponds to requesting the second item of a pair). To analyze this bound, the needed constraints, besides the state potentials also need to involve the numbers of pairs that are in these states. This gives rise to a non-linear optimization problem that we need to solve.

4.4.1 Analyzing pairs from set $\mathcal{P}_1$

The proof of the following lemma is deferred to Subsection 4.5.

▶ **Lemma 11.** *There exist parameters α^d, β^d, α^{oe}, β^o, β^{ne} and γ^{ne}, satisfying Assumption 10, such that for any step t, it holds that $\Delta F_{PM} + \sum_{\{x,y\}\in\mathcal{P}_1} \Delta\Phi_{xy} \le R \cdot \sum_{\{x,y\}\in\mathcal{P}_1} \Delta w^{xy}$.*

4.4.2 Analyzing pairs from set $\mathcal{P}_2$

▶ **Lemma 12.** *For any step t, it holds that $\sum_{\{x,y\}\in\mathcal{P}_2} \Delta\Phi_{xy} \le R \cdot \sum_{\{x,y\}\in\mathcal{P}_2} \Delta w^{xy}$.*

Proof. Recall that $\mathcal{P}_2$ contains all pairs $\{z^*, y\}$, such that $z^* \prec y$. Thus, it is sufficient to show that $\Delta\Phi_{z^*y} \le R \cdot \Delta w^{z^*y}$ holds for any such pair $\{z^*, y\}$. The lemma will then follow by summing over all pairs from $\mathcal{P}_2$.

By Assumption 10, we have

$$\alpha^d - \alpha^{oe} \le 0, \tag{1}$$

$$\max\{\alpha^d, \alpha^{oe}\} - \beta^{ne} = \alpha^{oe} - \beta^{ne} \le \tfrac{1}{2}R, \tag{2}$$

$$\max\{\beta^d, \beta^o, \beta^{ne}\} - \gamma^{ne} = \max\{\beta^d, \beta^o\} - \gamma^{ne} \le \tfrac{1}{2}R. \tag{3}$$

We consider two cases depending on the initial mode of the pair $\{z^*, y\}$. In each case, we upper-bound the potential change on the basis of possible state changes of this pair (cf. Table 2).

- The initial mode of $\{z^*, y\}$ is α. By Table 2, this mode remains α, and thus by Observation 3, $\Delta w^{z^*y} = 0$. Then,

$$\begin{aligned} \Delta\Phi_{z^*y} &\le \max\left\{\alpha^d - \alpha^d, \alpha^d - \alpha^{oe}\right\} && \text{(by Table 2)} \\ &= 0 = R \cdot \Delta w^{z^*y}. && \text{(by (1))} \end{aligned}$$

- The initial mode of $\{z^*, y\}$ is β or γ. By Table 2, its mode changes due to the request to z^*, and hence, by Observation 3, $\Delta w^{z^*y} = \frac{1}{2}$. Then,

$$\begin{aligned} \Delta\Phi_{z^*y} &\le \max\{\alpha^d - \alpha^d, \alpha^d - \beta^d, \alpha^d - \alpha^{oe}, \alpha^d - \beta^o, \\ &\qquad \max\{\alpha^d, \alpha^{oe}\} - \beta^{ne}, \\ &\qquad \max\{\beta^d, \beta^o, \beta^{ne}\} - \gamma^{ne}\} && \text{(by Table 2)} \\ &\le \max\{-\beta^d, -\beta^o, \tfrac{1}{2}R, \tfrac{1}{2}R\} && \text{(by } \alpha^d = 0\text{, (1), (2) and (3))} \\ &= \tfrac{1}{2}R = R \cdot \Delta w^{z^*y}. \end{aligned}$$

◀

4.4.3 Analyzing pairs from set $\mathcal{P}_3$

▶ **Lemma 13.** *For any step t, it holds that $\sum_{\{x,y\}\in\mathcal{P}_3} \Delta\Phi_{xy} \le R \cdot \sum_{\{x,y\}\in\mathcal{P}_3} \Delta w^{xy}$.*

Proof. As in the previous lemma, we show that the inequality $\Delta\Phi_{xy} \le R \cdot \Delta w^{xy}$ holds for any pair $\{x, y\} \in \mathcal{P}_3$, i.e., for a pair $\{x, y\}$, such that $x \ne z^*$ and $y \ne z^*$. The lemma will then follow by summing over all pairs $\{x, y\} \in \mathcal{P}_3$.

Possible state transitions of such a pair $\{x, y\}$ are given in Table 3. Hence, such a pair either does not change its state (and then $\Delta\Phi_{xy} = 0$) or it changes it from β^o to β^{ne} (and then $\Delta\Phi_{xy} = \beta^{ne} - \beta^o \le 0$ by Assumption 10). In either case, $\Delta\Phi_{xy} \le 0 \le R \cdot \Delta w^{xy}$. ◀

4.4.4 Proof of R-competitiveness

We now show that the three lemmas above imply that F_{PM} is R-competitive.

▶ **Theorem 14.** *For an appropriate choice of parameters, the competitive ratio of F_{PM} is at most $R = \frac{1}{8}(23 + \sqrt{17}) \leq 3.3904$.*

Proof. Fix any sequence σ consisting of T requests. By summing the guarantees of Lemma 11, Lemma 12, and Lemma 13, we obtain that for any step t, it holds that

$$\Delta_t F_{PM} + \sum_{\{x,y\}\in\mathcal{P}} \Delta_t \Phi_{xy} \leq R \cdot \sum_{\{x,y\}\in\mathcal{P}} \Delta_t w^{xy}.$$

By summing over all steps, observing that the potentials are non-negative and the initial potential is zero (cf. Assumption 10), we immediately obtain that $F_{PM}(\sigma) \leq R \cdot \sum_{t=1}^{T} \sum_{\{x,y\}\in\mathcal{P}} \Delta w^{xy} \leq R \cdot O_{PT}(\sigma)$. The second inequality follows by Lemma 2. ◀

4.5 Proof of Lemma 11

Again, we focus on a single step t, in which the requested item is denoted z^*. We let A^d, B^d, A^{oe}, B^o, B^{ne}, C^{ne} be the number of items y preceding z^* such that pairs $\{z^*, y\}$ have states α^d, β^d, α^{oe}, β^o, β^{ne} and γ^{ne}, respectively. Let

$$V \triangleq [A^d, B^d, A^{oe}, B^o, B^{ne}, C^{ne}].$$

Note that $\|V\|_1$ is the number of items preceding z^*, and thus also the access cost F_{PM} pays for the request. Moreover, $A^d + B^d$ is the number of items that precede θ_{z^*}. We use $\odot$ to denote scalar product (point-wise multiplication) of two vectors.

We define three row vectors G_{OPT}, G_{PM}, and G_{FM}, such that

$$\begin{pmatrix} G_{\mathrm{PM}} \\ G_{\mathrm{FM}} \\ G_{\mathrm{OPT}} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} + \begin{pmatrix} \beta^{ne} & (\gamma^{ne} - \beta^d) & (\max\{\beta^d, \beta^o\} - \alpha^{oe}) & (\alpha^{oe} - \beta^o) & -\beta^{ne} & -\gamma^{ne} \\ \beta^d & -\beta^d & (\beta^d - \alpha^{oe}) & -\beta^o & -\beta^{ne} & -\gamma^{ne} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Expressing costs as vector products. Recall that to prove Lemma 11, we need to relate the $\mathcal{P}_1$ portion of the amortized cost of F_{PM}, i.e., $\Delta F_{PM} + \sum_{\{x,y\}\in\mathcal{P}_1} \Delta\Phi_{xy}$ and the corresponding portion of the cost of pair-based O_{PT}, i.e., $\sum_{\{x,y\}\in\mathcal{P}_1} \Delta w^{xy}$. In the following two lemmas, we show how to express both terms as vector products.

▶ **Lemma 15.** *It holds that $\sum_{y\prec z^*} \Delta w^{z^* y} = G_{\mathrm{OPT}} \odot V$.*

Proof. The right-hand side of the lemma relation is equal to $\frac{1}{2}(A^d + B^d + A^{oe} + B^o + B^{ne})$. By Observation 5, due to request to z^*:

- $A^d + A^{oe}$ pairs change mode from α to β, and
- $B^d + B^o + B^{ne}$ pairs change mode from β to γ.

By Observation 3, each such mode change contributes $\frac{1}{2}$ to the left hand side of the lemma equation. Note that C^{ne} pairs of mode γ do not change their mode due to the request. ◀

▶ **Lemma 16.** *It holds that $\Delta F_{PM} + \sum_{y\prec z^*} \Delta\Phi_{z^* y} = G \odot V$, where*

- *$G = G_{\mathrm{PM}}$ if F_{PM} performs a partial move, and*
- *$G = G_{\mathrm{FM}}$ if F_{PM} performs a full move.*

Proof. First, assume that FPM performs a partial move. By the definitions of A^d, B^d, A^{oe}, B^o, B^{ne}, and C^{ne}, $\Delta FPM = [1,1,2,2,2,2] \odot V$. By Table 1,

$$\begin{aligned}\sum_{y \prec z^*} \Delta\Phi_{z^*y} &\\ &\leq \left[\beta^{ne} - \alpha^d, \gamma^{ne} - \beta^d, \max\{\beta^d, \beta^o, \beta^{ne}\} - \alpha^{oe}, \alpha^{oe} - \beta^o, \alpha^d - \beta^{ne}, \alpha^d - \gamma^{ne}\right] \odot V \\ &= \left[\beta^{ne}, \gamma^{ne} - \beta^d, \max\{\beta^d, \beta^o\} - \alpha^{oe}, \alpha^{oe} - \beta^o, -\beta^{ne}, -\gamma^{ne}\right] \odot V,\end{aligned}$$

where the second equality follows as $\alpha^d = 0$ and $\beta^o \geq \beta^{ne}$ (by Assumption 10). Thus, the lemma holds for a partial move.

Next, assume FPM performs a full move. Then, $\Delta FPM = [2,2,2,2,2,2] \odot V$. By Table 1,

$$\begin{aligned}\sum_{y \prec z^*} \Delta\Phi_{z^*y} &= \left[\beta^d - \alpha^d, \alpha^d - \beta^d, \beta^d - \alpha^{oe}, \alpha^d - \beta^o, \alpha^d - \beta^{ne}, \alpha^d - \gamma^{ne}\right] \odot V \\ &= \left[\beta^d, -\beta^d, \beta^d - \alpha^{oe}, -\beta^o, -\beta^{ne}, -\gamma^{ne}\right] \odot V,\end{aligned}$$

where in the second equality we used $\alpha^d = 0$ (by Assumption 10). Thus, the lemma holds for a full move as well. ◀

Recall that FPM is defined to choose the move that minimizes $\Delta FPM + \sum_{y \prec z^*} \Delta\Phi_{z^*y}$.

▶ **Corollary 17.** *It holds that* $\Delta FPM + \sum_{y \prec z^*} \Delta\Phi_{z^*y} = \min\{G_{\mathrm{PM}} \odot V, G_{\mathrm{FM}} \odot V\}$.

Finding Parameters. We may now prove Lemma 11, restated below for convenience.

▶ **Lemma 11.** *There exist parameters α^d, β^d, α^{oe}, β^o, β^{ne} and γ^{ne}, satisfying Assumption 10, such that for any step t, it holds that* $\Delta FPM + \sum_{\{x,y\} \in \mathcal{P}_1} \Delta\Phi_{xy} \leq R \cdot \sum_{\{x,y\} \in \mathcal{P}_1} \Delta w^{xy}$.

Proof. We choose the following values of the parameters:

$$\begin{aligned}&\alpha^d = 0 && \beta^d = \tfrac{1}{16}(5 + 3\sqrt{17}) \approx 1.086 && \alpha^{oe} = 2 \\ &\beta^o = \tfrac{1}{16}(1 + 7\sqrt{17}) \approx 1.866 && \beta^{ne} = \tfrac{1}{16}(9 - \sqrt{17}) \approx 0.305 && \gamma^{ne} = 2\end{aligned}$$

It is straightforward to verify that these values satisfy the conditions of Assumption 10. We note that relation $\alpha^{oe} \leq \beta^{ne} + \frac{1}{2}R$ holds with equality.

By Corollary 17, $\Delta FPM + \sum_{\{x,y\} \in \mathcal{P}_1} \Delta\Phi_{xy} = \Delta FPM + \sum_{y \prec z^*} \Delta\Phi_{z^*y} = \min\{G_{\mathrm{PM}} \odot V, G_{\mathrm{FM}} \odot V\}$. On the other hand, by Lemma 15, $\sum_{\{x,y\} \in \mathcal{P}_1} \Delta w^{xy} = \sum_{y \prec z^*} \Delta w^{z^*y} = G_{\mathrm{OPT}} \odot V$. Hence, it remains to show that $\min\{G_{\mathrm{PM}} \odot V, G_{\mathrm{FM}} \odot V\} \leq R \cdot G_{\mathrm{OPT}} \odot V$.

We observe that

$$\begin{aligned}G_{\mathrm{PM}} &= \tfrac{1}{16} \cdot \left[25 - \sqrt{17},\ \ 43 - 3\sqrt{17},\ 1 + 7\sqrt{17},\ 63 - 7\sqrt{17},\ 23 + \sqrt{17},\ 0\right], \\ G_{\mathrm{FM}} &= \tfrac{1}{16} \cdot \left[37 + 3\sqrt{17},\ 27 - 3\sqrt{17},\ 5 + 3\sqrt{17},\ 31 - 7\sqrt{17},\ 23 + \sqrt{17},\ 0\right].\end{aligned}$$

Let $c = \frac{1}{4}(\sqrt{17} - 1)$ and let $G_{\mathrm{COMB}} = c \cdot G_{\mathrm{PM}} + (1 - c) \cdot G_{\mathrm{FM}}$. Then,

$$G_{\mathrm{COMB}} = \tfrac{1}{16} \cdot \left[23 + \sqrt{17},\ 23 + \sqrt{17},\ 23 + \sqrt{17},\ 23 + \sqrt{17},\ 23 + \sqrt{17},\ 0\right].$$

Now for any vector V,

$$\begin{aligned}\min\{G_{\mathrm{PM}} \odot V, G_{\mathrm{FM}} \odot V\} &\leq c \cdot G_{\mathrm{PM}} \odot V + (1 - c) \cdot G_{\mathrm{FM}} \odot V \\ &= (c \cdot G_{\mathrm{PM}} + (1 - c) \cdot G_{\mathrm{FM}}) \odot V \\ &= G_{\mathrm{COMB}} \odot V = R \cdot G_{\mathrm{OPT}} \odot V,\end{aligned}$$

completing the proof. ◀

5 Final Remarks

The most intriguing question left open in our work is whether competitive ratio of 3 can be achieved. We have shown computationally (see the full version of the paper [14]) that 3-competitive algorithms exist for lists with up to 6 items.

However, even for short lists the definition of such 3-competitive algorithm remains elusive. For many online problems, the most natural candidate is the generic work function algorithm. This algorithm is 2-competitive in the LUP_S model [9]. However, our computer-aided calculation of its performance shows that its ratio is larger than 3 already for 5 items (see the full version of the paper [14]). It is 3-competitive for lists of length up to 4, though.

We do not know whether the analysis of FPM is tight. For the specific choice of parameters used in the paper, we verified that FPM is 3-competitive for lists of length 3, but not better than 3.04-competitive for lists of length 5 (see the full version of the paper [14]).

The focus of this paper is on the LUP_1 model (also known as P^1); we believe that the setting of $d = 1$ captures the essence and hardness of the deterministic variant. That said, extending the definition and analysis of FPM to the P^d model (for arbitrary d) is an interesting open problem that deserves further investigation.

References

1 Susanne Albers. A competitive analysis of the list update problem with lookahead. *Theoretical Computer Science*, 197(1):95–109, 1998. doi:10.1016/S0304-3975(97)00026-1.

2 Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, 1998. doi:10.1137/S0097539794277858.

3 Susanne Albers. Online list update. In *Encyclopedia of Algorithms*, pages 598–601. Springer, 2008. doi:10.1007/978-0-387-30162-4_266.

4 Susanne Albers and Maximilian Janke. New bounds for randomized list update in the paid exchange model. In *37th Int. Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.STACS.2020.12.

5 Susanne Albers and Sonja Lauer. On list update with locality of reference. *Journal of Computer and System Sciences*, 82(5):627–653, 2016. doi:10.1016/J.JCSS.2015.11.005.

6 Susanne Albers, Bernhard Von Stengel, and Ralph Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*, 56(3):135–139, 1995. doi:10.1016/0020-0190(95)00142-Y.

7 Susanne Albers and Jeffery Westbrook. Self-organizing data structures. In *Online Algorithms, The State of the Art*, pages 13–51. Springer, 1996. doi:10.1007/BFb0029563.

8 Christoph Ambühl, Bernd Gärtner, and Bernhard von Stengel. Optimal lower bounds for projective list update algorithms. *ACM Transactions on Algorithms*, 9(4):1–18, 2013. doi:10.1145/2500120.

9 Eric J. Anderson, Kirsten Hildrum, Anna R. Karlin, April Rasala, and Michael E. Saks. On list update and work function algorithms. In *Proc. 7th Annual European Symposium on Algorithms (ESA)*, pages 289–300. Springer, 1999. doi:10.1007/3-540-48481-7_26.

10 Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. List update with locality of reference. In *Proc. 8th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 399–410. Springer, 2008. doi:10.1007/978-3-540-78773-0_35.

11 Yossi Azar, Shahar Lewkowicz, and Varun Suriyanarayana. List update with prediction. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 15436–15444. AAAI Press, 2025. doi:10.1609/AAAI.V39I15.33694.

12 Yossi Azar, Shahar Lewkowicz, and Danny Vainstein. List update with delays or time windows. In *Proc. 51st Int. Colloqium on Automata, Languages, and Programming (ICALP)*, pages 15:1–15:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.15.

13 Ran Bachrach, Ran El-Yaniv, and M. Reinstadtler. On the competitive theory and practice of online list accessing algorithms. *Algorithmica*, 32(2):201–245, 2002. doi:10.1007/S00453-001-0069-8.

14 Mateusz Basiak, Marcin Bienkowski, Martin Böhm, Marek Chrobak, Łukasz Jeż, Jiří Sgall, and Agnieszka Tatarczuk. A 3.3904-competitive online algorithm for list update with uniform costs. arXiv:2503.17264. Full version.

15 Jon L. Bentley and Catherine C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985. doi:10.1145/3341.3349.

16 Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. On the list update problem with advice. *Information and Computation*, 253:411–423, 2017. doi:10.1016/J.IC.2016.06.007.

17 Reza Dorrigiv, Martin R Ehmsen, and Alejandro López-Ortiz. Parameterized analysis of paging and list update algorithms. *Algorithmica*, 71(2):330–353, 2015. doi:10.1007/S00453-013-9800-5.

18 Alexander Golynski and Alejandro López-Ortiz. Optimal strategies for the list update problem under the MRM alternative cost model. *Information Processing Letters*, 112(6):218–222, 2012. doi:10.1016/J.IPL.2011.12.001.

19 Sandy Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, 1991. doi:10.1016/0020-0190(91)90086-W.

20 Shahin Kamali. Online list update. In *Encyclopedia of Algorithms*, pages 1448–1451. Springer, 2016. doi:10.1007/978-1-4939-2864-4_266.

21 Shahin Kamali and Alejandro López-Ortiz. *A Survey of Algorithms and Models for List Update*, pages 251–266. Springer, 2013. doi:10.1007/978-3-642-40273-9_17.

22 Conrado Martınez and Salvador Roura. On the competitiveness of the move-to-front rule. *Theoretical Computer Science*, 242(1-2):313–325, 2000. doi:10.1016/S0304-3975(98)00264-3.

23 J. Ian Munro. On the competitiveness of linear search. In *Proc. 8th Annual European Symposium on Algorithms (ESA)*, pages 338–345. Springer, 2000. doi:10.1007/3-540-45253-2_31.

24 Nick Reingold, Jeffery Westbrook, and Daniel D Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994. doi:10.1007/BF01294261.

25 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

26 Boris Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47(1):5–9, 1993. doi:10.1016/0020-0190(93)90150-8.

Tolerant Testers for Subgraph-Freeness

Reut Levi
Efi Arazi School of Computer Science, Reichman University, Herzliya, Israel

Jonathan Meiri
Efi Arazi School of Computer Science, Reichman University, Herzliya, Israel

Abstract

In this paper we study the problem of tolerantly testing the property of being H-free (which also implies distance approximation from being H-free).

In the general-graphs model, we show that for tolerant K_k-freeness testing can be achieved with query complexity that is polynomial in the arboricity of the input graph G, $arb(G)$, and independent of the size of G (for graphs in which the average degree is $\Omega(1)$).

Specifically for triangles, our algorithm distinguished graphs which are ϵ-close to being triangle-free from graphs that $3\epsilon(1+\eta)$-far from being triangle-free with expected query complexity which is $\tilde{O}(arb^3(G))$ (for constant η and ϵ).

For general k-cliques our algorithm distinguishes graphs which are ϵ-close to being K_k-free from graphs which are $\binom{k}{2}\epsilon(1+\eta)$-far from being K_k-free with expected query complexity which is polynomial in k, ϵ, γ and $arb(G)$.

We then generalize our result and provide a similar result for any motif H which is 2-connected of radius 1. This includes for example the wheel-graph.

Finally, we show that our tester can be applied to the bounded-degree model for tolerantly testing H-freeness for any motif H. The query complexity of the algorithm is polynomial in the degree bound, d, improving the previous state-of-the-art by Marko and Ron (TALG 2009) that obtained quasi-polynomial query complexity in d.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Tolerant Testing, Property Testing, Subgraph freeness, distance approximation, arboricity

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.77

Funding *Reut Levi*: The author was supported by the Israel Science Foundation under Grant 1867/20.
Jonathan Meiri: The author was supported by the Israel Science Foundation under Grant 1867/20.

1 Introduction

The problem of testing H-freeness, where H is a small *motif* (namely, of size which is independent of the input graph), has received a lot of attention due to its fundamental importance for various fields including biology, sociology and network science. Detecting specific patterns, such as subgraphs, can reveal community structures, vulnerabilities, or anomalous behavior.

In the realm of property testing the decision problem is relaxed such that we are only aiming to distinguish graphs which are H-free from graphs that are *far* from being H-free (according to some predetermined distance measure). However, handling real-world data often means dealing with noise, incompleteness, or small errors. In these scenarios, tolerant testing (introduced by Parnas, Ron and Rubinfeld [15]) for H-freeness becomes crucial, as it allows us to distinguish graphs that are nearly H-free from those that are far from being H-free, thereby bypassing the problem of imperfections in the data. More specifically, a tolerant property testing algorithm is required to distinguish objects that are ϵ_1-close to having a given property $\mathcal{P}$ from objects that are ϵ_2-far from having $\mathcal{P}$, for some parameters


© Reut Levi and Jonathan Meiri;
licensed under Creative Commons License CC-BY 4.0



$0 \leq \epsilon_1 < \epsilon_2 \leq 1$. Clearly, by definition, tolerant-testing is at least as hard as testing. In fact, the separation between tolerant and non-tolerant testing can be very dramatic [8]. Another benefit of tolerant testing is that it implies (by a simple reduction) distance approximation where the ratio of approximation depends on the parameters ϵ_1, ϵ_2 (see more details in [15]). Therefore, whenever feasible, we should aim to develop tolerant testers, ideally without incurring a significant overhead in query complexity.

One of the early works on tolerant testing by Marko and Ron [13] showed that it is possible to tolerantly test H-freeness in the bounded-degree model [9] for any motif H with query complexity that is quasi-polynomial in the degree bound and independent of the size of the graph.

In the regime of non-tolerant testers, Levi [12] showed that it is possible to test triangle-freeness with query complexity that depends linearly in the arboricity of the graph (which is tight for graphs with arboricity $O(\sqrt{n})$) and is independent of the size of the graph (assuming the average degree is $\Omega(1)$). Since graphs of bounded degree have bounded arboricity, this result applies to a broader family of graphs. However, this tester is not tolerant.

One natural question is whether the result in [12] can be extended to show that tolerant testing of triangle-freeness (and ideally other motifs) is possible in the general-graphs model with query complexity that depends only on the arboricity of the graph. Such a result would also generalize [13] (since bounded-degree graphs inherently have bounded arboricity). Another important question is whether the query complexity of subgraph freeness in the bounded-degree model can be improved to be polynomial in the degree bound (rather than quasi-polynomial). In this paper, we answer both questions affirmatively.

1.1 Our Results

1.1.1 The general-graphs model

1.1.1.1 Tolerant Testing K_k-freeness

Our tester for the property of being K_k-free distinguishes graphs that are ϵ-close to being K_k free from graphs that are $\epsilon\binom{k}{2}(1+\eta)$-far from being K_k-free, where $\epsilon \in (0,1]$ and $\eta \in (0,1]$ are parameters. The expected query complexity of the algorithm is $\tilde{O}\left(\frac{k^4\theta^{2k-3}}{\eta^2\epsilon\bar{d}} + k^5\theta^{2k-4}\theta^{\min\{2,k-2\}}\right)$ where $\theta = \Theta(arb(G)/(\eta\epsilon))$ and $\bar{d}$ is the average degree (see Theorem 21).

Specifically, assuming $\bar{d} = \Omega(1)$, for the case of triangles, our algorithm distinguished graphs which are ϵ-close to being triangle-free from graphs that $3\epsilon(1+\eta)$-far from being triangle-free with expected query complexity which is $\tilde{O}(arb^3(G)) \cdot poly(\epsilon^{-1}, \eta^{-1}, k)$.

For k-cliques where $k > 3$, our algorithm distinguishes graphs which are ϵ-close to being K_k-free from graphs which are $\binom{k}{2}\epsilon(1+\eta)$-far from being K_k-free with expected query complexity which is $\tilde{O}(arb^{2k-2}(G)) \cdot poly(\epsilon^{-1}, \eta^{-1}, k)$.

1.1.1.2 Tolerant H-freeness for H which is 2-connected with radius 1

We extend our result and apply our tester for motifs that are 2-connected with radius 1. This includes for example the wheel-graph [1]. Specifically our tester distinguishes graphs that are ϵ-close to being H free from graphs that are $\epsilon|E(H)|(1+\eta)$-far from being H-free. The expected query complexity of the algorithm is $\tilde{O}\left(h^7/\eta^2 + h^3/(\eta\epsilon\bar{d})\right) \cdot \theta^{3h-2}$) where $\theta = \Theta(arb(G)/(\eta\epsilon))$ and $h = |V(H)|$ (see Theorem 28).

[1] Any connected graph with an additional vertex that is incident to all other vertices is 2-connected with radius 1.

Specifically, assuming $\bar{d} = \Omega(1)$, the expected query complexity of our tester is $\tilde{O}(arb^{3h-2}(G)) \cdot poly(\epsilon^{-1}, \eta^{-1}, h)$.

1.1.2 The bounded-degree graphs model

Tolerant H-freeness for any H

We show that our tolerant tester can be applied to the bounded-degree model to test H-freeness for any motif H. Specifically the tester distinguishes graphs that are ϵ-close to being H free from graphs that are $\epsilon|E(H)|(1+\eta)$-far from being H-free. The expected query complexity of the algorithm is $\tilde{O}\left(\left(h^7/\eta^2 + h^3/(\eta\epsilon\bar{d})\right) \cdot d^{3h-2}\right)$ where $h = |V(H)|$ and d is the degree bound.

Specifically, assuming $\bar{d} = \Omega(1)$, the expected query complexity of our tester is $\tilde{O}(d^{3h-2}) \cdot poly(\epsilon^{-1}, \eta^{-1}, h)$.

This improves the result in [13] that has query complexity that is quasi-polynomial in d to polynomial dependence in d.

▶ Remark 1. As noted in [13], the result of Bogdanov, Obata and Trevisan [2] implies a linear lower bound on the query complexity of approximating the distance from being triangle-freeness for some small multiplicative and/or additive error (even in bounded degree graphs). Therefore, we can not hope to obtain tolerant testers of subgraph freeness for all parameters ϵ_1, ϵ_2.

▶ Remark 2. As shown in [3], testing C_4-freeness requires $\Omega(n^{1/4})$ queries. Therefore we can not hope to obtain tolerant testers (in the general-graphs model) for general motifs whose query complexity depends only on the arboricity of the graph.

1.2 Our Algorithm

In this section we describe our algorithm for tolerantly test K_k-freeness. This gives a good perspective of our approach. The first ingredient of our algorithm, which also appears in [12], is to perform the test on a subgraph of G. This subgraph, which we refer to as G_θ is the graph after removing the edges between θ-heavy vertices, where a vertex is θ-heavy if its degree is at least θ where $\theta = \Theta(arb(G)/(\eta\epsilon))$. By doing this modification we might decrease the distance by at most $\Theta(\eta\epsilon)$, so if the graph is sufficiently far from being K_k-free, we will still have enough witnesses of violation. On the positive side, since this subgraph does not have edges in which both endpoints are θ-heavy, any K_k in this graph will include at most a single θ-heavy vertex and therefore can be revealed by exploring the neighborhood of the non-heavy vertices of the clique.

The second ingredient of our algorithm is to approximate the number of copies of K_k in G_θ (it will become clear later why we need to approximate this parameter). This can be done by sampling vertices, v, u.a.r. from G, checking if the degree of v is bounded by θ and if so selecting $k-1$ random indices in $[\theta]$. We then perform $k-1$ neighbor-queries on v, according to the selected indices, and check if the subgraph induced on these neighbors (if such neighbors exist) induces a $k-1$-clique. Thus, our sample space is of size $\Theta(n\theta^{k-1})$ and if the graph is ϵ-far from being K_k-free then we have at least ϵm copies of K_k so we need $\Theta(n\theta^{k-1}/(\epsilon m))$ (which is $O(arb(G)^{k-1}/\epsilon)$ when $\bar{d} = \Omega(1)$) trials to get a good approximation.

Before we describe the last ingredient of our algorithm we shall define the *H-copies graph* of G_θ, which we denote by $(G_\theta)^H$. In this graph the vertex set is the set of H-copies in G_θ and a pair of copies are adjacent iff they share at least one edge. Consider a maximal independent set (MIS), S, of this graph. By definition, the copies of this set are edge-disjoint.

Thus the minimum number of edges we need to remove from G_θ in order to make it H-free, which we denote by $dist_H(G_\theta)$, is at least $|S|$. On the other hand, by the maximality of S it follows that $dist_H(G_\theta) \leq |E(H)||S|$ (since we can remove all H-copies in G_θ by removing all the edges of the copies in S). Therefore if we can get a good approximation of *any* MIS of $(G_\theta)^H$, then we will get a good approximation (up to a $|E(H)|$-factor) to the distance of G_θ from being H-free. This is also true for any MIS that is constructed greedily according to some order on the vertices.

Fortunately, as shown by the ingenious work of Yoshida, Yamamoto and Ito [17], if we pick a random vertex v in a graph and a random ordering on the vertices of the graph then the expected number of recursive calls we need to perform in order to locally simulate the outcome of the greedy MIS algorithm on that graph (according to the random ordering we picked) equals the average degree of the graph plus 1. Therefore, we can get a good approximation to the *average* probability that a random H-copy belongs to a random greedy MIS (which is determined by a random ordering) by executing the local simulation on random H-copies and a random orderings (*both* selected uniformly and independently on each trial). Since the ratio between any MIS in a graph F to the total number of vertices in F is at most $1/(\Delta(F)+1)$, where $\Delta(F)$ denotes the maximum degree in F, we obtain that it suffices to evaluate the outcome of a greedy MIS (each time according to a new random ordering) on $\Theta(\Delta((G_\theta)^H))$ copies selected uniformly at random. This will give us a good approximation to the *average* probability that a random H-copy belongs to a random greedy MIS. When we multiply this by the approximation to the total number of H-copies we previously obtained, this will provide a good approximation to the distance of G from being H-free.

1.2.1 The extension for other motifs

We describe the above mentioned algorithm as a general framework for tolerant testing of H-freeness and then implement (according to the defined interface) the necessary specific procedures for testing K_k-freeness and then provide more general procedures for H-freeness of any motif H which is 2-connected and has radius 1. Finally, we show that the procedures for the latter family of motifs can be used to test H-freeness for any motif H in the bounded degree model.

1.3 Related Work

1.3.1 Tolerant tesing of H-freeness in the bounded degree model

As mentioned-above Marko and Ron [13] studied the problem of tolerantly testing H-freeness in the bounded degree model. They begin by describing a global algorithm that constructs a cover of edges by first performing $O(\log d)$ iterations in which the edges of some of the H-copies are added to the cover by a random process (that can be viewed as a distributed randomized algorithm for constructing a large independent set of copies) and then a single edge is added to the cover from each one of the uncovered copies. Since the global algorithm has a distributed nature they show they can locally simulate its outcome by performing $d^{O(\log d)}$ queries and therefore obtain a good approximation to the size of the cover which gives a good approximation to the distance from being H-free.

1.3.2 Non-tolerant testing of subgraph-freeness

In the bounded degree model Goldreich and Ron [9] observed that it is possible to test triangle freeness with query complexity $O(1/\epsilon)$ in graphs of maximum degree bounded by some constant.

The problem of testing triangle freeness in the general graph model was first studied by Alon, Kaufman, Krivelevich, Ron [1]. The query complexity of their algorithms depends on n and $\bar{d}$, the number of vertices in the graph and the average degree, respectively. They provided sublinear upper bounds for almost the entire range of parameters. Moreover, their upper bounds are at most quadratic in their lower bounds. However, they are tight only when $d_{\max} = O(\bar{d})$ and $\bar{d} \leq \sqrt{n}$, where $d_{\max}$ denotes the maximum degree and $\bar{d}$ denotes the average degree of the graph or when $\bar{d} = \Theta(1)$. Shortly after, Rast [16] and Gugelmann [10] improved the upper bound and lower bound of [1], receptively, for some ranges of the parameters.

More recently, Levi [12] studied the complexity of testing triangle freeness as a function of the arboricity of the graph (in the general graphs model). It was shown that the complexity of testing triangle freeness depends linearly in the arboricity of the graph for graphs in which the arboricity is $O(\sqrt{n})$ and the average degree is $\Omega(1)$.

Even more recently, Eden, Levi and Ron [3] studied the problem of testing C_k-freeness and showed that if the motif is not a k-clique then it is not always possible to provide a tester whose query complexity is independent of the size of the graph. In particular, they showed that testing C_4-freeness requires $\Omega(n^{1/4})$ queries.

1.3.3 Sublinear algorithms that receive the arboricity of the graph as a parameter

Several sublinear-time graph algorithms for counting and sampling give improved results when the graph G has bounded arboricity. This includes the following results. Eden, Ron and Rosenbaum [4] designed an algorithm for sampling edges almost uniformly.

Eden, Ron and Seshadhri [6] estimate the degree distribution moments of an undirected graph and in particular estimate the average degree of a graph.

In another paper, Eden, Ron and Seshadhri [7] give a $(1 \pm \epsilon)$-approximation for the number of k-cliques. In a more recent work, Eden, Ron, and Rosenbaum [5] showed that in this setting sampling cliques is harder than counting.

2 Preliminaries

We consider the general-graphs model [14, 11], where the algorithm can perform the following types of queries:

1. For any $v \in V$, query for v's degree;
2. For any vertex $v \in V$ and index i, query for v's i-th neighbor (if $i > d(v)$, then a special symbol is returned);
3. For any pair of vertices u, v, query whether $\{u, v\} \in E$.

We denote by $\Delta(G)$ the maximum degree of the graph G. We denote by $\delta_H(G)$ the distance of G from being H-free, namely, the fraction of edges that we need to remove from G to make it H-free. We denote by $dist_H(G) = \delta_H(G)|E(G)|$, namely, the minimum number of edges we need to remove from G in order to make is H-free. For a graph F we use $V(F)$ to denote its vertex set and $E(F)$ to denote its edge set.

When we say with high constant probability (w.h.c.p.) we mean we can adjust the high constant probability without changing the asymptotic complexity of the algorithm.

▶ **Definition 3** (Copies of a motif H). *For a graph G and a motif H, we say that a subgraph $\mathcal{H}$ of G is a copy of H in G if $\mathcal{H}$ is isomorphic to H.*

We use $n_H(G)$ to denote the number of H-copies in G.

Given a threshold θ, we call the vertices, v, in G such that $d_G(v) > \theta$, *θ-heavy* and otherwise we call them *θ-light*.

▶ **Definition 4** (The graph G_θ)**.** *For a graph $G = (V, E)$ and a threshold θ, the graph G_θ is the graph whose vertex set is V and its edge set is:*

$E \setminus \{\{u, v\} : u \text{ and } v \text{ are } \theta\text{-heavy}\}$.

▶ **Definition 5.** *The arboricity of a graph G, denoted by $arb(G)$, is the minimum number of forests into which the edges of the graph can be partitioned.*

It is known that for $\theta = arb(G)/\eta$ where $\eta \in (0, 1/2]$, $|E(G_\theta)| \geq (1 - 2\eta)m$ (see, e.g., Claim 4 in [12]).

We assume that the arboricity of the input graph G as well as the number of edges, m, and vertices, n, are known to the algorithm.

▶ Remark 6. If the algorithm is not provided with the arboricity of the graph, then it may run the procedure from [12] that obtains the *effective arboricity* of the graph. More specifically, it was shown in [12] how to obtain a value α^* that with high constant probability satisfies the following: (1) $\alpha^* \leq 2arb(G)$; (2) The number of edges between vertices whose degree is at least $\Theta(\alpha^*/\epsilon)$ is at most ϵm, which is suffices for our needs. Up to polylogarithmic factors in n and polynomial factors in $1/\epsilon$, the query complexity of the procedure is $O(arb(G))$ assuming the average degree is $\Omega(1)$.

2.1 The H-copies graph of G

In order to describe our algorithm we shall use the following definitions.

▶ **Definition 7.** *Given a graph $G = (V, E)$ and a motif H we define the H-copies graph of G, G^H to be the graph whose vertex set are the copies of H in G, $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$ and $e = \{\mathcal{H}_1, \mathcal{H}_2\}$ belongs to the edge set of G^H iff $\mathcal{H}_1$ and $\mathcal{H}_2$ share at least one edge.*

Next, we define the notion of an estimator for the number of H-copies of a graph G. The estimator is given a guessed lower bound on $n_H(G_\theta)$. If the guess is accurate, the estimator produces a reliable estimate; otherwise, it does not significantly overshoot, as detailed in the following definition.

▶ **Definition 8.** *We say that an algorithm is a H-*copies number estimator *for a graph G if it receives as parameters s, a guess of a lower bound on $n_H(G)$, and γ, the approximation parameter, and possibly other parameters for which the following holds: If $s \leq n_H(G)$ then w.h.c.p. the return value of the algorithm is in $(1 \pm \gamma)n_H(G)$. Otherwise, w.h.c.p. it is at most $(1 + \gamma)s$.*

▶ Remark 9. The role of the guess is to provide some (indirect) control to the user of the estimator on the number of attempts to sample a copy from the graph by the estimator (e.g. if the graph is H-free then the estimator might make too many attempts to sample a H-copy with no success).

▶ **Definition 10.** *An algorithm $\mathcal{A}$ is a H-*copies oracle *of a graph $G = (V, E)$ if it supports the following queries.*

1. *On query* Random-copy *with parameters $t \in \mathbb{N}$ and $\delta \in (0, 1]$, it returns a copy of H in G uniformly at random or fails. If the number of copies in G is at least t then it returns a copy w.p. at least $1 - \delta$.*
2. *On query* All-neighbors*$(\mathcal{H})$, where $\mathcal{H}$ is a copy of H in G, it returns all the neighbors of $\mathcal{H}$ in G^H (see Definition 7).*

2.2 Local Simulation of Greedy Maximal IS

Our algorithm uses a local simulation of a greedy MIS algorithm. Algorithm 1 describes this local simulation. For a vertex set V, we let S_V denote the set of all permutations over V.

Algorithm 1 Local-Simulation-Greedy-MIS.

Input: Query access to a graph $G = (V, E)$ and parameters $\pi \in S_V$ and $v \in V$

1. Perform an All-Neighbors query on v and sort its neighbors according to π. Let $v_1, \ldots, v_\ell$ denote its neighbors according to this order.
2. For $i = 1, \ldots, \ell$:
   a. If v_i precedes v in π, recursively call Local-Simulation-Greedy-MIS with parameters π and v_i. If the return value is *YES* then return *NO*.
3. Return *YES*.

Let $R^G_\pi(v)$ denote the number of (recursive) calls to Local-Simulation-Greedy-MIS during the evaluation of Local-Simulation-Greedy-MIS on G, π and v. We shall use the following result from the work of Yoshida, Yamamoto and Ito [17].

▶ **Theorem 11** ([17]). *For any graph $G = (V, E)$ with n vertices and m edges,*

$$\mathbb{E}_{\pi \in S_V, v \in V}[R^G_\pi(v)] \leq 1 + \frac{m}{n} .$$

3 The algorithm for tolerant testing of H-freeness

Before describing our algorithm we first relate the distance from being H-free to the size of a Maximal-Independent-Set of G^H as stated in the following claim. We then use this relation to provide a relation between the expectation of a random variable to the distance from being H-free.

▷ Claim 12. For any maximal independent set of G^H, S, it holds that

$$|S| \leq dist_H(G) \leq |S| \cdot |E(H)|.$$

Proof. Since S is an independent set in G^H, it follows that any pair $\mathcal{H}_1, \mathcal{H}_\in \in S$ is a pair of edge-disjoint copies of H in G (by definition). Since S is maximal, it follows that if we remove for each $\mathcal{H} \in S$ all its edges from G then G becomes H-free (since we remove at least one edge from each copy of H in G). ◁

▷ Claim 13. For v that is drawn u.a.r. from $V(G^H)$ and π which is drawn u.a.r. from S_V it follows that:

$$\Pr[v \in MIS_\pi(G^H)] \in \left[\frac{dist_H(G)}{|E(H)|n_H(G)}, \frac{dist_H(G)}{n_H(G)}\right]. \tag{1}$$

Moreover,

$$\Pr[v \in MIS_\pi(G^H)] \geq 1/(\Delta(G^H) + 1). \tag{2}$$

Proof. By Claim 12, for any maximal IS in G^H, S, it holds that $\frac{dist_H(G)}{|E(H)|} \leq |S| \leq dist_H(G)$. Therefore, for any such S it holds that $\Pr[v \in S] \in \left[\frac{dist_H(G)}{|E(H)|n_H}, \frac{dist_H(G)}{n_H}\right]$, where v is drawn u.a.r. from $V(G_H)$. This is true since $|V(G_H)| = n_H$. Since $MIS_\pi(G^H)$ is maximal for any

π (by definition), the claim follows. Equation 2 follows from the fact that each vertex v in the independent set covers at most $d(v)+1$ vertices from the graph. Thus, any maximal independent set is of size at least $n_H(G)/(\Delta(G^H)+1)$, in particular, $MIS_\pi(G^H)$ (for any π). This concludes the proof. ◁

By using the previous claims we arrive to the following claim.

▷ Claim 14. If $s \geq n_H(G)$ then with high constant probability, the return value of Approximate-Average-MIS-In-Auxiliary-Graph is in

$$\left[(1-\gamma)\frac{dist_H(G)}{|E(H)|n_H(G)}, (1+\gamma)\frac{dist_H(G)}{n_H(G)}\right]. \tag{3}$$

Proof. Let E_1 denote the event that in all the executions of Step 2a of the algorithm, a copy of H was returned. By definition 10, the setting of δ and since $s \geq n_H(G)$, w.h.c.p., E_1 occurs. We henceforth assume E_1 occurs. Consider the random variable Z_i (see Step 2c of the algorithm). By construction Z_i takes values in $\{0,1\}$. By Claim 13, Equation 1, $\mathbb{E}(Z_i) \in \left[\frac{dist_H(G)}{|E(H)|n_H(G)}, \frac{dist_H(G)}{n_H(G)}\right]$. By Equation 2, $\mathbb{E}(Z_i) \geq 1/(\Delta(G^H)+1)$. The claim follows from the setting of ℓ and the multiplicative Chernoff's bound (see Theorem 30). ◁

Algorithm 2 Approximate-Average-MIS-In-Auxiliary-Graph.

Input: γ - approximation parameter, s - guess on a lower bound on $n_H(G)$, Δ - an upper bound on $\Delta(G^H)$, and access to H-copy oracle and H-copies estimator of G (see Definitions 8, 10)

1. Set $\ell = \Theta(\Delta/\gamma^2)$ and $\delta = \Theta(1/\ell)$.
2. For $i = 1, \ldots, \ell$
   a. Run a H-copy oracle of G with parameters s and δ, and obtain a H-copy, $\mathcal{H}$, uniformly at random.
   b. Draw π u.a.r. and run Algorithm 1 on $\mathcal{H}$, π and the H-copies oracle of G, (see Definition 7).
   c. Set $Z_i = 1$ if the algorithm returned YES and set $Z_i = 0$ otherwise.
3. Return $\hat{Z} = \sum_{i\in[\ell]} Z_i/\ell$.

Algorithm 3 Tolerant-Test-Subgraph-freeness.

Input: ϵ, η and query access to a graph G

1. Set $\gamma = \eta/64$ and $\theta = arb(G)/(2\gamma\epsilon)$
2. Run the number of H-copies estimator for the graph G_θ with parameters $\epsilon m/2$ and γ (see Definition 8) and obtain $\hat{n}$.
3. If $\hat{n}$ is less than ϵm then return ACCEPT.
4. Run Approximate-Average-MIS-In-Auxiliary-Graph on the graph G_θ with parameters $\epsilon m/2$ and γ and $\Delta((G_\theta)^H$. Let $\hat{Z}$ denote the returned value.
5. if $\hat{n}\hat{Z} > \epsilon m(1+\gamma)^2$ then return REJECT.

We next prove our main theorem for this section. In order to realize this theorem we provide in the next sections concrete H-copies oracles and H-copies number estimators (first for H which is a k-clique and then generalize, for H which is 2-connected and has radius 1).

▶ **Theorem 15.** *Given parameters $\epsilon \in (0,1]$, $\eta \in (0,1]$ and query access to a graph G,* Tolerant-Test-Subgraph-freeness *has the following guarantees. If G is ϵ-close to being H-free then w.h.c.p. it accepts G. If G is $\epsilon|E(H)|(1+\eta)$-far from being H-free then w.h.c.p. it rejects G.*

Proof. Assume G is ϵ-close to being H-free. If the algorithm accepts G in Step 3 then we are done. Otherwise, it follows that $\hat{n} \geq \epsilon m$.

Let E_1 denote the event that the estimator in Step 2 returns a good approximation as described in Definition 8. Conditioned on E_1 and the fact that $\hat{n} \geq \epsilon m$ it is implied that $n_H(G_\theta) \geq \epsilon m/2$ (assume otherwise and reach a contradiction to the fact that $\hat{n} \geq \epsilon m$ as $(1+\gamma) < 2$)) and hence its return value is in $(1 \pm \gamma) n_H(G_\theta)$.

Let E_2 denote the event that Approximate-Average-MIS-In-Auxiliary-Graph returns a good approximation as described in Claim 14. Conditioned on $E_1 \cap E_2$ we obtain that

$$\hat{n}\hat{Z} \in \left[(1-\gamma)^2 \frac{dist_H(G_\theta)}{|E(H)|}, (1+\gamma)^2 dist_H(G_\theta)\right]. \tag{4}$$

Since $dist_H(G_\theta) \leq dist_H(G) \leq \epsilon m$ we obtain that $\hat{n}\hat{Z} \leq (1+\gamma)^2 \epsilon m$ and hence the algorithm accepts G.

Assume G is β-far from being H-free where $\beta = \epsilon|E(H)|(1+\eta)$. By the setting of θ it follows that $dist_H(G_\theta) \geq (1-\gamma)\beta m$. Thus, $n_H(G_\theta) \geq (1-\gamma)\beta m$. Therefore, conditioned on $E_1 \cap E_3$ we obtain that $\hat{n} \in (1 \pm \gamma) n_H(G_\theta)$. In particular, it holds that $n_H(G_\theta) \geq \epsilon m/2$. Hence conditioned on $E_1 \cap E_2 \cap E_3$ we obtain that Equation 4 holds in this case as well. In particular $\hat{n}\hat{Z} \geq (1-\gamma)^3 \frac{\beta m}{|E(H)|} > \epsilon m (1+\gamma)^2$ (since $(1+\eta) > (1+2\gamma)^5 > (1+\gamma)^2/(1-\gamma)^3$). Thus, conditioned on $E_1 \cap E_2 \cap E_3$, the algorithm rejects G, as desired. ◀

▷ Claim 16. The expected query complexity of Tolerant-Test-Subgraph-freeness is bounded above by the query complexity of performing $\ell = \Theta(\Delta/\eta^2)$ random-copy queries to $(G_\theta)^H$ with parameters $s = \epsilon m/2$ and $\delta = \Theta(\eta^2/\Delta)$ (see Definition 7), $\Theta(\ell \cdot \Delta)$ all-neighbor queries to $(G_\theta)^H$, where $\Delta = \Delta((G_\theta)^H)$, and a single execution of a H-copies number estimator for G_θ (see Definition 8).

Proof. By construction, linearity of expectation and Theorem 11. ◁

4 The oracles for k-cliques

In this section we describe our K_k-copies oracle and K_k-copies number estimator. We conclude with our main theorem about tolerant testing of K_k-freeness (Theorem 21).

▷ Claim 17. For every iteration $j \in [t]$ of Algorithm 4 it holds that:

$$\mathbb{E}(Y_j) = n_{K_k}(G_\theta) / \left(n \cdot \binom{\theta}{k-1}\right). \tag{5}$$

Proof. Consider a copy, $\mathcal{K}$, of K_k in G_θ. Let $y(\mathcal{K})$ denote the number of θ-light vertices in $\mathcal{K}$. The probability that $\mathcal{K}$ is found in Step 2f of Approximate-K_k-copies-in-G_θ is exactly $y(\mathcal{K}) \cdot 1/(n\binom{\theta}{k-1})$. Conditioned on the event that indeed $\mathcal{K}$ was found in Step 2f of iteration j the probability that Y_j is set to be 1 is $1/y(\mathcal{K})$. Thus, for every j, $\Pr(Y_j = 1) = n_{K_k}(G_\Theta)/(n\binom{\theta}{k-1})$. The claim follows. ◁

▷ Claim 18. Approximate-K_k-copies-in-G_θ is a K_k-copies number estimator of the graph G_θ. Its query complexity is $O(k^2 n\binom{\theta}{k-1}/(\gamma^2 s))$.

Algorithm 4 Approximate-K_k-copies-in-G_θ .

Input: $\theta, s, \gamma \in (0, 1/2]$

1. Set $t = \Theta(n\binom{\theta}{k-1}/(\gamma^2 s))$
2. For $j = 1, \ldots, t$
   a. Sample a vertex $v \in V$ u.a.r. and a set of $k-1$ indexes, $i_1, \ldots, i_{k-1}$ u.a.r. from $\binom{[\theta]}{k-1}$.
   b. If v θ-heavy then set $Y_j = 0$ and go to the next iteration.
   c. Otherwise, query (v, i_ℓ) for each $\ell \in [k-1]$. Let $u_1, \ldots, u_{k-1}$ denote the respective return values.
   d. Let y denote the number of θ-light vertices in $\{u_i\}_{i\in[\ell]}$.
   e. If $y < k-2$ then set $Y_j = 0$ (there is more than a single θ-heavy vertex in the set).
   f. If the subgraph induced on $\{u_i\}_{i\in[\ell]}$ is a $(k-1)$-clique then set $Y_j = 1$ w.p. $1/(y+1)$. Otherwise set $Y_j = 0$.
3. Return $n \cdot \binom{\theta}{k-1} \cdot \sum_{j\in[t]} Y_j/t$.

Proof. For $s \leq n_{K_k}(G_\theta)$ the claim follows from Claim 16, the setting of t and multiplicative Chernoff's bound (see Theorem 30). To see the correctness of the second part, consider $s = n_{K_k}(G_\theta)$. By the first part of the claim, w.h.c.p. the return value of the algorithm is in $(1 \pm \gamma)n_{K_k}(G_\theta) = (1 \pm \gamma)s$. Namely, at most $(1+\gamma)s$. Now consider the case in which $s > n_{K_k}$. By a coupling argument, w.h.c.p. the return value is at most $(1+\gamma)s$ in this case. To see this, couple the return value of the algorithm to the return value of the algorithm on the graph after we add $s - n_{K_k}$ copies to the graph (arbitrarily). Since we added copies, the return value of the algorithm on the modified graph dominates the return value on the original graph (to see this consider executing the algorithm in parallel in both graphs - the outcome in the modified graph is always at least as high) but is guaranteed to be at most $(1+\gamma)s$ w.h.c.p. by the above. The claim about the query complexity follows from construction. This concludes the proof. ◁

Algorithm 5 Get-Random-K_k-Copy-in-G_θ .

Input: θ, s, δ

1. Repeat $\Theta(n\binom{\theta}{k-1}\log(1/\delta)/s)$ times:
   a. Sample a vertex $v \in V$ u.a.r. and a set of $k-1$ indexes, $i_1, \ldots, i_{k-1}$ u.a.r. from $\binom{[\theta]}{k-1}$.
   b. If v is not θ-light then break.
   c. Otherwise, query (v, i_ℓ) for each $\ell \in [k-1]$. Let $u_1, \ldots, u_{k-1}$ denote the respective return values.
   d. Let y denote the number of θ-light vertices in $\{u_i\}_{i\in[\ell]}$.
   e. If $y < k-2$ or if the subgraph induced on $\{u_i\}_{i\in[\ell]}$ is not a clique, then go to the next iteration.
   f. Otherwise, w.p. $1/(y+1)$, return the k-clique induced on v and $\{u_i\}_{i\in[\ell]}$.

▷ Claim 19. The query complexity of Get-Random-K_k-Copy-in-G_θ is $\Theta(k^2 n\binom{\theta}{k-1}\log(1/\delta)/s)$. If $s < n_{K_k}(G_\theta)$ then it returns a copy w.p. at least $1-\delta$.

Proof. For any copy of K_k, $\mathcal{K}$, in each iteration of the while-loop, the probability that we return $\mathcal{K}$ in Step 1f of the algorithm is exactly $1/(n\binom{\theta}{k-1})$. Therefore if the number of copies is greater than s then the probability that it will return a copy during in a single iteration is at least $s/(n\binom{\theta}{k-1})$. The claim follows by the setting of the number of iterations. ◁

Algorithm 6 K_k-All-Neighbors-in-G_θ .

Input: θ, $\mathcal{K}$

1. Set $C = \emptyset$
2. If $k = 3$, reveal the neighbors of all the θ-light vertices of $\mathcal{K}$. Let S denote the set of revealed neighbors. If there is a θ-heavy vertex, u, in $\mathcal{K}$, perform a pair query between u and each one of the vertices in S.
3. Otherwise, for each θ-light vertex, v, of $\mathcal{K}$: Reveal all the neighbors of v in G and then for each θ-light neighbor, reveal all its neighbors.
4. Add all the K_k-copies that were revealed in the previous steps that belong to G_θ (namely, that have at most one θ-heavy vertex) and share an edge with $\mathcal{K}$.
5. Return C.

▷ Claim 20. The query complexity of K_k-All-Neighbors-in-G_θ is $O(k\theta)$ for $k = 3$ and $O(k\theta^2)$ for $k > 3$.

Proof. The claim follows from construction. ◁

▶ **Theorem 21.** *There exists a tolerant tester for the property of being K_k-free that distinguishes graphs that are ϵ-close to being K_k free from graphs that are $\epsilon\binom{k}{2}(1+\eta)$-far from being K_k-free, where $\epsilon \in (0,1]$ and $\eta \in (0,1]$ are parameters. The expected query complexity of the algorithm is $\tilde{O}\left(\frac{k^4\theta^{2k-3}}{\eta^2\epsilon\bar{d}} + k^5\theta^{2k-4}\theta^{\min\{2,k-2\}}\right)$ where $\theta = \Theta(arb(G)/(\eta\epsilon))$.*

Proof. The theorem follows from the fact that $\Delta((G_\theta)^H) \leq \binom{k}{2}\theta^{k-2}$, Theorem 15 and claims 16-20. ◀

5 The oracles for 2-connected motifs of radius 1

In this section, H is a motif which is 2-connected and has radius 1. We make the following observation on the copies of H in G_θ.

▶ **Observation 22.** *For any copy of H, $\mathcal{H}$, in G_θ, at least one of the following holds:*

1. *there exists a θ-light vertex in $\mathcal{H}$ which is incident to all the vertices in $\mathcal{H}$.*
2. *there is exactly one θ-heavy vertex in $\mathcal{H}$.*

Proof. Let v be a vertex in $\mathcal{H}$ which is incident to all other vertices in $\mathcal{H}$ (such vertex exists since H has radius 1). If v is θ-light then Item 1 holds. Otherwise, v is θ-heavy. Since there are no edges in G_θ such that both endpoints are θ-heavy, the claim follows. ◀

We say that a BFS is a *θ-restricted* if it explores only the neighbors of the θ-light vertices it reaches. We say that an exploration *reveals a copy* $\mathcal{H}$ if all the vertices of $\mathcal{H}$ appeared in the exploration.

▷ Claim 23. Let H be a motif that is 2-connected and has radius 1. In every copy of H in G_θ, $\mathcal{H}$, for every θ-light vertex v in $\mathcal{H}$, a θ-restricted BFS from v of depth $|V(H)|$ reveals $\mathcal{H}$.

Proof. If item 1 of observation 22 holds, then clearly the claim follows. Otherwise, there is exactly one θ-heavy vertex, u, in $\mathcal{H}$. Since H is 2-connected, after we remove u from $\mathcal{H}$, $\mathcal{H}$ remains connected and its diameter is at most $|V(H)| - 1$. Thus, a θ-restricted BFS of depth $|V(H)| - 1$ from any θ-light vertex of $\mathcal{H}$ reveals all the θ-light vertices of $\mathcal{H}$. The claim follows from the fact that at least one of the θ-light vertices in $\mathcal{H}$ is a neighbor of u. ◁

Therefore, by Claim 23, finding for a θ-light vertex, v, all the H-copies in G_θ that include v, can be done by using $O(\theta^{|V(H)|})$ queries. Consequently, a query for all-neighbors in $(G_\theta)^H$ can be answered by making $O(|V(H)|\theta^{|V(H)|})$ queries to G. This can be achieved by going over all vertices of the queried copy, $\mathcal{H}$, and for each θ-light vertex, finding all its copies. From the union of these sets we then remove all the copies that do not share an edge with $\mathcal{H}$ and obtain all the neighbors of $\mathcal{H}$ in $(G_\theta)^H$.

To support random copy queries we assign each copy of H to its θ-light vertex of lowest id. To sample a H-copy u.a.r. from G_θ we first pick a vertex $v \in V$ u.a.r., if it is θ-light then we find all its H-copies as described above. We then pick a copy that is assigned to v u.a.r. and return it w.p. $a_{H,\theta}(v)/a^{max}_{H,\theta}$ where $a_{H,\theta}(v)$ denotes the number of copies that are assigned to v in G_θ and $a^{max}_{H,\theta}$ denotes the maximum number of copies that include a specific θ-light vertex (recall that the copies in G_θ are assigned only to θ-light vertices). Clearly, the above algorithm samples each copy of H in G_θ w.p. $1/(n \cdot a^{max}_{H,\theta})$. Consequently, the probability that it returns a copy is $n_H(G_\theta)/(n \cdot a^{max}_{H,\theta})$. Thus, the success probability of the algorithm depends on $a^{max}_{H,\theta}$). A straightforward upper-bound on $a^{max}_{H,\theta}$ is $|V(H)|!\binom{\theta^{|V(H)|}}{|V(H)|}$ where the first term is due to all possible labeling of the vertices and the second term is due to all possible ways to pick the vertices of the copy from the set of vertices we explored. In the next claim we provide a tighter bound on $a^{max}_{H,\theta}$.

▷ Claim 24. For any motif H, it holds that $a^{max}_{H,\theta} \leq |V(H)| \cdot \theta^{|V(H)|-1}$.

Proof. Fix an arbitrary labeling of H. Let $v \in V$. We next bound $a_{H,\theta}(v)$. Label the H-copies that v belongs to according to the fixed labeling of H (each copy has its own labeling). If there is more than one such labeling per copy (due to automorphism), pick one arbitrarily. We next describe a one-to-one mapping from the set of labeled copies to a set of sequences of integers. We then bound the cardinality of the latter set and obtain a bound on the number of H-copies that v belongs to and hence a bound on $a^{max}_{H,\theta}$. Fix a labeled copy of v, $\mathcal{H}$, and consider the BFS exploration of $\mathcal{H}$ starting at v, where vertices with smaller label are explored first. Given the label of v and the sequence of the vertices in $\mathcal{H}$ by their BFS order we can reconstruct the labels of all the vertices in the copy. The set of vertices and their labels are mapped to a single copy. Moreover, by the above, we can identify the copy by the identity of the root, its label and a sequence of $|V(H) - 1|$ indexes in $[\theta]$. Each index indicates the index of the respective element in its parent adjacency-list. The indexes are listed according to the BFS exploration. Since the identity of the root is known, we can inductively reconstruct the tree by the sequence of indexes. Note that the parents are always θ-light (while the leaves may be θ-heavy) therefore it suffices to use indexes in $[\theta]$. From the above we see that for a specific label of v there are at most $\theta^{|V(H)|-1}$ copies of H to which v belongs. Since there are $|V(H)|$ possible labels, we get the desired bound. ◁

▷ Claim 25. Approximate-H-copies-in-G_θ is a H-copies number estimator of the graph G_θ. Its query complexity is $O(n \cdot a^{max}_{H,\theta} \cdot \theta^{|V(H)|}/(\gamma^2 s))$.

Proof. The claim about the query complexity follows by construction. Consider the random variable Y_j which is defined in Approximate-H-copies-in-G_θ . Since $\mathbb{E}(Y_j) \geq n_H(G_\theta)/n$ and $Y_j \leq a^{max}_{H,\theta}$, for $s \leq n_H(G_\theta)$, by the setting of t and the multiplicative Chernoff's bound (Theorem 30), w.h.c.p the output of the algorithm is in $(1 \pm \gamma)n_H(G)$, as required. For $s > n_H(G_\theta)$, by a coupling argument, w.h.c.p. the output of the algorithm is at most $(1 + \gamma)$ (see more details in the proof of Claim 14). ◁

Algorithm 7 Approximate-H-copies-in-G_θ .

Input: $s, \gamma \in (0, 1/2], \theta$

1. Set $t = \Theta(n \cdot a^{max}_{H,\theta}/(\gamma^2 s))$
2. For $j = 1, \ldots, t$
   a. Sample a vertex $v \in V$ u.a.r.
   b. If v is not θ-light then set $Y_j = 0$ and go to the next iteration.
   c. Otherwise, perform a θ-restricted BFS from v to depth $|V(H)|$ and reveal all the H-copies assigned to v in G_θ. Set Y_j to be this number.
3. Return $n \cdot \sum_{j\in[t]} Y_j/t$.

Algorithm 8 Get-Random-H-Copy-in-G_θ .

Input: θ, s, δ

1. Repeat $\Theta(n \cdot a^{max}_{H,\theta} \log(1/\delta)/s)$ times:
   a. Sample a vertex $v \in V$ u.a.r.
   b. If v is θ-heavy then go to the next iteration.
   c. Otherwise, perform a θ-restricted BFS from v to depth $|V(H)|$ and reveal all the H-copies assigned to v in G_θ.
   d. Pick a random copy from the H-copies assigned to v and return it with probability $a_{H,\theta}(v)/a^{max}_{H,\theta}$.

▷ Claim 26. The query complexity of Get-Random-H-Copy-in-G_θ is $\Theta(\theta^{|V(H)|} \cdot n \cdot a^{max}_{H,\theta} \log(1/\delta)/s)$. It returns a H-copy w.p. at least $1 - \delta$.

Proof. The claim about the query complexity follows by construction. In each iteration of the algorithm, each copy of H is returned with probability $1/(n \cdot a^{max}_{H,\theta})$. Thus, in each iteration a copy is returned w.p. $n_H(G_\theta)/(n \cdot a^{max}_{H,\theta})$. Thus, if $s \leq n_H(G)$, a copy is returned w.p. at least $1 - \delta$ by the setting of the number of iterations. ◁

Algorithm 9 H-All-Neighbors-in-G_θ .

Input: θ

1. Set $C = \emptyset$
2. For each vertex v of $\mathcal{H}$:
   a. Reveal all the H-copies of v in G^θ by performing a θ-restricted BFS to depth $|V(H)|$ from v in G.
   b. Add to C all the revealed H-copies that share an edge with $\mathcal{H}$.
3. Return C.

▷ Claim 27. The query complexity of K_k-All-Neighbors-in-G_θ is $O(|V(H)|\theta^{|V(H)|})$.

Proof. By Construction. ◁

▶ **Theorem 28.** *There exists a tolerant tester for the property of being H-free for H that is 2-connected and has radius 1 that distinguishes graphs that are ϵ-close to being H free from graphs that are $\epsilon|E(H)|(1+\eta)$-far from being H-free, where $\epsilon \in (0, 1]$ and $\eta \in (0, 1]$ are parameters. The expected query complexity of the algorithm is $\tilde{O}\left(\left(h^7/\eta^2 + h^3/(\eta\epsilon\bar{d})\right) \cdot \theta^{3h-2}\right)$ where $\theta = \Theta(arb(G)/(\eta\epsilon))$ and $h = |V(H)|$.*

Proof. The theorem follows from the fact that $\Delta((G_\theta)^H) \leq h \cdot a^{max}_{H,\theta}$, Theorem 15 and claims 24–27. ◀

6 The tester for the bounded-degree model

Clearly, Tolerant-Test-Subgraph-freeness applies also to bounded degree graphs. Moreover, for the setting where $\theta = d + 1$ the graph G_θ is simply G and all the vertices in the graph are θ-light. Consequently, we can test H-freeness for *any* motif H by using the oracles in Section 5 without making any modifications [2]. We obtain the following result.

▶ **Theorem 29.** *There exists a tolerant tester, in the bounded degree model, for the property of being H-free for any motif H. Specifically the tester distinguishes graphs that are ϵ-close to being H free from graphs that are $\epsilon|E(H)|(1+\eta)$-far from being H-free, where $\epsilon \in (0,1]$ and $\eta \in (0,1]$ are parameters. The expected query complexity of the algorithm is $\tilde{O}\left(\left(h^7/\eta^2 + h^3/(\eta\epsilon\bar{d})\right) \cdot d^{3h-2}\right)$ where $h = |V(H)|$ and d is the degree bound.*

Proof. By the proof of Theorem 21. ◀

References

1 Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM Journal on Discrete Mathematics*, 22(2):786–819, 2008. doi:10.1137/07067917X.

2 Andrej Bogdanov, Kenji Obata, and Luca Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 93–102. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181886.

3 Talya Eden, Reut Levi, and Dana Ron. Testing c_k-freeness in bounded-arboricity graphs. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 60:1–60:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.60.

4 Talya Eden, Dana Ron, and Will Rosenbaum. The arboricity captures the complexity of sampling edges. In *46th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 52:1–52:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.52.

5 Talya Eden, Dana Ron, and Will Rosenbaum. Almost optimal bounds for sublinear-time sampling of k-cliques in bounded arboricity graphs. In *49th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 56:1–56:19, 2022. doi:10.4230/LIPIcs.ICALP.2022.56.

6 Talya Eden, Dana Ron, and C. Seshadhri. Sublinear time estimation of degree distribution moments: The arboricity connection. *SIAM J. Discret. Math.*, 33(4):2267–2285, 2019. doi:10.1137/17M1159014.

7 Talya Eden, Dana Ron, and C. Seshadhri. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1467–1478, 2020. doi:10.1137/1.9781611975994.89.

8 Eldar Fischer and Lance Fortnow. Tolerant versus intolerant testing for boolean properties. *Theory Comput.*, 2(9):173–183, 2006. doi:10.4086/TOC.2006.V002A009.

9 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. doi:10.1007/S00453-001-0078-7.

[2] In fact the query complexity of Algorithms 7- 9 can be slightly improved by setting the depth of the BFS exploration to be the diameter of H (instead of $|V(H)|$).

10 L. Gugelmann. Testing triangle-freeness in general graphs: Lower bounds. Bachelor thesis, Dept. of Mathematics, ETH, Zurich, 2006.

11 Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on Computing*, 33(6):1441–1483, 2004. doi:10.1137/S0097539703436424.

12 Reut Levi. Testing triangle freeness in the general model in graphs with arboricity $o(\sqrt{n})$. In *48th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 198, pages 93:1–93:13, 2021. doi:10.4230/LIPICS.ICALP.2021.93.

13 Sharon Marko and Dana Ron. Approximating the distance to properties in bounded-degree and general sparse graphs. *ACM Trans. Algorithms*, 5(2):22:1–22:28, 2009. doi:10.1145/1497290.1497298.

14 Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Structures and Algorithms*, 20(2):165–183, 2002. doi:10.1002/RSA.10013.

15 Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6):1012–1042, 2006. doi:10.1016/j.jcss.2006.03.002.

16 T. Rast. Testing triangle-freeness in general graphs: Upper bounds. Bachelor thesis, Dept. of Mathematics, ETH, Zurich, 2006.

17 Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM J. Comput.*, 41(4):1074–1093, 2012. doi:10.1137/110828691.

A Appendix

▶ **Theorem 30** (Multiplicative Chernoff's Bound). *Let $X_1, \ldots, X_n$ be identical independent random variables ranging in $[0,1]$, and let $p = \mathbb{E}[X_1]$. Then, for every $\gamma \in (0,2]$, it holds that*

$$\Pr\left[\left|\frac{1}{n}\cdot\sum_{i\in[n]} X_i - p\right| > \gamma\cdot p\right] < 2\cdot e^{-\gamma^2 pn/4}\,. \tag{6}$$

Testing Depth First Search Numbering

Artur Czumaj
Department of Computer Science and DIMAP, University of Warwick, Coventry, UK
University of Cologne, Germany

Christian Sohler
University of Cologne, Germany

Stefan Walzer
Karlsruhe Institute of Technology, Germany

Abstract

Property Testing is a formal framework to study the computational power and complexity of sampling from combinatorial objects. A central goal in standard graph property testing is to understand which graph properties are testable with sublinear query complexity. Here, a graph property P is testable with a sublinear query complexity if there is an algorithm that makes a sublinear number of queries to the input graph and accepts with probability at least 2/3, if the graph has property P, and rejects with probability at least 2/3 if it is ε-far from every graph that has property P.

In this paper, we introduce a new variant of the bounded degree graph model. In this variant, in addition to the standard representation of a bounded degree graph, we assume that every vertex v has a unique label $\texttt{num}(v)$ from $\{1, \ldots, |V|\}$, and in addition to the standard queries in the bounded degree graph model, we also allow a property testing algorithm to query for the label of a vertex (but *not* for a vertex with a given label).

Our new model is motivated by certain graph processes such as a DFS traversal, which assign consecutive numbers (labels) to the vertices of the graph. We want to study which of these numberings can be tested in sublinear time. As a first step in understanding such a model, we develop a *property testing algorithm for discovery times of a DFS traversal* with query complexity $O(n^{1/3}/\varepsilon)$ and for constant $\varepsilon > 0$ we give a matching lower bound.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Randomized Algorithms, Graph Algorithms, Property Testing

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.78

Related Version *Full Version*: https://arxiv.org/abs/2509.05132

Funding *Artur Czumaj*: Research partially supported by the Centre for Discrete Mathematics and its Applications (DIMAP), by a Weizmann-UK Making Connections Grant, and by an IBM Award. *Stefan Walzer*: Research funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 465963632.

1 Introduction

Depth-first search (DFS) is one of the most useful and frequently used algorithmic primitives in graph algorithms. The DFS algorithm is known for well over a century [17, 23] as a technique for threading mazes and has been widely used in the design of graph and network algorithms since the late 1950s. DFS is a basic tool to traverse a graph in a structured way and is central to solve textbook problems such as connectivity, topological sorting [22], determining strongly connected components in directed graphs [21], biconnected components in undirected graphs [21], and to test planarity [15]. Because of its versatility and usefulness for solving many graph problems, DFS has become one of the most important tools in the design of algorithms for graphs, taught in the first year of many computing science study


© Artur Czumaj, Christian Sohler, and Stefan Walzer;
licensed under Creative Commons License CC-BY 4.0

programs. One of the most useful combinatorial properties of DFS is the *DFS numbering of vertices*, which is *the order in which a DFS algorithm discovers all vertices of the input graph* [21]. (See also Algorithms 1 and 2, where $N(v)$ denotes the set of neighbors of vertex v.)

Algorithm 1 DFS numbering of G.

Input: undirected graph $G = (V, E)$
Output: $\mathtt{num} : V \to \{1, \ldots, |V|\}$

```
1 mark all vertices as undiscovered
2 t ← 1 // smallest unused ID
3 for v ∈ V do // arbitrary order
4     if v is undiscovered then
        DSFVisit(v)
```

Algorithm 2 DSFVisit(v).

```
1 mark v as discovered
2 num(v) ← t
3 t ← t + 1
4 for u ∈ N(v) do // arbitrary order
5     if v is undiscovered then
        DSFVisit(v)
```

We remark that sometimes, e.g., in the widely used textbook by Cormen et. al. (see Chapter 20.3 in [5]), the DFS numbering is also used together with finishing numbers. We also remark that the labeling is not unique and depends on the ordering in which the vertices are traversed in the for-loops of Algorithms 1 and 2.

▶ **Definition 1** (DFS numberings). *Let $G = (V, E)$ be a labeled undirected graph on n vertices. A labeling $\mathtt{num} : V \to \{1, \ldots, n\}$ is called a* DFS numbering of G *if DFS gives rise to it for some order of processing the vertices in the for-loops of Algorithms 1 and 2.*

Given a wide applicability of DFS and DFS numbering, a natural problem is to verify whether a given graph is correctly DFS-numbered, i.e., the labels assigned to the vertices are a DFS numbering for some ordering of vertices and neighbors. The latter problem is easy to solve by simulating a DFS using the given numbering and locally verifying that the DFS is performed correctly. However, is it also possible to *approximately* verify whether the graph is properly DFS-numbered using a sampling based algorithm? An answer to this question sheds some light on how the local structure of a DFS-numbered graph (as implicitly given by the sample distribution) is related to the global DFS numbering.

We will study this question in the *framework of Property Testing* (see, e.g., the monographs [4, 9]). Property Testing provides a formal setting to study approximate decision problems. The goal is to distinguish objects that satisfy the tested property from those that are ε-far from every object that satisfies the property. Here, ε-far means that one has to modify more than an ε-fraction of the object's description to obtain an object that has the property. A property testing algorithm requires some form of sampling access to the object.

In this paper we will introduce a variant of the standard bounded-degree graph model [12] that in addition to the standard setting, allows also *label queries*. We will assume that there exists some labeling that assigns unique labels from the set $\{1, \ldots, |V|\}$ to the vertex set V, i.e., the labeling is a bijection $\mathtt{num} : V \to \{1, \ldots, |V|\}$. We say that a graph with maximum degree d is ε-far from a DFS-numbered graph, if we have to modify (insert or delete) more than $\varepsilon|V|$ edges to obtain a graph that is correctly DFS-numbered[1]. We do not allow to modify labels, though we notice that allowing to modify labels would also be a valid model (see Section 2.1 for some discussion). Our motivation to not consider it here was merely to stick as closely to the standard testing model as possible.

[1] While one often uses the bound of $\varepsilon d|V|$ edges instead of $\varepsilon|V|$ edges, in the setting when $d = O(1)$ these terms differ only by a constant factor and as such there are essentially indistinguishable.

In this new framework we study a fundamental problem of *whether the input labeled undirected graph is properly DFS-numbered* or *it is ε-far from having a valid DFS numbering*. Our main result is a tight (for constant d and ε) complexity bound for this task. First, we show that any tester for the DFS numbering requires $\Omega(n^{1/3})$ queries, and then, we complement this bound with an algorithm that tests DFS numbering with $O(n^{1/3}/\varepsilon)$ queries.

The proof of the lower bound (Theorem 7 in Section 4) is by constructing two families $(G_n)_{n\in\mathbb{N}}$ and $(B_n)_{n\in\mathbb{N}}$ of *good* and *bad* random labeled graphs for which we will show that distinguishing between these families is necessary for any DFS-tester. Then we will show that distinguishing between these families requires $\Omega(n^{1/3})$ queries.

The proof of the upper bound (Theorem 11 in Section 5) relies on a characterization of labelings that are ε-far from a valid DFS numbering. The characterization is described in a form of some conflicts between the labelings and we design two algorithms detecting such conflicts: one for local conflicts and one for global conflicts. By combining these two algorithms we will obtain an algorithm that tests DFS numbering with $O(n^{1/3}/\varepsilon)$ queries.

1.1 Related work

Property Testing was introduced by Rubinfeld and Sudan [20] and first studied in the dense graph setting by Goldreich and Ron [10]. Constant time testability in the dense graph model has been fairly well understood [2] and is closely related to the regularity lemma. In our paper we build upon the bounded degree graph model, as introduced by Goldreich and Ron [12]. First results in this model included testers for properties such as connectivity, k-connectivity and being Eulerian [12]; bipartiteness [11] and cycle freeness [6] can be tested in the bounded degree graph model in $O(\frac{\sqrt{n}}{\varepsilon^{O(1)}})$ time. A series of papers proved that every hyperfinite property of bounded degree graphs is testable in constant time [3, 7, 14, 18]. Also, every constant time testable property in bounded degree graphs is either finite or contains a hyperfinite property [8]. Furthermore, it is known that every first-order logic property of bounded degree graph with an $\exists\forall$ quantification is constant time testable while some properties with a $\forall\exists$ quantification are not [1]. Further works in the bounded degree graph model include, e.g., testability using proximity oblivious testers [13].

2 Preliminaries: The model

In this paper, let $G = (V, E)$ denote an undirected graph with $n = |V|$ vertices and $m = |E|$ edges. Let $N(v)$ denote the set of *neighbors of* v, i.e., $N(v) := \{u \in V : \{v, u\} \in E\}$, and d be an upper bound on the maximum number of edges incident to any vertex in V. We will assume that G is a labeled graph, i.e., there is a bijection (permutation) $\mathtt{num} : V \to [n]$ that assigns numbers to the vertices of G and we use $[n] := \{1, \ldots, n\}$.

2.1 Bounded degree graph model for *labeled* graphs

In this section we describe how our algorithm can access the input graph $G = (V, E)$ of maximum degree d with vertex labeling $\mathtt{num}$. Our model is a slight modification of the standard bounded degree graph model [12] that in addition *allows access to labels*. As usual in this model, we assume that $V = [n]$ and n as well as d are given to the algorithm. The algorithm can ask the following two types of queries and receives an answer in constant time:

- **Neighbor queries:** for every vertex $v \in V$ and every $1 \le i \le d$, one can query the ith neighbor of vertex v.
- **Label queries:** for every vertex $v \in V$, one can query the label of v, $\mathtt{num}(v)$.

Observe that we allow access to vertices and their neighbors, and we can check the label $\mathtt{num}(v)$ of any vertex v, ***but we have no access to vertices through their labels*** $\mathtt{num}$. In particular, *to access vertex v with a given label* $\mathtt{num}$*, we have to query the oracle until such vertex is returned.* This is a special feature distinguishing our model from the standard model used in graph property testing and making the problem challenging: we have a labeled graph, but we cannot access the graph (its vertices) through the labels! Instead, the only way to access the graph is by taking any vertex $v \in V = [n]$ and either querying its label $\mathtt{num}(v)$ (via the label query) or querying its ith neighbor (via a neighbor query).

The *query complexity* of a testing algorithm is the number of oracle queries.

A property testing algorithm (in short, *property tester*) for DFS-numbered graphs is an algorithm that has access to an input graph as described above and that accepts the input with probability at least 2/3, if G is a graph with $\mathtt{num}$ being a valid DFS-numbering. The algorithm rejects G with probability at least 2/3 if G is ε-far from having a valid DFS numbering $\mathtt{num}$ according to the following definition. (The algorithm developed in this paper is a *property tester with one-sided error*, i.e., it always accepts, if $\mathtt{num}$ is a DFS-numbering.)

▶ **Definition 2** (ε-far from DFS numberings in bounded degree graphs)**.** *Let $G = (V, E)$ be an undirected graph on n vertices with maximum degree at most d and let $\mathtt{num} : V \to [n]$ be a bijection that assigns labels to V. We say the labeled graph G $\mathtt{num}$* is ε-far from having a valid DFS numbering*, if one has to modify[2] more than εn edges in G to obtain a graph $G' = (V, E')$ of maximum degree at most d for which $\mathtt{num}$ is a valid DFS numbering.*

Implicitly labeled graphs. To simplify the presentation, we will often assume $\mathtt{num}(v) = v$. However, *we will not use this knowledge in the algorithm* as the model prevents us from querying $\mathtt{num}^{-1}$. Our tester will only ask for a random vertex or for a vertex that occurred as the neighbor of a previously queried vertex.

Further thoughts about the model: Modifying the labels. Observe that in general, it might be natural to consider a revised definition of a labeling $\mathtt{num}$ being ε-far from a valid DFS numbering, where while defining graph G' in Definition 2, in addition to edge modifications, one would allow also for modifications of the labels[3]. We observe that for bounded degree graphs, adding the labels modifications does not change the problem significantly.

▶ **Lemma 3.** *Let $\mathtt{num}$ be a permutation of V to $\{1, \ldots, n\}$. For a given bounded degree graph $G = (V, E)$, a labeling $\mathtt{num}$ is ε-far from a valid DFS numbering according to the edges modifications if and only if $\mathtt{num}$ is $\Theta(\varepsilon)$-far from a valid DFS numbering according to the edges and labels modifications.*

Proof. Observe that the number of modifications of the edges to obtain a valid DFS numbering is not smaller than the number of modifications of the edges and the labels to obtain a valid DFS numbering. Therefore, if for a given bounded degree graph G, a labeling $\mathtt{num}$ is ε-far from a valid DFS numbering according to the edges and labels modifications then $\mathtt{num}$ is also ε-far from a valid DFS numbering according to the edges modifications.

---

[2] Modification of the edges means edge insertions and deletions, i.e., we require $|E \triangle E'| > \varepsilon n$, where $\triangle$ is the symmetric difference.

[3] We use the following revised definition: a labeling $\mathtt{num}$ *is ε-far from a valid DFS numbering* of G if for any graph $G' = (V, E')$ of maximum degree at most d with a valid DFS numbering $\mathtt{num}'$ we have $|E \triangle E'| + |\{v \in V : \mathtt{num}(v) \neq \mathtt{num}'(v)\}| \geq \varepsilon n$.

For the other direction, if num is ε-far from a valid DFS numbering according to the edges modifications then we can simulate modification of the labels by modification of the edges: to assign a given label to a vertex we just remove all its incident edges and add all edges incident to the vertex with the label sought. (Observe that a vertex might have had some edges of its own that have to be removed but by correcting the labels one by one, we can ensure that once we fix vertex v by assigning it to label i with vertex u having label i before, then later we will have to fix the label of vertex u, resolving the issue.) Observe that if we apply this operation to all vertices with the labels to be changed, then we do not increase the maximum degree, and hence, modifying of k labels can be simulated by modifying at most $2dk$ edges. Therefore, if a labeling num is ε-far from a valid DFS numbering according to the edges modifications then num is also $(2d\varepsilon)$-far from a valid DFS numbering according to the edges and labels modifications. ◀

Why should the labels be a permutation? Our model assumes that the input labeling num is a permutation (bijection) of V to $\{1 \dots, n\}$, but it may be natural to consider the case when num is *not necessarily a permutation* but rather an arbitrary function from V to $\{1 \dots, n\}$, allowing repetitions of labels. Observe that already the simple problem of testing if num is a permutation or is ε-far from being a permutation (also known as the element distinctness problem) is known to require $\Theta(\sqrt{n}/\varepsilon)$ queries (see, e.g., [19]). In view of that bound, the best what we could hope for without assuming that num is a permutation of V to $\{1 \dots, n\}$ is to achieve query complexity of $\Theta(\sqrt{n}/\varepsilon)$. And this can be easily obtained by combining our algorithm in Theorem 11 with the known algorithms testing element distinctness, leading to a testing algorithm with $\Theta(\sqrt{n}/\varepsilon)$ queries.

3 Basic Properties of DFS Numberings

We begin with a review of basic DFS terminology, introduce some notions of our own, and make a few useful observations. (We also refer to standard textbooks, e.g., [5, 16].)

The DFS algorithm, as given in Algorithm 1, numbers every vertex even if G is *disconnected* due to the outer loop over all vertices. Every vertex is initially *undiscovered*. When DSFVisit(v) is called, v becomes *discovered*. We say v is *active* as long as the call DSFVisit(v) persists and is *finished* as soon as it ends. The vertex p that initiates the call of DSFVisit(v) is the *parent* of v. If the call of DSFVisit(v) is initiated from the outer loop then we say that v is an *orphan* with *virtual parent* 0. The idea is that the outer loop iterating over all vertices is like a call to DSFVisit(0) where 0 is a virtual vertex connected to all other vertices. Each connected component of G has exactly one orphan, which receives the smallest number in its connected component. At any point during the execution the *white path* consists of all active vertices including the virtual vertex 0, with every active vertex (other than 0) connected to its parent. The DFS numbers appear in increasing order along the white path. In an implicitly labeled graph $G = (V, E)$, we define $p(v)$ for $v \in V$ as

$$p(v) := \begin{cases} \max\{u \in N(v) \cap [v-1]\} & \text{if } N(v) \cap [v-1] \neq \emptyset \\ 0 & \text{otherwise.} \end{cases},$$

▷ Claim 4. If the numbers in an implicitly labeled graph correspond to a DFS numbering then $p(v)$ is the (possibly virtual) parent of v.

Proof. If v is an orphan then it has the smallest number in its connected component so $\{u \in N(v) : u < v\} = \emptyset$. Hence $p(v) = 0$, which is the virtual parent of orphans by definition.

Now assume v was discovered from a non-virtual parent $x \in V$. Then $x \in N(v)$ with $x < v$ so $x \in \{u \in N(v) : u < v\}$. To see that x is the maximum of $\{u \in N(v) : u < v\}$, consider $x' \in \{x+1, \dots, v-1\}$. Immediately before v is discovered, x is at the end of the white path, hence the largest active vertex. Since $x' > x$ has been discovered, x' must be finished. Hence all neighbours of x' have been discovered. Since v is not discovered we have $x \notin N(v)$ so $x \notin \{u \in N(v) : u < v\}$. ◁

Observe that any edge $e = \{u, v\}$ with $u < v$ in an implicitly labeled graph corresponds to an *interval* $[u, v]$ representing a range of elements with respect to the DFS numbering. The analysis of the inter-relation between such intervals plays a central role in our analysis.

▶ **Definition 5.** *A pair* $(v, \{u, w\}) \in V \times E$ *is a* ***conflicting pair*** *if* $p(v) < u < v < w$.

The following central lemma shows that the absence of conflicting pairs is necessary and sufficient for the validity of a DFS numbering (we defer a simple proof to the full version).

▶ **Lemma 6.** *Let* $G = (V, E)$ *be an implicitly labeled graph. The following are equivalent.*

(i) *G has a valid DFS numbering.*

(ii) *There exists no conflicting pair.*

4 Testing DFS Numbering Requires $\Omega(n^{1/3})$ Queries

In this section, we prove our first main result.

▶ **Theorem 7.** *Every property tester for the property of having a valid DFS-labeling has a query complexity of* $\Omega(n^{1/3})$.

Let $\varepsilon > 0$ be sufficiently small (any $\varepsilon \leq \frac{1}{33}$ would do). The proof of Theorem 7 is by constructing two families $(G_n)_{n\in\mathbb{N}}$ and $(B_n)_{n\in\mathbb{N}}$ of *good* and *bad* random labeled graphs for which we will show that distinguishing between these families is necessary for any DFS-tester. Then we will show that distinguishing between these families requires $\Omega(n^{1/3})$ queries.

4.1 Construction of good and bad random labeled graphs (G_n) and (B_n)

Let $n, N \in \mathbb{N}$. Each graph G_n and B_n consists of $\lfloor \frac{n}{8N} \rfloor$ *arms* of $8N$ vertices each. The roots of these arms are connected with some binary tree that is the same for G_n and B_n.

There are four types of arms, as described in details in Figure 1. Each arm of G_n is a copy of $(G1)$ or $(G2)$ chosen independently and uniformly at random. Similarly, each arm of B_n is a copy of $(B1)$ or $(B2)$ chosen independently and uniformly at random.

The labeling of G_n is then obtained by starting in the root of the tree T and using within the arms the relative ordering defined by the numbering of the arms $(G1)$ or $(G2)$. For an example, see, e.g., Figure 3. (Since each arm has the same number of vertices, the choice of one arm, whether in $(G1)$ or $(G2)$, does not affect the numbering of other arms.)

Similarly (see Figure 4), the labeling of B_n is defined by starting at a root of T and using within the arms the relative ordering defined by the numbering of the arms $(B1)$ or $(B2)$.

4.2 Properties of DFS numberings of random labeled graphs G_n and B_n

Let us first notice that our construction of good random labeled graphs $(G_n)_{n\in\mathbb{N}}$ easily ensures that they have valid DFS numberings (with probability 1).

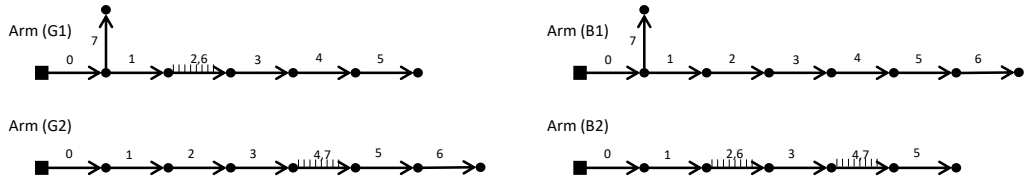
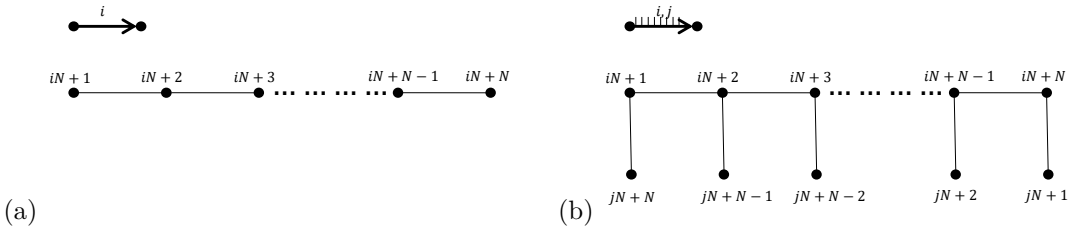


Figure 1 Four types of *arms* $(G1)$, $(G2)$, $(B1)$, and $(B2)$ used in our construction of $(G_n)_{n\in\mathbb{N}}$ and $(B_n)_{n\in\mathbb{N}}$. Each arm starts with a root, denoted by ■. Each link $\xrightarrow{\ i\ }$ corresponds to a path on N vertices and labels $iN+1,\ldots,i(N+1)$ ascending in the direction of the arrow (see Figure 2(a)). We also have a *comb graph* $\xrightarrow{\ i,j\ }$ (see Figure 2(b)), which is obtained from $\xrightarrow{\ i\ }$ by adding N new vertices, adding a matching between the new vertices and the vertices from $\xrightarrow{\ i\ }$, and labeling the new vertices with $jN+1,\ldots,(j+1)N$, this time descending in the direction of the arrow (i.e., the vertex with label $iN+k$ is adjacent to the vertex with label $jN+N-k+1$).



(a) (b)

Figure 2 Graphs corresponding to (a) a path $\xrightarrow{\ i\ }$ and (b) a comb graph $\xrightarrow{\ i,j\ }$.

▶ **Lemma 8.** *G_n has a valid DFS numbering.* ◀

A situation is different for bad random labeled graphs $(B_n)_{n\in\mathbb{N}}$. While the arms of type $(B1)$ also locally maintain a valid DFS-order, the arms of type $(B2)$ do not (because of the two comb graphs). As the result, the construction of $(B_n)_{n\in\mathbb{N}}$ typically gives labelings that are ε-far from valid DFS numberings.

▶ **Lemma 9.** *Let $0 \le \varepsilon \le \frac{1}{33}$ and let n be sufficiently large. Then B_n has a labeling that is ε-far from a valid DFS numbering with probability $1-o(1)$.*

Proof. Consider the numbering for $(B2)$. Let us fix any $k \in \{1,2,\ldots,N\}$ and focus on vertices p_1, c_1, p_2, c_2 with labels $2N+k, 7N-k+1, 4N+k, 8N-k+1$, respectively. (For example, in Figure 4 in the top branch (where numbers have an offset of 1), if we took $k=2$ then we would have $\langle p_1, c_1, p_2, c_2\rangle$ to be vertices with labels $\langle 11, 28, 19, 32\rangle$ in B_{64}.) Observe that the fact that $\mathtt{num}(p_1) < \mathtt{num}(p_2) < \mathtt{num}(c_1) < \mathtt{num}(c_2)$ implies that this is not a valid DFS numbering as long as c_1 is the DFS-child of p_1 and c_2 is the DFS-child of p_2. Therefore, to turn the labeled graph into one consistent with DFS numbering, one has to add or delete at least one edge incident to a vertex from $\{p_1, c_1, p_2, c_2\}$. Since such a quadruple is obtained for each $k \in \{1,\ldots,N\}$ and since each of these quadruples is disjoint (for example, in Figure 4, the there are four such quadruples formed by vertices with labels $\langle 10, 29, 18, 33\rangle$, $\langle 11, 28, 19, 32\rangle$, $\langle 12, 27, 20, 31\rangle$, and $\langle 13, 26, 21, 30\rangle$), one needs to modify at least $\frac{N}{2}$ edges to obtain a valid DFS numbering. Since the expected number of copies of $(B2)$ in B_n is $\frac{1}{2}\cdot\lfloor\frac{n}{8N}\rfloor$, the expected number of changes required in B_n to obtain a valid DFS numbering is at least $\frac{N}{4}\cdot\lfloor\frac{n}{8N}\rfloor$. For sufficiently large n, a standard Chernoff bound implies that graph B_n is $\frac{1}{33}$-far from having a valid DFS numbering with high probability. ◀

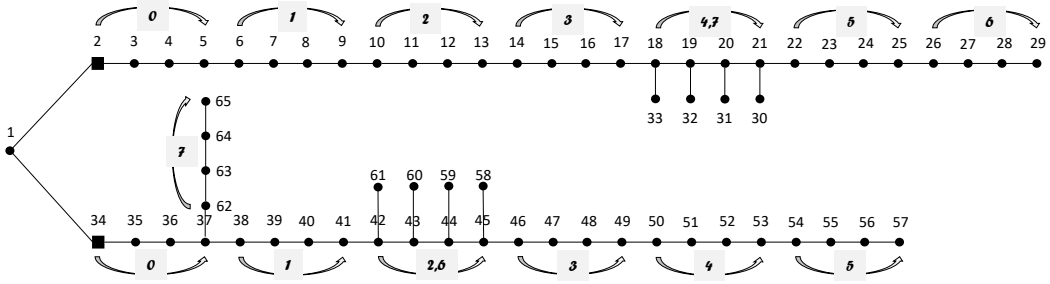


Figure 3 An example of labeling of G_n with $n = 64$, $N = 4$, using $\lfloor \frac{n}{8N} \rfloor = 2$ arms of size $8 \cdot 4 = 32$. Vertices with labels 1, 2, 34 are in the tree T, the top branch (with labels 2–33) corresponds to arm $(G2)$ and the bottom branch (with labels 34–65) corresponds to arm $(G1)$. For each arm, we also marked the corresponding parts defining it.

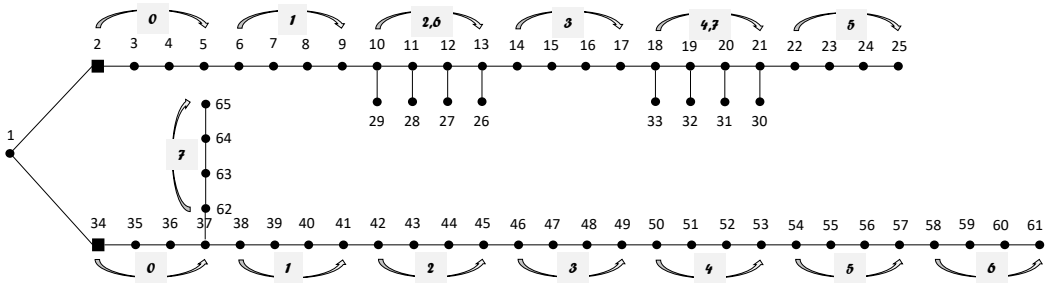


Figure 4 An example of labeling of B_n with $n = 64$, $N = 4$, using $\lfloor \frac{n}{8N} \rfloor = 2$ arms of size $8 \cdot 4 = 32$. Vertices with labels 1, 2, 34 are in the tree T, the top branch (with labels 2–33) corresponds to arm $(B2)$ and the bottom branch (with labels 34–65) corresponds to arm $(B1)$. For each arm, we also marked the corresponding parts defining it.

4.3 Hardness of distinguishing between good and bad labeled graphs

Our next central lemma provides a lower bound for the number of queries of any algorithm ALG that distinguishes between the families of random labeled graphs $(G_n)_{n\in\mathbb{N}}$ and $(B_n)_{n\in\mathbb{N}}$. We will assume that the input I_n for $n \in \mathbb{N}$ on which ALG requires q_n oracle queries is obtained by first selecting a random bit $b \in \{0, 1\}$ and then setting $I_n = \begin{cases} G_n & \text{if } b = 0, \\ B_n & \text{if } b = 1. \end{cases}$

▶ **Lemma 10.** *Let $0 < \xi \leq \frac{N^3}{n}$. For every randomized algorithm ALG' that receives I_n as input, performs $q_n \leq \sqrt{\frac{\xi n}{4N}}$ queries, and outputs a bit $b' \in \{0, 1\}$, we have $\Pr[b = b'] \leq \frac{1}{2} + \xi$.*

Proof. Let ALG' be an algorithm that receives I_n as its input. We assume that ALG' can query the oracle for any vertex u by submitting an ID of u, and the oracle returns the label $\mathfrak{num}(u)$ of u, and the IDs of all neighboring vertices. Further, without loss of generality, we assume that the IDs $1, \ldots, n$ are randomly assigned to the vertices of the graph. Therefore ALG' is limited to querying for a random vertex (a *random query*) or for a vertex that has been previously found as a neighbor of an earlier queried vertex (an *explorative query*).

Without loss of generality, we can assume that the vertices selected by random queries are chosen in the order $v_1, v_2, \ldots$ before the run of ALG'; let $V_R = \{v_1, \ldots, v_{q_n}\}$ be the sequence of these first q_n random vertices (ALG can select only these vertices). Let $\mathcal{E}$ be the random

event that no two vertices $u, u' \in V_R$ come from the same arm of I_n. We prove the following:

$$\Pr[\neg\mathcal{E}] \leq \xi \tag{1}$$
$$\Pr[b = b'|\mathcal{E}] = \tfrac{1}{2} \tag{2}$$

Observe that the two inequalities (1)–(2) imply Lemma 10:

$$\Pr[b = b'] = \Pr[b = b'|\mathcal{E}] \cdot \Pr[\mathcal{E}] + \Pr[b = b'|\neg\mathcal{E}] \cdot \Pr[\neg\mathcal{E}] \leq \tfrac{1}{2} \cdot 1 + 1 \cdot \xi \leq \tfrac{1}{2} + \xi \ .$$

In order to prove inequality (1), let us first define C to be the number of pairs $i \neq j$ with $i, j \leq q_n$ such that both v_i and v_j come from the same arm of I_n, or in other words, so that v_i and v_j belong to the same copy of arm $(B1)$. We have,

$$\mathbb{E}[C] = \sum_{1 \leq i < j \leq q_n} \Pr[v_i \text{ and } v_j \text{ are in the same arm}] \leq \binom{q_n}{2} \cdot \frac{8N}{n} \leq \frac{4q_n^2 N}{n} \leq \frac{4(\frac{\xi n}{4N})N}{n} \leq \xi \ .$$

Therefore we can conclude (1) by Markov's inequality: $\Pr[\neg\mathcal{E}] = \Pr[C \geq 1] \leq \mathbb{E}[C] \leq \xi$.

Next, we want to prove inequality (2). Let V_R^* be the set of vertices reachable from vertices in V_R in at most q_n steps, that is, $V_R^* = \{u : \exists_{1 \leq i \leq q_n} \operatorname{dist}(v_i, u) \leq q_n\}$. Let $I_n(V_R^*)$ be the subgraph of I_n induced by the vertex set V_R^*. We observe that ALG' will make its decision solely on seeing some subgraph of $I_n(V_R^*)$. Hence, the output b' of ALG' is a (random) function of $I_n(V_R^*)$. Now, we claim that conditioned on $\mathcal{E}$, the random variables b and $I_n(V_R^*)$ are stochastically independent, which in turn, would imply identity (2).

To prove that b and $I_n(V_R^*)$ are independent, let us consider a single vertex v_i from V_R and the subgraph $I_n(v_i)$ induced by vertices with distance at most q_n from v_i. If $I_n(v_i)$ contains at least one vertex from T, then $I_n(v_i)$ consists solely of some of the vertices from T and some vertices that are close to the roots of some of the arms (within parts denoted by $\xrightarrow{0}$ of the arms, since each such part has length N and we have $q_n \leq \frac{N}{2}$ since $q_n \leq \sqrt{\frac{\xi n}{4N}}$ and $\xi \leq \frac{N^3}{n}$). Since these parts are identical in G_n and B_n, such paths share no information on b.

Otherwise, if $I_n(v_i)$ contains no vertex from T, then $I_n(v_i)$ is a part of an arm of I_n. But then, due to $q_n \leq \frac{N}{2}$, at most one joint[4] of this arm is contained in $I_n(v_i)$. Since the labels of the roots of the arms are the same in G_n and B_n, it is always well-defined what the offset of a label within its arm is. For more details, see the full version of the paper. ◀

4.4 Hardness of testing DFS numbering (proof of Theorem 7)

By Lemmas 8 and 9, any algorithm ALG that accepts a labeled graph with a valid DFS numbering with probability at least $\frac{2}{3}$ and rejects a labeled graph with a DFS numbering that is ε-far (for $0 < \varepsilon \leq \frac{1}{33}$) from being valid DFS numbering with probability at least $\frac{2}{3}$, must be able to distinguish with probability at least $\frac{5}{8}$ between the families of good and bad random labeled graphs $(G_n)_{n \in \mathbb{N}}$ and $(B_n)_{n \in \mathbb{N}}$. However by Lemma 10, by setting $\xi = \frac{1}{8}$ and $N = \lfloor n^{1/3} \rfloor$, the tasks of distinguishing between these families requires $\Omega(n^{1/3})$ queries. ◀

5 Testing DFS Numbering with $O(n^{1/3}/\varepsilon)$ Queries

Our second main result shows that the lower bound in Theorem 7 is asymptotically tight.

[4] By a *joint* we mean an endpoint of a path or comb from which the arm was stitched together.

▶ **Theorem 11.** *Let $0 < \varepsilon < 1$. There is an algorithm that with oracle access to a labeled bounded degree undirected graph G on n vertices, performs $O(n^{1/3}/\varepsilon)$ queries to the oracle and accepts, if G has a valid DFS numbering, and rejects with probability at least $\frac{2}{3}$, if G is ε-far from having a valid DFS numbering.*

(One can show that our tester achieves also the same $O(n^{1/3}/\varepsilon)$ running time.)

For the rest of the section, let G be a corresponding labeled graph equipped with a possibly inappropriate DFS numbering $\mathtt{num} : V \to [n]$. As in Section 3, we assume that $\mathtt{num}$ *is implicit* so that we can, for instance, write $v < w$ for $v, w \in V$ instead of $\mathtt{num}(v) < \mathtt{num}(w)$.

Outline. Our approach to prove Theorem 11 is first to extend Lemma 6 (which characterizes valid DFS numberings) to describe a simple and useful property of labelings that are ε-far from a valid DFS numbering. In Lemma 13 in Section 5.1, we will show that if the numbering is ε-far from a valid DFS numbering then not only we have $\Omega(\varepsilon n)$ conflicting pairs (in the sense of Lemma 6), but in fact we have $\Omega(\varepsilon n)$ conflicting pairs that are "unrelated." Once we have this property, the task in hand will be to detect any of such conflicting pair. We observe that there are two types of conflicting pairs, *local pairs* involving vertices whose DFS numbers are close to each other, and *global conflicts*. In order to detect local conflicts, we first develop (in Section 5.2) some basic tools to traverse a given graph following DFS numbering. Once we know how to traverse the graph, we can design an algorithm that can determine if a given pair $(v, \{u, w\})$ is conflicting pair. Unfortunately, this algorithm is efficient only if vertex u or w is close to $p(v)$ or v (that is, if one of $\mathtt{num}(u) - \mathtt{num}(p(v))$, $\mathtt{num}(v) - \mathtt{num}(u)$, $\mathtt{num}(w) - \mathtt{num}(v)$ is small), and so we can use this approach to deal with local conflicts. In order to study global conflicts, we notice that if vertices u and w are far away from vertices $p(v)$ and v, then in fact a conflicting pair $(v, \{u, w\})$ can be also extended to multiple vertices v. Once we have that property, we will show that if we sample randomly $\Theta(n^{1/3}/\varepsilon)$ vertices and $\Theta(n^{1/3}/\varepsilon)$ edges, then if there were many global conflicts, then there would be one that is determined by one of the sampled vertices and one of the sampled edges.

5.1 Properties of labelings that are ε-far from any valid DFS numbering

We begin with describing a useful property of labelings that are ε-far from a valid DFS numbering that they have $\Omega(\varepsilon n)$ conflicting pairs that are "unrelated." Recall that by Lemma 6 a numbering is a valid DFS numbering if and only if there is no conflicting pair $(v, \{u, w\}) \in V \times E$ with $p(v) < u < v < w$. In that case, when $p(v) < u < v < w$, we speak of a conflict involving vertex v and edge $\{u, w\}$. While Lemma 6 characterizes valid and invalid DFS numberings, we will need a stronger claim about properties of numberings that are ε-far from valid DFS numberings. For that, we need to understand numberings that are not ε-far from valid DFS numberings because we can modify the input graph with at most εn edge modifications to ensure that the resulting graph will have a valid DFS numbering.

Observe that Lemma 6 provides a simple tool to edit the input labeled graph to obtain a valid DFS numbering – to remove all conflicting pairs. With that in mind, the following claim provides a simple fix to remove all conflicts involving a specific vertex or all conflicts involving a specific edge (notice that the resulting graph may violate our degree bound d, but one can address this issue using a "degree reduction framework," see the full version.

▶ **Lemma 12.** *Let $v \in V$ and $\{u, w\} \in E$ with $u < w$.*

- **(i)** *Adding the edge $\{v-1, v\}$ to G (and doing nothing if $\{v-1, v\}$ is already present) does not create any new conflicts and resolves all conflicts involving v.*
- **(ii)** *Removing $\{u, w\}$ from G and adding $\{w-1, w\}$ (if not already present) does not create any new conflicts and resolves all conflicts involving $\{u, w\}$.*

Proof.

(i) After the edit we have $p(v) = v - 1$ so no $u \in V$ can satisfy $p(v) < u < v$ any more, meaning all conflicts involving v are resolved. We do not create any new conflicts because $p(v-1)$ does not change and the new edge $\{v-1, v\}$ cannot be involved in a conflict because (again) no v' can satisfy $v - 1 < v' < v$.

(ii) Deleting $\{u, w\}$ clearly resolves all conflicts of this edge. However, new conflicts involving w may arise since $p(w)$ may change if we had $p(w) = u$ before. By (i) we can fix these by adding $\{w - 1, w\}$. ◀

Lemma 12 shows that adding or removing a few edges can resolve many related conflicts. We use this claim to show that in order for G to be ε-far from having a valid DFS numbering, not only do there have to be many conflicts, but in fact there have to be *many mutually unrelated conflicts*. To formalize this idea we consider the bipartite *conflict graph* $\mathcal{C} = (V, E, C)$ where $C \subseteq V \times E$ contains an edge $(v, \{u, w\})$ precisely if it is a conflicting pair. The intuition of mutually unrelated conflicts corresponds to a matching in $\mathcal{C}$. We have the following.

▶ **Lemma 13** (ε-far DFS numberings). *If G is ε-far from having a valid DFS numbering then there is a matching $M \subseteq C$ of size $|M| \geq \varepsilon n/5$ in $\mathcal{C}$.*

Proof. We will prove Lemma 13 by showing that if G is ε-far from having a valid DFS numbering then $\mathcal{C}$ has a vertex cover of size at least $\varepsilon n/5$, for if not, then we could modify at most εn edges of G to make the labeling to be a valid DFS numbering of the resulting graph. Then Lemma 13 follows from König's theorem.

Let $M \subseteq C$ be a maximum matching in $\mathcal{C}$. By Kőnig's theorem there is a vertex cover $\mathcal{VC} \subseteq V \cup E$ of size $|\mathcal{VC}| = |M|$, i.e., a set of vertices and edges of G (vertices of $\mathcal{C}$) such that for every conflict pair $(v, \{u, w\})$ (for every edge of $\mathcal{C}$) $v \in \mathcal{VC}$ or $\{u, w\} \in \mathcal{VC}$. We can fix all conflicts in G in $\leq 2|\mathcal{VC}|$ edits by applying for each $v \in \mathcal{VC} \cap V$ the fix of Lemma 12 (i) (one edit) and for each $\{u, w\} \in \mathcal{VC} \cap E$ the fix of Lemma 12 (ii) (two edits). We obtain a graph G^* with a valid DFS numbering. However, the vertices that appeared in a fix in the role of v or w may have degree $d + 1$ in G^*, higher than permitted. By our "degree reduction framework" (see full version), another $3|\mathcal{VC}|$ edits suffices to transform G^* into $G^{**} = (V, E^{**})$ with maximum degree d while maintaining the validity of the DFS numbering. Overall $|E^{**} \triangle E| \leq 5|\mathcal{VC}|$ and since G is ε-far we have $|E^{**} \triangle E| \geq \varepsilon n$. Hence $|M| = |\mathcal{VC}| \geq \varepsilon n/5$. ◀

Lemma 13 immediately implies a simple tester detecting ε-far instances: we randomly sample $\Omega(\sqrt{n/\varepsilon})$ vertices and $\Omega(\sqrt{n/\varepsilon})$ edges, and then with a constant probability one of the sampled vertices v and one of the sampled edges e will form a conflicting pair (v, e).

▶ **Corollary 14.** *There is an algorithm that with oracle access to a labeled bounded degree graph G on n vertices, performs $O(\sqrt{n/\varepsilon})$ queries to the oracle and accepts, if G has a valid DFS numbering, and rejects with probability $\frac{2}{3}$, if G is ε-far from having a valid DFS numbering.* ◀

In what follows, we will show how to improve this result, as promised in Theorem 11.

5.2 Navigating (would-be) DFS trees

In this section, we develop tools to find conflicting pairs for vertices. We first show how to traverse a graph using consecutive labels (a simple proof is deferred to the full version).

▶ **Fact 15.** *Let T be an ordered tree of bounded degree and S its canonical DFS ordering*[5]*. For a randomly selected vertex v from T, given oracle access to T, it is possible to locate the successor (and the predecessor) of v in S, if one exists, with $O(1)$ queries to T in expectation.*

Using Fact 15, we can prove the following lemma.

▶ **Lemma 16** (DFS-NEXT)**.** *There is an algorithm DFS-NEXT that for a given vertex $v \in V$:*

- *If the numbering of G is a valid DFS numbering then either vertex $v+1$ is returned or END-OF-COMPONENT is returned if v is the largest vertex in its connected component.*
- *The expected (over vertices in V) number of queries to G of algorithm DFS-NEXT is $O(1)$.*

There also exists an algorithm DFS-PREV with corresponding properties.

Proof. Let T be the ordered tree on $V \cup \{0\}$ rooted at 0 with the edges $\{p(v), v\}$ for $v \in V$ and children ordered ascending by number. If the numbering of G is a valid DFS numbering then (by Claim 4) T is precisely the DFS tree that gave rise to the numbering. We can navigate to successors and predecessors in T (by Fact 15) with an expected number of $O(1)$ queries to G per step. The only problem is that the virtual vertex 0 cannot be accessed and has unbounded degree. Whenever we would have to access it, we return END-OF-COMPONENT instead, which is appropriate because a valid DFS backtracks to 0 precisely if a connected component has been fully explored. (If the numbering of G is invalid, the produced answer may be meaningless, but it is still obtained within the claimed expected time budget.) ◀

5.3 Testing for conflicts

Lemma 13 implies that an ε-far instance has $\Omega(\varepsilon n)$ pairwise distinct vertices and edges involved in conflicts. In this section, we will extend that approach and study separately *local conflicting pairs* and *global conflicting pairs* in order to improve the tester from Corollary 14. This notion depend on a locality parameter that we will later choose as $\ell = n^{1/3}$.

▶ **Definition 17.** *Let $(v, \{u, w\}) \in V \subseteq E$ with $p(v) < u < v < w$ be a conflicting pair. We speak of a **local conflict** of the following (not mutually exclusive) types* **(L1)** *if $u - p(v) \leq \ell$ and $p(v) \neq 0$,* **(L2)** *if $v - u \leq \ell$,* **(L3)** *if $w - v \leq \ell$. In all other cases we speak of a **global conflict**:* **(G)** *if $p(v) = 0$ and $\max\{v-u, w-v\} > \ell$; or if $\max\{u - p(v), v-u, w-v\} > \ell$.*

Informally, local conflicts occur when u is close (in the sense of its DFS number) to $p(v)$ or v, or w is close v, in which case one can traverse the input graph to efficiently detect the conflict. For global conflicts, some more global approach will be needed.

5.3.1 Testing for local conflicts

We can detect local conflicts by sampling vertices at random and walking forwards and backwards in the (supposed) DFS order using Lemma 16 as follows.

▶ **Lemma 18.** *There is an algorithm that with $O(\ell/\varepsilon)$ queries in expectation accepts all valid DFS numberings and rejects with probability $2/3$ if there is a matching $M \subseteq V \subseteq E$ of size at least $\frac{\varepsilon n}{30}$ consisting of conflicting pairs of type* **(L1)**. *The same applies to types* **(L2)**, **(L3)**.

Proof. Consider the following procedure WALK-FROM-$p(v)$ that is given $v \in V$ as input. It first checks if $p(v) \neq 0$. If so, it attempts to locate vertices $p(v)+1, p(v)+2, \ldots$ using DFS-NEXT from Lemma 16 until one of the following happens.

[5] The DFS starts at the root and respects the order of children when visiting them in pre-order traversal.

- If DFS-NEXT reaches vertices out of order, then report the error.
- If DFS-NEXT reaches a vertex u that has a neighbour $w > v$ then report the error.
- If v or $p(v) + \ell$ is reached without finding an error, then conclude that v is not involved in a conflict of type **(L1)**.

Clearly WALK-FROM-$p(v)$ finds an error if v is involved in a conflict of type **(L1)** and by Lemma 16 it performs $O(\ell)$ queries in expectation (over all $v \in V$). Our algorithm to detect conflicts of type **(L1)** is now simply to repeat WALK-FROM-$p(v)$ for many v sampled uniformly at random. Since the $\frac{\varepsilon n}{30}$ vertices matched in M are pairwise distinct, a random vertex from V is involved in a conflict of type **(L1)** with probability $\frac{\varepsilon}{30}$. If we sample $\frac{60}{\varepsilon}$ vertices at random then the probability of sampling at least one v involved in a conflict of type **(L1)** is

$$\geq 1 - (1 - \varepsilon/30)^{60/\varepsilon} \geq 1 - \mathrm{e}^{-(\varepsilon/30)\cdot(60/\varepsilon)} = 1 - \mathrm{e}^{-2} \geq 2/3 \ .$$

The expected total number of queries amounts to $O(\ell/\varepsilon)$, which concludes the claim.

For conflicts of type **(L2)** a similar procedure WALK-BACKWARDS-FROM-v works. For conflicts of type **(L3)** a procedure WALK-FORWARDS-FROM-v would *not* work because DFS-NEXT might report END-OF-COMPONENT before w is reached (and without us being aware of w's existence). A procedure WALK-BACKWARDS-FROM-w does work, however. Note that we have to sample $\frac{60d}{\varepsilon}$ edges uniformly at random which is still $O(1/\varepsilon)$ because d is constant. ◀

5.3.2 Testing for global conflicts

Lemma 18 provides an efficient tool to detect local conflicts but for global conflicts we use a different approach. We rely on Lemma 13 that promises that there is a matching $M \subseteq V \subseteq E$ of size at least $\varepsilon n/10$ consisting of conflicting pairs. Therefore, if most of conflicts defining the matching M are global, we can use the following lemma.

▶ **Lemma 19.** *There is an algorithm that performs $O(\sqrt{n/\ell}/\varepsilon)$ queries in expectation and accepts all valid DFS numberings and that rejects with probability at least $2/3$ if there is a matching $M \subseteq V \subseteq E$ of size at least $\varepsilon n/10$ consisting of conflicting pairs of type (G).*

Proof. Let us partition matching M into strips $M_1 \oplus M_2 \oplus \cdots \oplus M_{\lceil \frac{n}{\ell} \rceil}$ according to the number of u, so that $M_j = \{(v, \{u, w\}) \in M : \lceil u/\ell \rceil = j\}$. Let us define sets $V_j := \{v \in V : \exists_{e \in E}\ (v, e) \in M_j\}$ and $E_j := \{e \in E : \exists_{v \in V}(v, e) \in M_j\}$. Let $\bar{v}_j$ be the median of V_j. Let V_j^- be those vertices from V_j that are at most $\bar{v}_j$ and E_j^+ those edges from E_j matched to a vertex that is at least $\bar{v}_j$. Let $m_j := |V_j^-|$ and note that $|E_j^+| = m_j$ and that $m_j \geq \frac{|M_j|}{2}$.

We claim that *every pair* in $V_j^- \times E_j^+$ is a conflicting pair. Indeed, let $v_1 \in V_j^-$, $\{u_2, w_2\} \in E_j^+$ with $u_2 < w_2$, and let $p_1 = (v_1, \{u_1, w_1\})$ and $p_2 = (v_2, \{u_2, w_2\})$ be the corresponding pairs in M_j. Then we observe the following:

$$p(v_1) \overset{(1)}{\leq} \max\{0, u_1 - \ell\} \overset{(2)}{<} u_2 \overset{(2)}{<} u_1 + \ell \overset{(1)}{<} v_1 \overset{(3)}{\leq} \bar{v}_j \overset{(3)}{\leq} v_2 \overset{(4)}{<} w_2 \ .$$

To see this, we notice the following:

(1) p_1 is of type (G). Hence $u_1 - p(v_1) > \ell$, unless $p(v_1) = 0$; moreover $v_1 - u_1 > \ell$;
(2) since both p_1 and p_2 are in M_j, we have $|u_1 - u_2| < \ell$;
(3) by the definition of V_j^- and E_j^+ the values of v_1 and v_2 fall on the respective side of the median $\bar{v}_j$;
(4) p_2 is a conflicting pair.

In view of the above, since every pair in $V_j^- \times E_j^+$ is a conflicting pair, we observe that over all j we do not just have $m = |M|$ conflicting pairs to work with, but a collection of bicliques with $\sum_{i=1}^{\lceil n/\ell \rceil} (m_j)^2$ conflicting pairs in total. If we happen to get hold of a $v \in V_j^-$ and an $e \in E_j^+$ for the same j, then we have a conflicting pair (v, e).

In order to find a conflicting pair using the approach above, let us sample (i.u.r.) s vertices from V and then (i.u.r.) s edges from E, where s will be set up momentarily.

For any $1 \leq i, r \leq s$, let $X_{i,r}$ to be the indicator random variable that the ith sampled element from V and the rth sampled element from E are (respectively) from the sets V_j^- and E_j^+ for the same j. Observe that if we define random variable X as $X = \sum_{i=1}^{s} \sum_{r=1}^{s} \mathbb{E}[X_{i,r}]$, then by the arguments above, if X is positive then we have detected a conflicting pair. Using the second moment method, we get prove the following lemma.

▶ **Lemma 20.** *Let $s \geq c\sqrt{\frac{n^3}{\ell \cdot |M|^2}}$ for a sufficiency large constant c and let $s = \omega((d/\varepsilon)^3)$. Then* $\Pr[X > 0] \geq 0.99$.

Hence, if we sample i.u.r. at least $c \cdot \sqrt{\frac{n^3}{\ell |M|^2}}$ vertices from V and at least $c \cdot \sqrt{\frac{n^3}{\ell |M|^2}}$ edges from E for a sufficiently large constant c, then with a constant probability we will detect a conflicting vertex. At the same time, if the DFS numbering is valid then the algorithm will accept it. This yields Lemma 19 since in our setting $|M| = \Omega(\varepsilon n)$ and $\ell = n^{1/3}$. ◀

5.4 Putting all together: the proof of Theorem 11

Now we are ready to complete the proof of Theorem 11. Our algorithm runs the algorithms from Lemma 18 (responsible or conflicts of type **(L1)**, **(L2)** and **(L3)**) and from Lemma 19 one after the other. If any of them rejects G then we reject G, otherwise we accept G. The *expected* total running time is $O(\ell/\varepsilon + \sqrt{n/\ell}/\varepsilon)$, which with $\ell = \Theta(n^{1/3})$ gives $O(n^{1/3}/\varepsilon)$ as claimed. (The query complexity can be made $O(n^{1/3}/\varepsilon)$ using Markov inequality.)

Concerning correctness, it is clear that instances with valid DFS numberings are always accepted. If G is ε-far from a valid DFS numbering then by Lemma 13 there is a matching $M \subseteq V \subseteq E$ of $|M| \geq \varepsilon n/5$ conflicting pairs. Since each matching edge falls into (at least) one type, at least one of the following statements holds.

- There is a matching $M_{\textbf{(L1)}}$ of $|M_{\textbf{(L1)}}| \geq \varepsilon n/30$ conflicting pairs of type **(L1)**.
- There is a matching $M_{\textbf{(L2)}}$ of $|M_{\textbf{(L2)}}| \geq \varepsilon n/30$ conflicting pairs of type **(L2)**.
- There is a matching $M_{\textbf{(L3)}}$ of $|M_{\textbf{(L3)}}| \geq \varepsilon n/30$ conflicting pairs of type **(L3)**.
- There is a matching $M_{\textbf{(G)}}$ of $|M_{\textbf{(G)}}| \geq \varepsilon n/10$ conflicting pairs of type **(G)**.

In each case, the corresponding algorithm rejects G with probability $2/3$ so overall we reject G with probability at least $2/3$. ◀

6 Conclusions

In this paper we introduced a variant of the standard bounded-degree graph model in the property testing setting that works for labeled graphs and allows also label queries. We demonstrated the strength of the model on our new study of DFS numbering. Our main technical contribution is a tight analysis for detecting whether the input labeled graph is properly DFS-numbered or it is ε-far from having a valid DFS numbering. We demonstrated that this task can be solved with $\Omega(n^{1/3}/\varepsilon)$ queries and also $\Omega(n^{1/3})$ queries are necessary.

We observe that while our analysis is presented for undirected graphs, similar arguments hold also for ***directed graphs***. The lower bound from Theorem 7 trivially extends to directed graphs and a careful pass through the algorithm in Theorem 11 shows that the analysis can be extended accordingly. However, to implement this algorithm efficiently in our setting, we need to allow access to incoming and outgoing edges. (See the full version of the paper.)

Our analysis can be also extended (but only for undirected graphs) to the ***DFS finishing numbers*** (FIN-numberings [5]). We can show that (for undirected graphs) a numbering $\mathtt{num} : V \to [n]$ is a valid FIN numbering iff the reverse numbering $\overleftarrow{\mathtt{num}}$ with $\overleftarrow{\mathtt{num}}(i) = n+1-i$ is a valid DFS numbering. This immediately implies that our results for testing valid DFS numberings extend to testing valid FIN numberings in undirected graphs.

References

1 Isolde Adler, Noleen Köhler, and Pan Peng. On testability of first-order properties in bounded-degree graphs and connections to proximity-oblivious testing. *SIAM Journal on Computing*, 53(4):825–883, 2024. doi:10.1137/23M1556253.

2 Noga Alon, Eldar Fischer, Ilan Newman, and Asaf Shapira. A combinatorial characterization of the testable graph properties: It's all about regularity. *SIAM Journal on Computing*, 39(1):143–167, 2009. doi:10.1137/060667177.

3 Itai Benjamini, Oded Schramm, and Asaf Shapira. Every minor-closed property of sparse graphs is testable. *Advances in Mathematics*, 223(6):2200–2218, 2010.

4 Arnab Bhattacharyya and Yuichi Yoshida. *Property Testing – Problems and Techniques*. Springer Verlag, 2022. doi:10.1007/978-981-16-8622-1.

5 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 4th Edition*. MIT Press and McGraw-Hill, 2022.

6 Artur Czumaj, Oded Goldreich, Dana Ron, C. Seshadhri, Asaf Shapira, and Christian Sohler. Finding cycles and trees in sublinear time. *Random Structures & Algorithms*, 45(2):139–184, 2014. doi:10.1002/RSA.20462.

7 Artur Czumaj, Asaf Shapira, and Christian Sohler. Testing hereditary properties of non-expanding bounded-degree graphs. *SIAM Journal on Computing*, 38(6):2499–2510, 2009. doi:10.1137/070681831.

8 Hendrik Fichtenberger, Pan Peng, and Christian Sohler. Every testable (infinite) property of bounded-degree graphs contains an infinite hyperfinite subproperty. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 714–726, 2019. doi:10.1137/1.9781611975482.45.

9 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.

10 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

11 Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bunded degree graphs. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 289–298, 1998. doi:10.1145/276698.276767.

12 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. doi:10.1007/S00453-001-0078-7.

13 Oded Goldreich and Dana Ron. On proximity-oblivious testing. *SIAM Journal on Computing*, 40(2):534–566, 2011. doi:10.1137/100789646.

14 Avinatan Hassidim, Jonathan A. Kelner, Huy N. Nguyen, and Krzysztof Onak. Local graph partitions for approximation and testing. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 22–31, 2009. doi:10.1109/FOCS.2009.77.

15 John E. Hopcroft and Robert E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974. doi:10.1145/321850.321852.

16 Jon M. Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2013.

17 Édouard Lucas. *Récreations Mathématiques*. Librairie Albert Banchard, Paris, 1882.

18 Ilan Newman and Christian Sohler. Every property of hyperfinite graphs is testable. *SIAM Journal on Computing*, 42(3):1095–1112, 2013. doi:10.1137/120890946.

19 Sofya Raskhodnikova, Dana Ron, Amir Shpilka, and Adam D. Smith. Strong lower bounds for approximating distribution support size and the distinct elements problem. *SIAM Journal on Computing*, 39(3):813–842, 2009. doi:10.1137/070701649.

20 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996. doi:10.1137/S0097539793255151.

21 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

22 Robert Endre Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6:171–185, 1976. doi:10.1007/BF00268499.

23 G. Tarry. Le problème des labyrinthes. *Nouvelles Annales de Mathématiques, 3e série*, 14:187–190, 1895.

Semi-Streaming Algorithms for Hypergraph Matching

Henrik Reinstädtler
Heidelberg University, Germany

S M Ferdous
Pacific Northwest National Laboratory, Richland, WA, USA

Alex Pothen
Purdue University, West Lafayette, IN, USA

Bora Uçar
CNRS and LIP, ENS de Lyon, France
UMR5668 (CNRS, ENS de Lyon, Inria, UCBL1), France

Christian Schulz
Heidelberg University, Germany

Abstract

We propose two one-pass streaming algorithms for the $\mathcal{NP}$-hard hypergraph matching problem. The first algorithm stores a small subset of potential matching edges in a stack using dual variables to select edges. It has an approximation guarantee of $\frac{1}{d(1+\varepsilon)}$ and requires $\mathcal{O}((\frac{n}{\varepsilon})\log^2 n)$ bits of memory, where n is the number of vertices in the hypergraph, d is the maximum number of vertices in a hyperedge, and $\epsilon > 0$ is a parameter to be chosen. The second algorithm computes, stores, and updates a single matching as the edges stream, with an approximation ratio dependent on a parameter α. Its best approximation guarantee is $\frac{1}{(2d-1)+2\sqrt{d(d-1)}}$, and it requires only $\mathcal{O}(n)$ memory.

We have implemented both algorithms and compared them with respect to solution quality, memory consumption, and running times on two diverse sets of hypergraphs with a non-streaming greedy and a naive streaming algorithm. Our results show that the streaming algorithms achieve much better solution quality than naive algorithms when facing adverse orderings. Furthermore, these algorithms reduce the memory required by a factor of 13 in the geometric mean on our test problems, and also outperform the offline Greedy algorithm in running time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Hypergraphs

Keywords and phrases hypergraph, matching, semi-streaming

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.79

Related Version *Full Version*: https://arxiv.org/abs/2502.13636 [38]

Supplementary Material *Dataset*: https://doi.org/10.5281/zenodo.15778413
Software: https://github.com/HeiHGM/Streaming [37]
archived at swh:1:dir:4e4522550296fb38202457ce7371bf7034ae45d9

Funding We acknowledge support by DFG grant SCHU 2567/8-1. Moreover, we like to acknowledge Dagstuhl Seminar 24201 on discrete algorithms on modern and emerging compute infrastructure. Alex Pothen's research was supported by the U.S. Department of Energy grant SC-0022260. S M Ferdous's research was supported by the Laboratory Directed Research and Development Program at PNNL.



© Henrik Reinstädtler, S M Ferdous, Alex Pothen, Bora Uçar, and Christian Schulz;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

We propose two streaming algorithms for the hypergraph matching problem, derive their approximation ratios, and implement them to evaluate their practical performance on a large number of test hypergraphs. Recall that hypergraphs are a natural extension of graphs and can help to model our ever evolving earth and society. In a hypergraph, a hyperedge is a subset of vertices and can contain any number of them, instead of just two. The hypergraph matching problem asks for a set of vertex-disjoint hyperedges. Two common objectives in the hypergraph matching problem are to maximize the number or total weight of the matching hyperedges. The hypergraph matching problem has applications ranging from personnel scheduling [17] to resource allocations in combinatorial auctions [21]. The hypergraph matching problem with either of the objective functions is $\mathcal{NP}$-complete [24], but a simple Greedy algorithm is $1/d$-approximate, where d is the maximum size of a hyperedge [5, 28].

There are a few papers with computational studies for the hypergraph matching problem, in a single CPU in-memory setting [11] and in a distributed computing setting [23]. However, little to no attention was paid to increasing data sizes and approximation guarantees. Streaming and semi-streaming algorithms address this trend of ever-increasing data size. In a streaming setting, hyperedges arrive one by one in arbitrary order. The amount of memory that can be used is strictly bounded by the size of the final solution. In the case of hypergraph matching, this is $\Theta(n)$, where n is the number of vertices in the hypergraph, because every vertex can be matched at most once. For a semi-streaming setting, this criterion is relaxed to allow for an additional polylog factor. Furthermore, establishing a bound on the degree of suboptimality is essential for evaluating the solution's effectiveness. For matchings in graphs, there is a semi-streaming algorithm having an approximation guarantee of $\frac{1}{2}$ by Paz and Schwartzman [33].

The algorithm by Paz and Schwartzman requires one variable per vertex, the dual variable, and uses a stack to store candidate matching edges. When an edge appears in the stream, the algorithm adds it to a stack if its weight dominates the sum of the duals of its vertices, and then updates the duals with the difference between the edge weight and the sum of duals. Otherwise, the edge is discarded. After all edges have been streamed, non-conflicting edges are added from the stack beginning at the top. Ghaffari and Wajc [18] give a simplified proof of the approximation guarantee of this algorithm using the primal-dual linear programming framework.

Our contributions are as follows. We first propose a novel streaming framework for hypergraph matching and prove an approximation guarantee in relation to the largest hyperedge size by extending the stack-based algorithm of Paz and Schwartzman [33] to hypergraphs. In essence, our algorithm puts hyperedges that potentially belong to a good matching on a stack, and in the end computes a matching out of those hyperedges. Given the maximum hyperedge size d (the maximum number of vertices in a hyperedge), we use primal-dual techniques to prove a $\frac{1}{d(1+\varepsilon)}$ approximation factor, where $\epsilon > 0$ is a parameter to be chosen. Our most memory-saving algorithm requires $\mathcal{O}(n \log^2 n/\varepsilon)$ bits of space. We then propose a second family of algorithms which do not need a stack and require less space and work by greedily swapping hyperedges from the current matching with the incoming ones. Like the first stack-based algorithm, this algorithm family requires $\mathcal{O}(|E| \cdot d)$ work. They have an approximation guarantee depending on a factor $\alpha > 0$, which can be tuned to result in a guarantee of $\frac{1}{(2d-1)+2\sqrt{(d-1)d}}$. In experiments, we show the competitiveness of our approaches and benchmark them on a set of social link hypergraphs and a large set of instances from hypergraph partitioning. We compare our algorithms with a Naive streaming

algorithm that maintains a maximal matching in the hypergraph by adding a hyperedge from the stream to the match if it does not overlap in any vertex with the current matching edges, and a non-streaming Greedy algorithm. The stack-based algorithms reduce the memory consumption by up to 13 times in comparison to the non-streaming Greedy algorithm on social link hypergraphs. We investigate the impact of ordering the hyperedges in the stream and show that our stack algorithm can handle them better than the non-streaming Greedy and Naive streaming algorithms, while requiring only 26% more time than the Naive algorithm. We show the impact of the parameters α and ε on their respective algorithms.

The rest of the paper is organized as follows. After introducing the notation and related work in Section 2, we show our approximation guarantee for an adaptation of the Paz-Schwartzman semi-streaming algorithm and discuss further improvements in Section 3. Section 4 introduces our greedy swapping algorithm for streaming based on McGregors [31] algorithm. These approaches are then extensively evaluated by experiments in Section 5. We conclude in Section 6. The Appendix of the full version of the paper contains the proof of the approximation guarantee for the algorithm from Section 4, and additional statistics on the test problems as well.

2 Preliminaries

2.1 Basic Concepts

Hypergraphs. A *weighted undirected hypergraph* $H = (V, E, W)$ consists of a set V of n vertices and a set E of m hyperedges. Each hyperedge e is a subset of vertices and is assigned a positive weight by the weight function $W : E \to \mathbb{R}_{>0}$. The number of vertices in a hyperedge e is its size, and is denoted by $|e|$, and the maximum size of a hyperedge or rank of the hypergraph is denoted by $d := \max_{e \in E} |e|$. For clarity and brevity, we refer to a hyperedge simply as an edge when it is evident from the context that a hypergraph is under consideration.

Matching. A subset of (hyper-)edges $M \subset E$ is a *matching*, if all (hyper-)edges in M are pairwise disjoint, i.e., only at most one (hyper-)edge is selected at every vertex. A matching M is called maximal, if there is no (hyper-)edge in E which can be added to M without violating the matching constraint. The weight of a matching is defined by $W(M) := \sum_{e \in M} W(e)$ and a maximum matching is a matching with the largest weight.

Related $\mathcal{NP}$-hard Problems. The unweighted hypergraph matching problem is closely related to the maximum independent set and the k-set packing problems. Both problems are $\mathcal{NP}$-hard [27]. An independent set in a graph is a subset of vertices in which no two vertices are adjacent. There is a simple transformation from hypergraph matching to maximum independent set using the line graph of the hypergraph; in the line graph, every hyperedge becomes a vertex, and two such vertices are connected if the corresponding hyperedges share a common vertex. Given a ground set S and some subsets $S_1, \ldots, S_n$, each of size at most k, the k-set packing problem asks to select the maximum number of disjoint subsets. It can be translated to the hypergraph matching setting, by choosing the set S to correspond to the vertices V, while the subsets $S_1, \ldots, S_n$ correspond to the hyperedges.

(Semi-)Streaming Algorithms. If the input size exceeds the memory of a machine, a typical solution is to stream the input. There are several definitions for streaming in graphs and hypergraphs. When (semi-)streaming, the (hyper-)edges of a (hyper-)graph are usually

presented in an arbitrary (even adverse) order one-by-one in several passes. In this paper we consider having only one pass over the input. In a streaming setting, the memory is strictly bounded by the solution size and, hence, for matching in hypergraphs, it is limited by the number of vertices. When using the semi-streaming model the memory is bounded by $\mathcal{O}(n \cdot \text{polylog}(n))$.

Approximation Factors. Algorithms can be classified into three categories: exact algorithms, heuristics without approximation guarantees, and approximation algorithms. The quality of an approximation algorithm is measured by comparing its solution's value to that of an optimal solution. For an instance I of a maximization problem, let the optimum objective be denoted by $M(I)$. If an algorithm $\mathcal{A}$ is guaranteed to find a solution that is bigger than $\alpha M(I)$ for every instance, where $\alpha \in \mathbb{R}^+_{<1}$ and is chosen to be as large as possible, then $\mathcal{A}$ provides an α-approximation guarantee. In some communities, the convention is to report $\frac{1}{\alpha} > 1$ as the approximation ratio, although we do not follow it here. For an overview of techniques to design approximation algorithms, we refer the reader to Williamson and Shmoys [42].

(Integer) Linear Programs. Many optimization problems can be formulated as integer linear programs (ILP). In a maximization problem, an (integer) linear program finds an (integer) vector x with components x_i, that maximizes a linear cost function $\sum c_i x_i$ such that a constraint $Ax \leq b$, where A is a matrix, is satisfied, with typically additional constraints on the components of x, e.g., $x_i \geq 0$. If the variables of the problem are integer, some problems are $\mathcal{NP}$-hard, while other problems (where the constraint matrix is unimodular) are solvable in polynomial time [43]. When the integer constraint is dropped, any linear program is solvable in polynomial time [44]. For every linear program one can find a dual problem [42]. Given a maximization problem as described before, the dual problem is to find a vector $y_i \geq 0$, that minimizes $\sum b_i y_i$ subject to $A^T y \geq c$. The weak duality theorem [42] states that for any primal maximization problem, the dual minimization problem for any feasible solution has an objective value larger than the optimal solution of the primal problem. The strong duality theorem states that if the primal problem has an optimal solution then the dual is solvable as well and the optimum values are the same. For a more detailed introduction, we refer the reader to the Appendix of [42].

2.2 Related Work

Matching is a well-studied problem in computer science, and here we give a brief overview of matchings in graphs and hypergraphs.

Matching in Graphs and Streaming. The polynomial-time complexity of matchings in graphs is one of the classical results in theoretical computer science [12]. While Preis [36] presents the first linear time $\frac{1}{2}$-approximation, Drake and Hougardy [10] show a simpler algorithm with the same approximation ratio by path growing (PGA) in linear time. A number of other 1/2-approximation algorithms have been developed, including the proposal-based Suitor algorithm of Manne and Halappanavar [29]. Pettie and Sanders [34] propose a $\frac{2}{3} - \varepsilon$ approximation with expected running time of $\mathcal{O}(m \log \frac{1}{\varepsilon})$. The GPA algorithm by Maue and Sanders [30] bridges the gap between greedy and path-searching algorithms, showing that a combination of both works best in practice. Pothen, Ferdous and Manne [35] survey these approximation algorithms. Birn et al. [6] develop a parallel algorithm in the CREW PRAM model with $\frac{1}{2}$-approximation guarantee and $\mathcal{O}(\log^2 n)$ work. Feigenbaum et al. [14] present a $\frac{1}{6}$-approximation for the weighted matching problem in the semi-streaming setting

using a blaming-based analysis. McGregor [31] develop a multipass algorithm that returns a $\frac{1}{2+\varepsilon}$ approximation in $\mathcal{O}(\varepsilon^{-3})$ rounds, with the initial matching having an approximation guarantee of $\frac{1}{3+2\sqrt{2}}$. Paz and Schwartzman [33] give a $\frac{1}{2+\varepsilon}$-approximation algorithm, which employs a dual solution to admit candidate edges into a stack, while also updating the dual solution. The matching is constructed by removing edges from the top of the stack, and those that do not violate the matching property are added to the solution. The resulting matching is not necessarily maximal. Ghaffari and Wajc [18] provide a simpler proof of the approximation ratio using a primal-dual analysis. Ferdous et al. [15] show empirically that the algorithm by Paz and Schwartzman can compete quality-wise with offline $\frac{1}{2}$-approximation algorithms like GPA, while requiring less memory and time. Ferdous et al. [16] present two semi-streaming algorithms for the related weighted k-disjoint matching problem, building upon the algorithms of Paz and Schwartzman, and Huang and Sellier [25, 33] for streaming b-matching.

Hypergraph Matching. Hazan et al. [24] prove that for the maximum k-set packing problem there is no approximation within a factor of $\Omega(k/\log k)$ unless $\mathcal{P} = \mathcal{NP}$. This directly translates to d-uniform cardinality hypergraph matching, where every edge has size d, with $k = d$ and the number of edges selected is maximized. Dufosse et al. [11] investigate reduction rules and a scaling argument for finding large matchings in d-partite, d-uniform hypergraphs. There are several approximation results and local search approaches, most notably by Hurkens and Schrijver [26] and Cygan [7] with an approximation guarantee of $\mathcal{O}(\left(\frac{d+1+\varepsilon}{3}\right))$. Hanguir and Stein [23] propose three distributed algorithms to compute matchings in hypergraphs, trading off between quality guarantee and number of rounds needed to compute a solution.

The Greedy algorithm for maximum weight hypergraph matching, which considers hyperedges for matching in non-increasing order of weights, is $1/d$-approximate, where d is the maximum size of a hyperedge [5, 28]. For the weighted k-set packing problem Berman [4] introduces a local search technique. Improving on these results, Neuwohner [32] presents a way to guarantee an approximation threshold of $\frac{k}{2}$. We are not aware of any practical implementations of these techniques. For the more general weighted hypergraph b-matching problem, Großmann et al. [22] present effective data-reduction rules and local search methods. In the online setting, when hyperedges arrive in an adversarial order, and one must immediately decide to include the incoming hyperedge or not in the matching, Trobst and Udwani [40] show that no (randomized) algorithm can have a competitive ratio better than $\frac{2+o(1)}{d}$.

We are unaware of any studies or implementations for streaming hypergraph matching.

3 Stack-based Algorithm

We now present our first algorithm to tackle the hypergraph matching problem in the semi-streaming setting. Our algorithm uses dual variables to evaluate whether a streaming hyperedge is good enough to be retained in a stack. Once the stream has been ingested, the stack is evaluated from top to bottom to determine a matching, achieving an approximation guarantee of $\frac{1}{d(1+\varepsilon)}$. We also discuss a more permissive and lenient update function that allows more hyperedges into the stack. Finally, we discuss the space complexity of our algorithms.

Algorithm 1 shows our framework for computing a hypergraph matching in a streaming setting. This algorithm is an extension of an algorithm by Paz and Schwartzman [33] proposed for graphs. The algorithm starts with an empty stack and keeps a dual variable ϕ_v for each vertex v of the hypergraph. Throughout the algorithm the stack contains candidate hyperedges for inclusion in a matching.

For each streamed hyperedge e, the algorithm checks if the weight of the dual variables of e's vertices and thereby the solution can be improved by adding e to the stack. More precisely, with $\Phi_e := \sum_{v \in e} \phi_v$, the algorithm checks if $W(e) \geq \Phi_e(1+\varepsilon)$. The parameter $\varepsilon \in \mathbb{R}_{\geq 0}$ is used to trade quality for memory. A smaller ε yields a better approximation guarantee, while a higher ε yields a smaller memory consumption. With $\varepsilon = 0$, the algorithm no longer provides memory guarantees. If $W(e) \geq \Phi_e(1+\varepsilon)$, then e is added to the stack, and the dual variables of the vertices of e are then updated using an update function. The update functions that we consider take $\mathcal{O}(1)$ time per vertex.

After all hyperedges have been streamed, a matching containing only the hyperedges stored in the stack is computed. To do that, our algorithm takes the hyperedges in reverse order from the stack and adds non-overlapping hyperedges to the matching. Note that hyperedges processed earlier that have conflicting heavier (later) hyperedges will be ignored. This is crucial to prove the performance guarantee later. The amount of total work needed for scanning one hyperedge e is $\mathcal{O}(|e|)$, because we need to sum up the dual variables of e's vertices and update them.

The update functions determine whether an approximation guarantee can be provided and directly impact the results. Upon processing a hyperedge e, the update function applied to the dual variable of a vertex $v \in e$ can use the prior value ϕ_v and sum $\Phi_e := \sum_{v \in e} \phi_v$ as well as e's size and weight. We define the following update function for proving the approximation guarantee

$$\phi_v^{\text{new}} := \delta_{\text{g}}(e, \phi_v, \Phi_e, W(e)) := \phi_v + \overbrace{W(e) - \Phi_e}^{w'_e}. \tag{1}$$

The δ_{g} function is exactly the function used by Paz and Schwartzman [33]. For a hyperedge e added to the stack, the $\phi(v)$ value of its endpoints is increased by the potential gain in matching weight, $w'_(e)$. This follows since $\phi(v)$ stores the gain in weight of all earlier edges incident on v that have been added to the stack thus far.

We introduce a different update function later.

At the core of this method are the variables ϕ_v for each v and the update mechanism. These core components and local ratio techniques are used to prove the approximation guarantee in the original work [33]. We follow the structure of the proof derived from a primal-dual analysis [18].

3.1 Approximation Guarantee

We make use of the primal-dual framework for showing the approximation guarantee. For general linear programming concepts, we refer the reader to books [1, 44], and for an excellent overview of primal-dual approximation method to the book [42].

We show that using δ_{g} update function (1) in our algorithm leads to a $\frac{1}{d(1+\varepsilon)}$-approximation, when the stack is unwound and hyperedges are selected in that order which they are on the stack. When streaming the hyperedges in descending order of weights, we can prove that the Greedy algorithm achieves a $1/d$ approximation factor.

We now proceed with a primal-dual analysis of an ILP formulation of the hypergraph matching problem. In our formulation, there is a binary decision variable associated with each hyperedge to designate if that hyperedge is selected to be in the matching. The objective function is to maximize the sum of the weights of the selected hyperedges. The constraint is to select at most one hyperedge containing a given vertex. The linear programming relaxation, shown in Figure 1a, is obtained by dropping the binary constraint on the hyperedge variables.

Algorithm 1 Simple Streaming Algorithm.

```
1: procedure StackStreaming(H = (V, E, W))
2:     S ← emptystack
3:    ∀v ∈ V : φ_v = 0
4:    for e ∈ E in any (even adverse) order do
5:       Φ_e ← Σ_{v∈e} φ_v
6:       if W(e) < Φ_e(1 + ε) then
7:          next
8:       end if
9:       S.push(e)
10:      for v ∈ e do
11:         φ_v ← δ(e, φ_v, Φ_e, W(e)) {update}
12:      end for
13:   end for
14:   M ← ∅
15:   while S ≠ ∅ do
16:      e ← S.pop()
17:      if ∀f ∈ M : f ∩ e = ∅ then
18:         M ← M ∪ {e}
19:      end if
20:   end while
21: end procedure
```

The objective value of the relaxation is naturally greater or equal than the integer version of the linear program. The dual problem of the relaxed hypergraph matching is given in Figure 1b. Following the weak duality theorem for linear programs, we know that any feasible solution of the dual has an objective value greater or equal to the objective value of any feasible primal solution. Furthermore, the optimal value of the linear program $\mathrm{OPT}(LP)$ is equal for both problems (strong duality). The first step for the proof is to check that the variables $(1+\varepsilon)\sum_{v\in e}\phi_v$ (from Algorithm 1) constitute a valid dual solution.

▶ **Observation.** *Function δ_g and Algorithm 1 generate valid dual solutions for all $\varepsilon \geq 0$.*

Proof. For each hyperedge e not on the stack, there was enough weight in the ϕ_v values of its vertices, when e was scanned in Line 6. In the update for every added hyperedge to the stack, all vertices ϕ_v values are increased by $w(e) - \Phi_e$ such that clearly the sum of vertex ϕ_v values is higher than the weight of the hyperedge just added. Therefore, for any hyperedge it holds $\sum_{v\in e}\phi_v \geq W(e)$, satisfying the dual equation (2). ◀

Such a valid dual solution has a greater objective value then the optimum solution of the relaxed dual, and the LP duality theorem gives an upper bound for every matching, including the optimal one M^* by $W(M^*) \leq \mathrm{OPT}(LP) \leq (1+\varepsilon)\sum_v \phi_v$.

Now, we connect the changes to the dual variables with the hyperedges that have already been processed. Define

$$\Delta\phi^e = \begin{cases} \sum_{v\in e}(\delta_g(e,\phi_v,\Phi_e,W(e)) - \phi_v) = |e|\,(W(e)-\Phi_e) & \text{if } e \in S \\ 0 & \text{else} \end{cases} \tag{3}$$

as the change to the dual $\sum_v \phi_v$ by inspecting e. We give a bound for the change of the dual variable w.r.t. the preceding hyperedges in Lemma 1.

Figure 1 Primal and Dual LPs for Hypergraph Matching.

(a) LP

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} W(e) x_e \\ \text{subject to} \quad & \\ & \forall v \in V \colon \sum_{e \ni v} x_e \leq 1 \\ & \forall e \in E \colon x_e \geq 0. \end{aligned}$$

(b) Dual LP

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} \phi_v \\ \text{subject to} \quad & \\ & \forall e \in E \colon \sum_{v \in e} \phi_v \geq W(e) \\ & \forall v \in V \colon \phi_v \geq 0. \end{aligned} \tag{2}$$

▶ **Lemma 1.** *For a hyperedge e, let $W'_e := W(e) - \Phi_e$. For each hyperedge $e \in E$ added to stack S, if we denote its preceding neighboring hyperedges (including itself) by $\mathcal{P}(e) := \{c \mid c \cap e \neq \emptyset, c \text{ added before } e\} \cup \{e\}$, then $W(e) \geq \sum_{e' \in \mathcal{P}(e)} \frac{1}{d} \Delta\phi^{e'} = \sum_{e' \in \mathcal{P}(e)} W'_{e'}$.*

Proof. From the definition (3), we have $\Delta\phi^e = |e|\, W'_e$ for δ_{g}, because of line 11 of Algorithm 1. $\Phi_e := \sum_{v \in e} \phi_v$ is defined as the previous value of the dual variables before inspecting e. Each of these dual values ϕ_v consists of the sum $\sum_{c \in \mathcal{P}(e) \text{ s.t. } v \in c} W'_c$ for all preceding hyperedges. This leads to $\Phi_e = \sum_{v \in e} \phi_v \geq \sum_{c \in \mathcal{P}(e) \setminus \{e\}} \frac{1}{|e|} \Delta\phi^{e'} \geq \sum_{c \in \mathcal{P}(e) \setminus \{e\}} \frac{1}{d} \Delta\phi^c$. So we can conclude $W(e) = W'_e + \Phi_e \geq \frac{1}{d} \Delta\phi^e + \sum_{e' \in \mathcal{P}(e) \setminus \{e\}} \frac{1}{d} \Delta\phi^{e'}$. ◀

The previous bound relates the weight of a hyperedge to the change in dual variables by its predecessors. In conclusion, we show that our algorithm returns a $\frac{1}{d(1+\varepsilon)}$-approximation.

▶ **Lemma 2.** *Algorithm 1 with δ_{g} function guarantees a $\frac{1}{d(1+\varepsilon)}$-approximation.*

Proof. We show a lower bound on the weight of any matching M constructed by the algorithm. For any hyperedge e not in the stack, $\Delta\phi^e = 0$, as when a hyperedge is not pushed into the stack, no dual variables are updated. Furthermore, any hyperedge in the stack that is not included in the matching must be a previously added neighbor of a matching hyperedge as defined in Lemma 1. Therefore, Lemma 1 applies, and the weight can be lower-bounded. The sum of changes $\sum_e \Delta\phi^e$ to ϕ is equal to the sum of dual variables $\sum_v \phi_v$ at the end. We have

$$W(M) = \sum_{e \in M} W(e) \overset{\text{L. 1}}{\geq} \sum_e \frac{1}{d} \Delta\phi^e \geq \frac{1}{d} \sum_e \Delta\phi^e = \frac{1}{d} \sum_v \phi_v \overset{\text{LP Duality}}{\geq} \frac{1}{d(1+\varepsilon)} W(M^*).$$

◀

3.2 Improving Solution Quality

Now we look into optimizing the solution quality. The design space for optimizations is vast, therefore we focus on simple yet effective techniques. We propose a second update function allowing more hyperedges than δ_{g} into the stack, with the same approximation guarantee.

Lenient Update Function. The δ_{g} in Equation 1 builds a dual solution much larger than needed. For every successfully added hyperedge, the difference between the current dual solution and the weight of the hyperedge is added to every vertex of the hyperedge. We address this by combining it with a scaling argument. The resulting function is

$$\delta_{\text{lenient}}(e, \phi_v, \Phi_e, W(e)) := \phi_v + (W(e) - \Phi_e)/\,|e|\,. \tag{4}$$

This function produces a valid dual solution, and Lemma 1 also holds. For every previously added neighboring hyperedge e' the change $\Delta\phi^e = W'_{e'}$ was distributed over all vertices of the hyperedge, so $\Phi_e \geq \sum_{e' \text{ added before } } \frac{1}{d}\Delta\phi^{e'}$. Lemma 2 follows and gives us the desired approximation factor of $\frac{1}{d(1+\varepsilon)}$.

3.3 Space Complexity Analysis

The space complexity of δ_g and δ_{lenient} can be deduced by simple counting arguments. Let W be the maximum normalized weight of a hyperedge in the hypergraph, i.e., $W := \frac{\max_{e\in E} W(e)}{\min_{e\in E} W(e)}$, and let W be $\mathcal{O}(\text{poly}(n))$. We discuss both update functions separately.

Guarantee Function. On every vertex we can observe up to $1 + \log_{1+\varepsilon}(W)$ incrementing events, because every change in the dual variables has to be bigger by a factor of $(1 + \varepsilon)$. This causes the stack to contain $\mathcal{O}(\sum_v (1 + \log W/\varepsilon))$ vertices in its edges. Each vertex in an edge requires $\log n$ bits. Therefore, the overall space complexity is $\mathcal{O}((1/\varepsilon)n \log^2 n)$ bits since W is $\mathcal{O}(poly(n))$.

Lenient Function. This function updates ϕ_v for every vertex in an edge e added to the stack by $\frac{W(e)-\Phi_v}{d}$, resulting in $1 + d \cdot \log_{1+\varepsilon}(W)$ possible increases to reach the total sum of W. Following the same argument, the stack's space complexity in bits is $\mathcal{O}((1/\varepsilon)nd \log^2 n)$. Note that with this update function the algorithm is semi-streaming only if d is $\mathcal{O}(\text{polylog}(n))$.

The time complexity per edge is in $\mathcal{O}(d)$ and $\Omega(md)$ overall in the scanning phase, as for every edge its vertices need to be scanned and optionally updated once. The unwinding of the stack takes d checks per hyperedge, when reusing the memory of ϕ_v from the previous step. Overall, since the stack size is naturally bounded by m, the complexity of this algorithm is $\mathcal{O}(md)$. In Section 5.2 we show the difference in the stack size in experiments under several orderings of the input.

4 Greedy Swapping Algorithm

We now propose a second streaming algorithm that computes, stores, and updates a matching in the hypergraph as the edges stream. It is conceptually similar to a streaming matching algorithm for graphs [31]. It requires a constant amount of memory per vertex, has an approximation factor that depends on the maximum size of a hyperedge d and a parameter α and obtains high-quality matchings in practice.

The proposed approach is described in Algorithm 2. We store for every vertex v a reference to the current matching hyperedge containing v; the $\perp$ sign symbolizes that no matching hyperedge contains v. For simplifying the presentation, we define $W(\perp) = 0$. When we inspect a hyperedge e, we compute the sum of the weights of its adjacent hyperedges that are currently in the matching. If the weight of the incoming hyperedge is larger than $(1 + \alpha)$ times the previous conflicting hyperedges, we first remove the previous hyperedges from all their vertices. Afterwards we can set the reference to the new incoming hyperedge. The overall space consumption of this algorithm is $\mathcal{O}(n)$. There are n references involved, and each hyperedge of size d holds the d vertices referencing it. Removing a hyperedge is linear in d, because up to d vertices in $\mathcal{B}_v$ must be set to $\perp$; it follows that the total work is $\mathcal{O}(d \cdot |E|)$.

We show its approximation guarantee of $\frac{1}{(1+\alpha)(\frac{d-1}{\alpha}+d)}$, which is optimal for $\alpha = \sqrt{(d-1)/d}$, in the Appendix of the full version. In our experimental evaluation, we look at various values of $\alpha \in \{0, 0.1, 1\}$ and $\alpha = \sqrt{(d-1)/d}$. Note that for $\alpha = 0$ the algorithm has no guarantee.

Algorithm 2 Swapping streaming algorithm.

```
1: procedure SwapSet((H = (V, E, W), α))
2:   ∀v ∈ V : B_v = ⊥{ Initialize best hyperedge to empty}
3:   W(⊥) := 0
4:   for e ∈ E in arbitrary order do
5:     C ← ⋃_{v∈e} B_v
6:     Φ_e ← W(C){ weight of hyperedges to be removed.}
7:     if W(e) ≥ (1 + α) · Φ_e then
8:       for v ∈ e do
9:         if B_v ≠ ⊥ then
10:          for w ∈ B_v do
11:            B_w ← ⊥ { Unmatch vertices in B.}
12:          end for
13:        end if
14:        B_v ← e
15:      end for
16:    end if
17:  end for
18:  M ← ⋃_{v∈V} B_v
19:  return M
20: end procedure
```

5 Experimental Evaluation

We now evaluate our algorithms with respect to solution quality, running time, and memory usage. Specifically, we address the following research questions:

- **RQ1:** How does the ordering of the hyperedges affect our metrics (memory, running time, and quality)?
- **RQ2:** How do other instance properties affect our algorithms in their memory needs?
- **RQ3:** How do our algorithms compare with offline greedy or naive streaming approaches?

5.1 Setup and Data Set

We implemented our approaches in C++ using g++-14.2 with full optimization turned on (-O3 flag). We tested on two identical machines, equipped with 128 GB of main memory and a Xeon w5-3435X processor running at 3.10 GHz having a L3 cache of 45 MB each. We repeat each experiment three times, and the results are compared only if experiments are run on the same machine and the same compute job. For memory consumption, we used the jemalloc malloc implementation [13]. The time needed for loading the hypergraph is not measured. We scheduled eight experiments (RQ1) and ten experiments (RQ2) to run in parallel, and the order of the experiments was randomized. In order to compare the results we use performance profiles as suggested by Dolan and Moré [9]. We plot the fraction of instances that could be solved within a factor $\tau < 1$ of the best result per instance. In the plot (Figure 7), the algorithm towards the top left corner is the best performer.

Our benchmark includes general hypergraphs that are primarily used for partitioning and social link hypergraphs generated from question-answering websites. In social link hypergraphs, a hypergraph matching can be used to summarize the overall websites, as it represents a subset of disjoint pages from different categories or users. In hypergraph

partitioning, matchings can be used to contract the hypergraph in a multilevel scheme. In a social hypergraph, each page (e.g., a post on StackOverflow or an article in Wikipedia) is considered as a hyperedge. The (Threads) graphs model participating users as vertices, whereas the (Tags) ones model the tags of the posts as vertices. We use three stackexchange networks by Benson et al. [3], and set the number of views as weights from [39]. For the StackOverflow instance, the weights are set by querying the public dataset from BigQuery [19]. We created an additional instance from this source. We also generated a new hypergraph from the English Wikipedia dump, where the categories represent the vertices and the articles represent the hyperedges. We selected a category as a vertex if it has at least 25 mentions, which resulted in 293K vertices for 8M articles. The access frequency of each article in December 2024 is set as weight. We answer **RQ1** and **RQ3** with this data set.

We use the hypergraph data set L_{HG} collected earlier [20] for hypergraph partitioning to address **RQ2** and **RQ3**. The set consists of 94 instances, spanning a wide range of applications from DAC routability-driven placement [41], general matrices [8] to SAT solving [2]. As weights we use $\max |e| - |e|$. This function maximizes the cardinality (number of edges matched). These instances have up to 1.4×10^8 hyperedges/-vertices and a maximum hyperedge size of 2.3×10^6 vertices. More statistics are in the Appendix of the full version.

For the stack-based approach of Section 3, we implemented the Guarantee and Lenient update functions. For simplicity, we refer to the algorithm with the stricter Guarantee function as Stack and the one with the lenient function as StackLenient. Both approaches can be configured by their ε parameter. The algorithms from Section 4 are named SwapSet. The SwapSet algorithm has a parameter $\alpha \geq 0$. For comparison, we use a non-streaming Greedy algorithm that sorts the hyperedges based on weight in descending order and greedily adds them to a matching. We also implemented a Naive streaming algorithm that maintains a maximal hypergraph matching (i.e, it includes a hyperedge if it is feasible w.r.t the current matching) in the order that they are streamed.

5.2 Impact of Streaming Order

We now investigate the impact of different streaming orders on our algorithms (**RQ1**). To this end, we stream the hyperedges of the social-link hypergraphs in three ways: by ascending weight, descending weight, and the original input order. The original input order is the order given by either the original files [3] or, in the case of the wikipedia instance, the order that the articles appear in the dump.

Memory. Figure 2 shows the geometric mean memory consumption for all approaches grouped by the ordering. Interestingly, for lower values of ε, the memory consumption is significantly higher for the ascending order, up to 2.97 times over the descending order (StackLenient $\varepsilon = 0$). For the swapping based algorithms (SwapSet) the differences between the orderings are only minimal, nearly reaching the memory consumption of the Naive algorithm. The Greedy algorithm utilizes the same amount of memory for all orderings, 13.43 times more memory than the stack approaches in the geometric mean. The ordering heavily affects our stack-based algorithms, but they still require less memory than the Greedy approach. The SwapSet algorithms are not significantly impacted by the order. Both results are in line with the theoretical results derived in Section 3.3 and 4.

Running time. In Figure 3 the geometric mean of the running times of the compared algorithms are shown. The Naive streaming algorithm is the fastest, requiring similar time over every ordering. The ascending ordering requires more time for both greedy swapping and

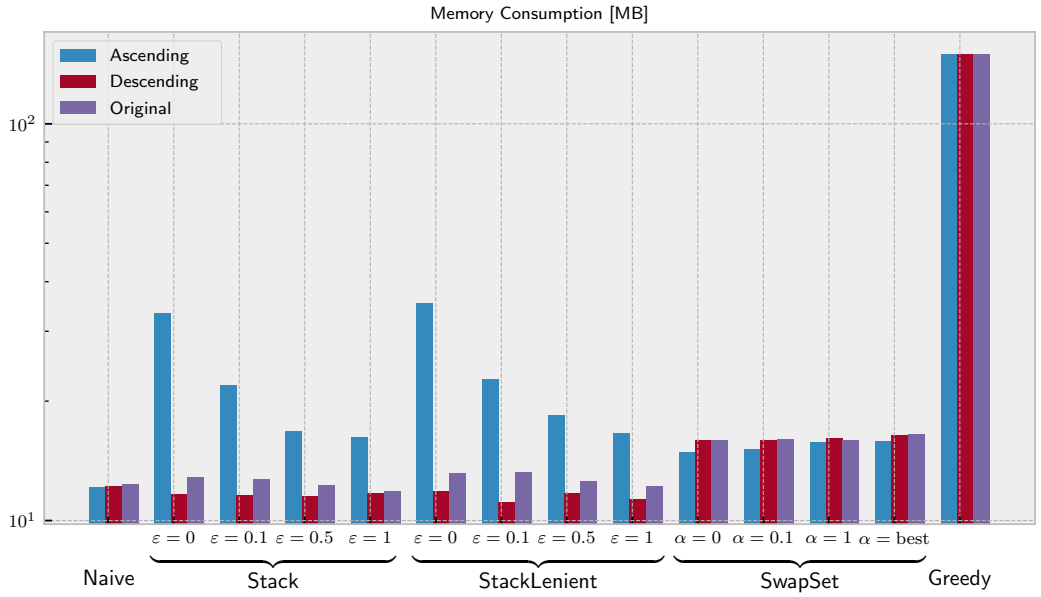


Figure 2 Geometric mean of the memory consumption on the social-link hypergraphs. Note the log-scale on the y-axis.

stack-based approaches since more replacements in the swapping algorithm and placements in the stack happen in that order. The stack-based approaches have a running time only 26% higher than the Naive. We observe that preordering the hyperedges speeds up the offline Greedy algorithm by a factor of 2.73 in comparison to the original ordering, but it is still slower than the stack-based algorithms. This is due to our use of std::sort that is partly optimized for ordered data. The running time of the SwapSet algorithms is comparable to the running time of Greedy with natural ordering. The cost for voiding edges in the SwapSet algorithm is higher than simply pushing edges on the stack. In general, the streaming process of Stack and StackLenient is similar to that of Naive. The small overhead is rooted in the required pass through the stack to build the matching. The ordering affects how many edges are added to the stack (see the following Memory section) and so the running time as well, but not as much as in the Greedy algorithm.

Quality. We present the geometric mean of the weights of matchings in Figure 4. As expected, the Greedy algorithm is not affected by the ordering since it computes its own order in the beginning. The quality of Naive algorithm is heavily affected by orderings, where the best weight is achieved when the hyperedges are streamed in descending order. On ascending and original order, the quality of the Naive is significantly worse. Under the descending orderings, the swapping algorithm (SwapSet) produces the same results as the Greedy algorithm, whereas the ascending order produces worse quality results in the SwapSet algorithm. The Stack and StackLenient approaches are more robust under different orderings and compute better quality matchings in the adversarial ascending order compared to other algorithms..

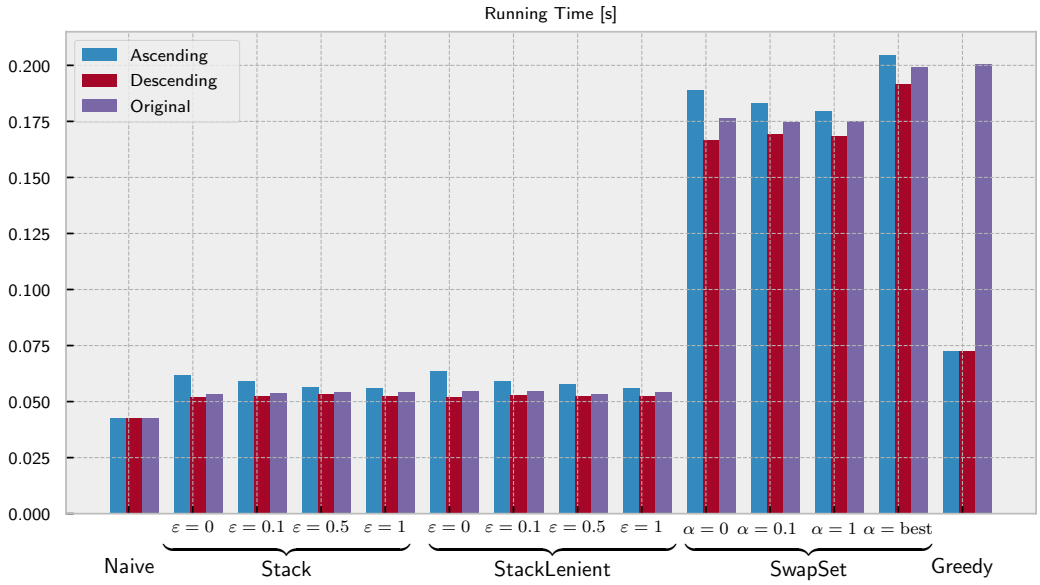


Figure 3 Geometric mean of running times on the social-link hypergraphs.

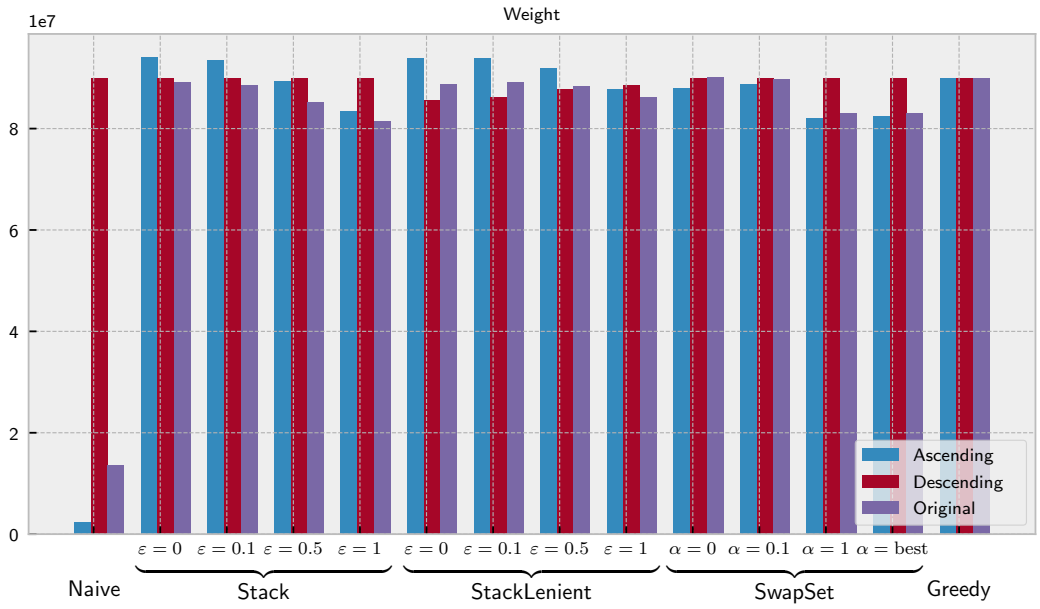


Figure 4 Geometric mean of weights on the social-link hypergraphs.

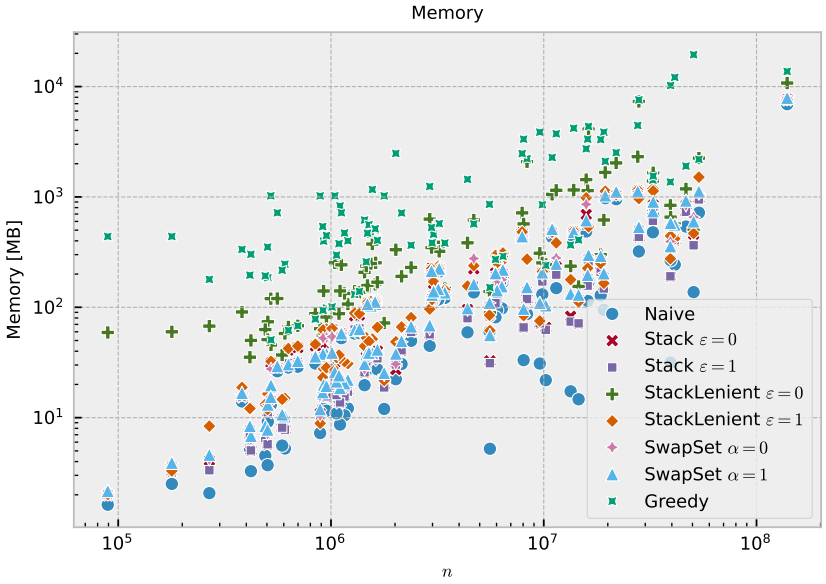


Figure 5 Geometric mean of the memory consumption on hypergraphs from partitioning plotted against the number of vertices.

5.3 Impact of Other Properties

We now test whether some other properties of the instances have an impact on the memory consumption of the algorithm (**RQ2**). Namely, we check if the number of vertices and the number of pins (sum of all edge sizes) are correlated with the memory consumption. We use the data set by Gottesbüren et al. [20] from hypergraph partitioning, which contains many larger-scale instances. This data set contains 94 diverse instances, making it more suitable for statistical testing. The ordering is the original ordering as given by the files in [20]. Instance details can be found in the Appendix of the full version. A hyperedge e in these instances has a weight of $(\max_{e \in E} |e|) - |e|$, optimizing the number of hyperedges matched. We set ε (and also α) to $\{0, 1\}$ since they represent the two extremes in terms of memory usage.

Number of Vertices. Figure 5 shows the memory consumption plotted against the number of vertices in each instance. The plot is on a log-log-scale. Naturally, Naive requires the least amount of memory. Our SwapSet algorithm follows, as well as the Stack algorithm and the StackLenient. Finally, the offline Greedy requires even more memory. This is also backed by the theoretical results of Sections 3.3 and 4. The higher the number of vertices, the smaller is the difference in magnitudes between the approaches. This is due to the additional memory needed for the list of finally matched hyperedges and the overhead in some allocations. The measured peak allocation may contain some overallocated memory caused by the growing result vectors.

Number of Pins. In Figure 6, a plot of the geometric mean memory consumption against the number of pins (sum of all edge sizes) for our approaches and the competitors is shown. Additionally, we added a linear regression line for the Greedy algorithm's memory consumption. The linear model achieves an R^2 score of 0.96 when (randomly) splitting the data set into a

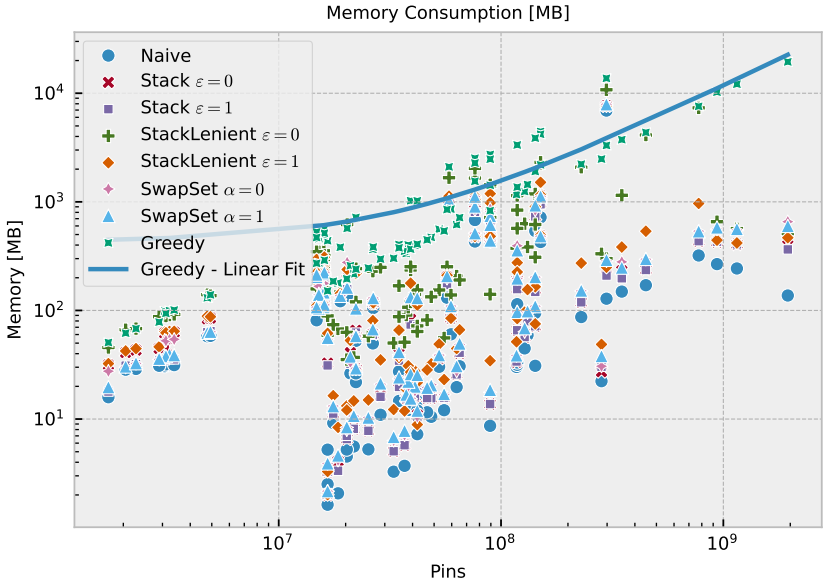


Figure 6 Geometric mean of the memory consumption on the hypergraphs from partitioning. Linear Regression line for the offline Greedy algorithm.

training ($n = 75$) and evaluation test set ($n = 19$), on the latter. This shows that the greedy algorithm requires memory linear in the number of pins since it loads the whole hypergraph at the start. Our proposed algorithms require less memory, especially the SwapSet algorithms nearly match the memory consumption of the Naive approach.

Other Properties. We also tested the dependence of the memory consumption on the average hyperedge size and the number of hyperedges, but found no pattern. Note that the number of pins is equal to the product of the number of edges and the average hyperedge size.

5.4 Comparison with Offline Greedy and Naive Streaming

In this section, we compare the algorithms on the instances stemming from hypergraph partitioning (**RQ3**). The Results on the social set of instances can be found in Section 5.2. Figure 7 shows a performance profile for the cardinality/size of the matchings for our algorithms as well as the Naive streaming and offline Greedy algorithm. The StackLenient variant with $\varepsilon = 0$ computes the biggest matchings and is the best performing algorithm, followed by SwapSet $\alpha = 0$. The Greedy algorithm has similar results to StackLenient $\varepsilon = 1$. The Stack $\varepsilon = 1$ and SwapSet $\alpha = 1$ are our worst-performing algorithms, having results very close to the Naive streaming algorithm. Their higher parameters ($\alpha = \varepsilon = 1$) cause the algorithm to converge to the naive algorithm.

6 Conclusion

We have proposed two (semi-)streaming algorithms for hypergraph matching. The first, inspired by Paz-Schwartzman [33], uses a stack and adds hyperedges to it, based on dual variables it keeps track of and updates according to an update function. The approximation

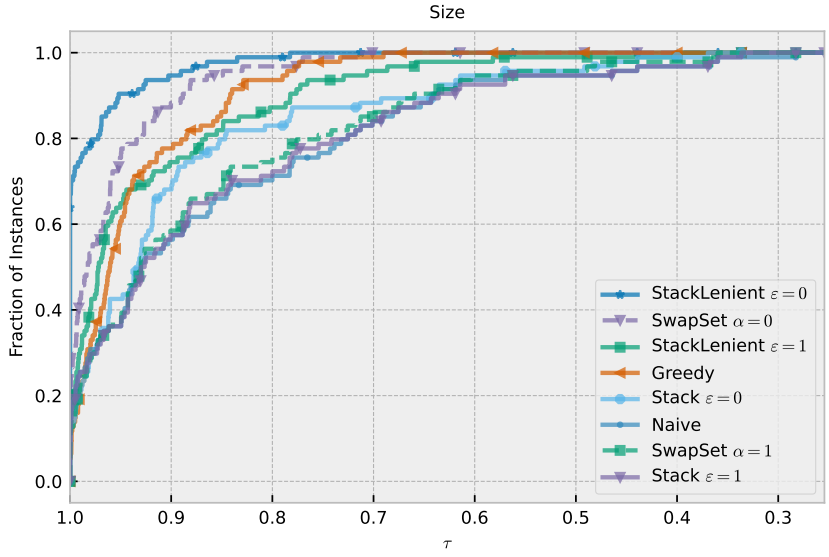


Figure 7 Performance Profile for the size of the matching in hypergraphs used in partitioning.

guarantee for this algorithm is $\frac{1}{d(1+\varepsilon)}$, and its running time per hyperedge is linear in hyperedge size. We have proposed two other update functions to be used in this algorithm. The proposed update functions result in a space complexity of $\mathcal{O}((1/\varepsilon)n\log^2 n)$ and $\mathcal{O}((1/\varepsilon)nd\log^2 n)$ bits. The second proposed algorithm works by greedily swapping out hyperedges and maintaining only one solution, requiring only $\mathcal{O}(n)$ memory. Inspired by McGregor's $\frac{1}{3+2\sqrt{2}}$-approximation guarantee [31], we have shown that if every swap increases the weight by at least $(1+\alpha)$, the algorithm has an approximation guarantee of $\frac{1}{(1+\alpha)\left(\frac{d-1}{\alpha}+d\right)}$. The best choice is $\alpha = \sqrt{(d-1)/d}$.

In extensive experiments, we have shown the competitiveness of the proposed algorithms in comparison to the standard non-streaming Greedy and a Naive streaming approach with respect to running time, memory consumption, and quality. We showed that the order of the hyperedges in the stream directly impacts the solution quality and that our stack algorithms handle worst-case orderings (like ascending weights for the Naive algorithm) even better than the offline Greedy algorithm. The running times of our Stack algorithm are only 26% higher than the Naive algorithm. Furthermore, we validated that the memory consumption for both families of algorithms is not linear in the number of pins, as it is for the Greedy algorithm; for the Stack approaches on the social link hypergraphs it is 13 times lower than Greedy. Lastly, we report that our algorithm obtains considerably better cardinality matchings in general hypergraphs from partitioning tasks.

Avenues of future work include improving the solution quality and extending to problems with relaxed capacity constraints. We aim to develop a streaming algorithm that can efficiently handle instances with capacity $b(v) > 1$ at each vertex, while maintaining a reasonable approximation ratio and computational overhead.

References

1 Mokhtar S Bazaraa, John J Jarvis, and Hanif D Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, 2011.

2 Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo. The SAT competition 2014. `http://www.satcompetition.org/2014/index.shtml`, 2014.

3 Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon M. Kleinberg. Simplicial closure and higher-order link prediction. *Proc. Natl. Acad. Sci. USA*, 115(48):E11221–E11230, 2018. doi:10.1073/pnas.1800683115.

4 Piotr Berman. A $d/2$ approximation for maximum weight independent set in d-claw free graphs. In Magnús M. Halldórsson, editor, *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, volume 1851 of *Lecture Notes in Computer Science*, pages 214–219, Berlin, Heidelberg, 2000. Springer. doi:10.1007/3-540-44985-X_19.

5 B. Besser and M. Poloczek. Greedy matching: Guarantees and limitations. *Algorithmica*, 77:201–234, 2017. doi:10.1007/S00453-015-0062-2.

6 Marcel Birn, Vitaly Osipov, Peter Sanders, Christian Schulz, and Nodari Sitchinava. Efficient parallel and external matching. In Felix Wolf, Bernd Mohr, and Dieter an Mey, editors, *Euro-Par 2013 Parallel Processing - 19th International Conference, Aachen, Germany, August 26-30, 2013. Proceedings*, volume 8097 of *Lecture Notes in Computer Science*, pages 659–670. Springer, Springer, 2013. doi:10.1007/978-3-642-40047-6_66.

7 Marek Cygan. Improved approximation for 3-dimensional matching via bounded pathwidth local search. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 509–518. IEEE, IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.61.

8 Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011. doi:10.1145/2049662.2049663.

9 Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2):201–213, 2002. doi:10.1007/s101070100263.

10 Doratha E. Drake and Stefan Hougardy. A simple approximation algorithm for the weighted matching problem. *Inf. Process. Lett.*, 85(4):211–213, 2003. doi:10.1016/S0020-0190(02)00393-9.

11 Fanny Dufossé, Kamer Kaya, Ioannis Panagiotas, and Bora Uçar. Effective heuristics for matchings in hypergraphs. In Ilias S. Kotsireas, Panos M. Pardalos, Konstantinos E. Parsopoulos, Dimitris Souravlias, and Arsenis Tsokas, editors, *Analysis of Experimental Algorithms - Special Event, SEA^2 2019, Kalamata, Greece, June 24-29, 2019, Revised Selected Papers*, volume 11544 of *Lecture Notes in Computer Science*, pages 248–264. Springer, Springer, 2019. doi:10.1007/978-3-030-34029-2_17.

12 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

13 Jason Evans. jemalloc. http://jemalloc.net/, 2006–. Accessed: 2025-04-15.

14 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Departmental Papers (CIS)*, page 236, 2005. doi:10.1016/j.tcs.2005.09.013.

15 S. M. Ferdous, Alex Pothen, and Mahantesh Halappanavar. Streaming matching and edge cover in practice. In Leo Liberti, editor, *22nd International Symposium on Experimental Algorithms, SEA 2024, July 23-26, 2024, Vienna, Austria*, volume 301 of *LIPIcs*, pages 12:1–12:22, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SEA.2024.12.

16 S. M. Ferdous, Bhargav Samineni, Alex Pothen, Mahantesh Halappanavar, and Bala Krishnamoorthy. Semi-streaming algorithms for weighted k-disjoint matchings. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 53:1–53:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2024.53.

17 Aurélien Froger, Olivier Guyon, and Eric Pinson. A set packing approach for scheduling passenger train drivers: the French experience. In *RailTokyo2015*, 2015.

18 Mohsen Ghaffari and David Wajc. Simplified and space-optimal semi-streaming (2+epsilon)-approximate matching. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 13:1–13:8, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.SOSA.2019.13.

19 Google Cloud Platform. Bigquery public datasets. https://cloud.google.com/bigquery/public-data. Accessed: 2025-04-19.

20 Lars Gottesbüren, Tobias Heuer, Nikolai Maas, Peter Sanders, and Sebastian Schlag. Scalable high-quality hypergraph partitioning. *CoRR*, abs/2303.17679, 2023. doi:10.48550/arXiv.2303.17679.

21 Georg Gottlob and Gianluigi Greco. Decomposing combinatorial auctions and set packing problems. *J. ACM*, 60(4):24:1–24:39, 2013. doi:10.1145/2508028.2505987.

22 Ernestine Großmann, Felix Joos, Henrik Reinstädtler, and Christian Schulz. Engineering hypergraph b-matching algorithms. *arXiv preprint arXiv:2408.06924*, 2024. doi:10.48550/arXiv.2408.06924.

23 Oussama Hanguir and Clifford Stein. Distributed algorithms for matching in hypergraphs. In Christos Kaklamanis and Asaf Levin, editors, *Approximation and Online Algorithms - 18th International Workshop, WAOA 2020, Virtual Event, September 9-10, 2020, Revised Selected Papers*, volume 12806 of *Lecture Notes in Computer Science*, pages 30–46. Springer, Springer, 2020. doi:10.1007/978-3-030-80879-2_3.

24 Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. *Comput. Complex.*, 15(1):20–39, 2006. doi:10.1007/s00037-006-0205-6.

25 Chien-Chung Huang and François Sellier. Semi-streaming algorithms for submodular function maximization under b-matching constraint. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 14:1–14:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.14.

26 Cor A. J. Hurkens and Alexander Schrijver. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM J. Discret. Math.*, 2(1):68–72, 1989. doi:10.1137/0402008.

27 Richard M. Karp. Reducibility among combinatorial problems. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2010. doi:10.1007/978-3-540-68279-0_8.

28 B. Korte and D. Hausmann. An analysis of the greedy algorithm for independence systems. *Annals of Discrete Mathematics*, 2:65–74, 1978.

29 Fredrik Manne and Mahantesh Halappanavar. New effective multithreaded matching algorithms. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 519–528. IEEE Computer Society, 2014. doi:10.1109/IPDPS.2014.61.

30 Jens Maue and Peter Sanders. Engineering algorithms for approximate weighted matching. In Camil Demetrescu, editor, *Experimental Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007, Proceedings*, volume 4525 of *Lecture Notes in Computer Science*, pages 242–255. Springer, Springer, 2007. doi:10.1007/978-3-540-72845-0_19.

31 Andrew McGregor. Finding graph matchings in data streams. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 170–181. Springer, Springer, 2005. doi:10.1007/11538462_15.

32 Meike Neuwohner. Passing the limits of pure local search for weighted k-set packing. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1090–1137. SIAM, SIAM, 2023. doi:10.1137/1.9781611977554.ch41.

33 Ami Paz and Gregory Schwartzman. A (2+ϵ)-approximation for maximum weight matching in the semi-streaming model. *ACM Trans. Algorithms*, 15(2):18:1–18:15, December 2019. doi:10.1145/3274668.

34 Seth Pettie and Peter Sanders. A simpler linear time 2/3-ε approximation for maximum weight matching. *Information Processing Letters*, 91(6):271–276, 2004. doi:10.1016/j.ipl.2004.05.007.

35 Alex Pothen, SM Ferdous, and Fredrik Manne. Approximation algorithms in combinatorial scientific computing. *Acta Numerica*, 28:541–633, 2019. doi:10.1017/S0962492919000035.

36 Robert Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In Christoph Meinel and Sophie Tison, editors, *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*, volume 1563 of *Lecture Notes in Computer Science*, pages 259–269. Springer, Springer, 1999. doi:10.1007/3-540-49116-3_24.

37 Henrik Reinstädtler, S M Ferdous, Alex Pothen, Bora Uçar, and Christian Schulz. HeiHGM/Streaming. Software, DFG-SCHU 2567/8-1, swhId: swh:1:dir:4e4522550296fb38202457ce7371bf7034ae45d9 (visited on 2025-09-05). URL: https://github.com/HeiHGM/Streaming, doi:10.4230/artifacts.24674.

38 Henrik Reinstädtler, S M Ferdous, Alex Pothen, Bora Uçar, and Christian Schulz. Semi-streaming algorithms for hypergraph matching, 2025. doi:10.48550/arXiv.2502.13636.

39 StackExchange. Stackexchange data explorer. https://data.stackexchange.com/. Accessed: 2025-04-19.

40 Thorben Tröbst and Rajan Udwani. Almost tight bounds for online hypergraph matching. *arXiv preprint arXiv:2402.08775*, 2024. doi:10.1016/j.orl.2024.107143.

41 Natarajan Viswanathan, Charles J. Alpert, Cliff C. N. Sze, Zhuo Li, and Yaoguang Wei. The DAC 2012 routability-driven placement contest and benchmark suite. In Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun, editors, *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012*, DAC '12, pages 774–782, New York, NY, USA, 2012. ACM. doi:10.1145/2228360.2228500.

42 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

43 Laurence A Wolsey. *Integer Programming*. John Wiley & Sons, 2020.

44 Stephen J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997. doi:10.1137/1.9781611971453.

Online Metric TSP

Christian Bertram
University of Copenhagen, Denmark

Abstract

In the *online metric traveling salesperson problem*, n points of a metric space arrive one by one and have to be placed (immediately and irrevocably) into empty cells of a size-n array. The goal is to minimize the sum of distances between consecutive points in the array. This problem was introduced by Abrahamsen, Bercea, Beretta, Klausen, and Kozma [ESA'24] as a generalization of the *online sorting problem*, which was introduced by Aamand, Abrahamsen, Beretta, and Kleist [SODA'23] as a tool in their study of online geometric packing problems.

Online metric TSP has been studied for a range of fixed metric spaces. For 1-dimensional Euclidean space, the problem is equivalent to online sorting, where an optimal competitive ratio of $\Theta(\sqrt{n})$ is known. For d-dimensional Euclidean space, the best-known upper bound is $O(2^d\sqrt{dn\log n})$, leaving a gap to the $\Omega(\sqrt{n})$ lower bound. Finally, for the uniform metric, where all distances are 0 or 1, the optimal competitive ratio is known to be $\Theta(\log n)$.

We study the problem for a general metric space, presenting an algorithm with competitive ratio $O(\sqrt{n})$. In particular, we close the gap for d-dimensional Euclidean space, completely removing the dependence on dimension. One might hope to simultaneously guarantee competitive ratio $O(\sqrt{n})$ in general and $O(\log n)$ for the uniform metric, but we show that this is impossible.

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

Keywords and phrases online algorithm, metric space, TSP

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.80

Related Version *Full Version*: `https://arxiv.org/abs/2504.17716` [4]

Funding Christian Bertram is part of BARC, Basic Algorithms Research Copenhagen, supported by the VILLUM Foundation grant 54451.

Acknowledgements I thank Anders Aamand, Mikkel Abrahamsen, Théo Fabris, Jonas Klausen, and Mikkel Thorup for helpful and inspiring conversations, and the reviewers for valuable comments.

1 Introduction

1.1 Problem definition

The *online metric traveling salesperson problem* (online metric TSP), recently introduced by Abrahamsen, Bercea, Beretta, Klausen, and Kozma [2], is as follows. Given a sequence $x_1, \ldots, x_n$ of points arriving one by one (with repetitions allowed) from a metric space (M, d), assign them bijectively to array cells $A[1], \ldots, A[n]$. The goal is to minimize $\sum_{i=1}^{n-1} d(A[i], A[i+1])$, which represents the length of the walk $A[1], \ldots, A[n]$. The problem is *online* in the sense that after receiving x_i, we must immediately and irrevocably set $A[j] = x_i$ for some previously unused array index j, without knowledge of x_k for $k > i$.

We can think of this as a metric traveling salesperson problem, where n cities are sequentially revealed, one by one, and must be placed on a unique date in the salesperson's n-day calendar. The cost to be minimized is the length of the final n-day trip.

In [2], the metric space is fixed as 1-dimensional Euclidean space, d-dimensional Euclidean space, or a space with uniform/discrete metric (where all distances are 0 or 1). In this paper, we study the problem for a general metric space, allowing the algorithm to query the distance $d(x_i, x_j)$ between x_i and x_j, if it has received x_i and x_j. Alternatively, the ith input can be assumed to be the vector $(d(x_i, x_1), d(x_i, x_2), \ldots, d(x_i, x_{i-1}))$.

© Christian Bertram;
licensed under Creative Commons License CC-BY 4.0

The term "online TSP" has also been used for a different, older problem where a salesperson moves through the metric space at unit speed as cities are revealed [3]. In contrast, our problem consists of constructing a fixed travel plan through irrevocable online decisions.

1.2 Prior work

In [1], the *online sorting problem* is introduced as part of their study of various geometric translational packing problems. The online sorting problem is equivalent to online (metric) TSP in the Euclidean unit interval $[0, 1]$. They further essentially assume that the points 0 and 1 always show up in the input, which simplifies the analysis, as the optimal offline cost becomes 1. They present a deterministic algorithm with competitive ratio $O(\sqrt{n})$ for online sorting (where *competitive ratio* is the worst-case ratio between the algorithm's cost and the optimal offline cost), and show an $\Omega(\sqrt{n})$ lower bound for deterministic algorithms.

In [2], the online sorting algorithm from [1] is generalized to points from the real line $\mathbb{R}$, not necessarily including 0 and 1, by employing a careful *doubling technique*. They maintain the $O(\sqrt{n})$ competitive ratio, and generalize the $\Omega(\sqrt{n})$ lower bound to randomized algorithms.

In [2], they further generalize the problem to online metric TSP, and consider this problem for a few fixed metric spaces. For d-dimensional Euclidean space, they present an $O(2^d\sqrt{dn\log n})$ competitive ratio algorithm, leaving a gap to their $\Omega(\sqrt{n})$ lower bound. For use as a subroutine in this algorithm, they consider online TSP in the uniform metric, i.e. the metric where all distances are 0 or 1. They argue that this problem has interest in its own right, corresponding to minimizing the number of task switches in scheduling, and present a tight $\Theta(\log n)$ bound on the competitive ratio. Finally, they ask, as an open question, whether $O(\sqrt{n})$ is the optimal competitive ratio for arbitrary metrics.

1.3 Our results

Our main result is an optimal algorithm for online metric TSP in a general metric space, settling a question posed in [2].

▶ **Theorem 1.** *There exists a deterministic algorithm for online metric TSP with competitive ratio $O(\sqrt{n})$.*

This is optimal by the $\Omega(\sqrt{n})$ lower bound for the Euclidean unit interval [1], which holds even for randomized algorithms [2]. As a direct corollary, we get an optimal algorithm for online TSP in d-dimensional Euclidean space, improving upon the best-known $O(2^d\sqrt{dn\log n})$ competitive ratio algorithm [2], completely removing the dependence on dimension.

▶ **Corollary 2.** *There exists a deterministic algorithm for online TSP in d-dimensional Euclidean space with competitive ratio $O(\sqrt{n})$.*

Note, though, that the $O(\sqrt{n})$ bound of Theorem 1 does not match the known $\Theta(\log n)$ bound for online TSP with uniform metric [2]. One might hope for an even stronger algorithm, obtaining an asymptotically optimal competitive ratio for every fixed metric space. We show that no such algorithm exists, hinting that our algorithm is the best one can hope for.

▶ **Theorem 3.** *No randomized algorithm for online metric TSP obtains both $O(\log n)$ expected competitive ratio for the uniform metric and $O(\sqrt{n})$ expected competitive ratio in general.*

1.4 Structure of the paper

In Section 2, we recall relevant definitions and fix notation. In Section 3, we present our algorithm, proving Theorem 1 and Corollary 2. In Section 4, we confirm the worst-case optimality of our algorithm via known lower bounds, and then we show that no algorithm is simultaneously optimal for every fixed metric, proving Theorem 3.

2 Preliminaries

We will often let (M, d) be a metric space, where M is the set of points, and $d\colon M \times M \to \mathbb{R}_{\geq 0}$ is the metric. Our main examples are d-dimensional Euclidean space and the uniform metric.

The *uniform metric* is defined by $d(x, y) = 1$ for all $x \neq y$. This is also known as the *discrete metric*, but we adopt the former term for consistency with [2]. Any set of points can form a metric space under the uniform metric.

Though our algorithms run in polynomial time, we do not focus on their exact running times. Instead, we evaluate performance via *competitive analysis*, as is standard in the study of online algorithms. Let $\texttt{OPT}(X)$ denote the optimal offline cost for an input sequence $X \in M^n$. An online algorithm is said to have *competitive ratio* $C(n)$, for a function $C\colon \mathbb{N} \to \mathbb{R}$, if its cost is at most $C(n)\,\texttt{OPT}(X)$ for all $n \in \mathbb{N}$ and all $X \in M^n$.

In our setting, the cost is $\sum_{i=1}^{n-1} d(A[i], A[i+1])$, and $\texttt{OPT}(X)$ is the minimum value of this sum, minimized over all bijections from the input points in X to array cells $A[1], \ldots, A[n]$. By the triangle inequality, $\texttt{OPT}(X)$ is equal to the length of a shortest walk visiting all points in X. We will sometimes write $\texttt{OPT}(X)$ for a subset $X \subseteq M$, as the optimal offline cost is invariant under permutations and repetitions of the input points.

3 Our algorithm

Our algorithm is based on the algorithms for online sorting presented in [1] and [2]. Let us first give a high-level explanation of their approach. Denote by X' the set of currently received points. They consider an interval J containing X', and partition J into $\sqrt{n}$ subintervals. Specifically, they assume without loss of generality that $x_1 = 0$, and let J be the smallest interval of the form $[-2^k, 2^k]$ containing X', increasing k as necessary. Note that $\texttt{OPT}(X') \geq 2^{k-1}$, so the subintervals have size $O(\texttt{OPT}(X')/\sqrt{n})$. Similarly, they partition the array A into $2\sqrt{n}$ subarrays, called *blocks*. The algorithm generally places the input points into the array, such that no block contains points from different subintervals. This ensures, that they only pay a small $O(\texttt{OPT}(X')/\sqrt{n})$ cost between points inside each block, and pay a large $O(\texttt{OPT}(X'))$ cost only between blocks. The former cost is paid $O(n)$ times, and the latter $O(\sqrt{n})$ times, totaling a cost of $O(\sqrt{n}\,\texttt{OPT}(X'))$. It is always possible to place the first half of the points in this manner, after which they consider the remaining empty array cells as one contiguous array, and recursively fill this simulated array with the remaining input points. This leads to a total cost of $O(\sqrt{n}\,\texttt{OPT}(X))$ as the cost per recursion falls geometrically. Each time k is increased, we incur some additional cost of changing the intervals, but this only happens when the bound on $\texttt{OPT}(X')$ doubles, so again it all sums to $O(\sqrt{n}\,\texttt{OPT}(X))$.

Let us now give a high-level explanation of how we will extend this algorithm from the Euclidean line to a general metric space. There are two main problems to solve: the setting of $J = [-2^k, 2^k]$ and bound of $\texttt{OPT}(X') \geq 2^{k-1}$ no longer make sense. One might naively try letting J be a ball of radius 2^k, but this cannot generally be covered by $\sqrt{n}$ balls of radius $O(2^k/\sqrt{n})$. To see this, think of d-dimensional Euclidean space, where 2^d unit cubes are required to cover a cube of sidelength 2. This is related to why there is a term of size 2^d in

the best-known bound for d-dimensional Euclidean space [2]. We instead present an online algorithm maintaining a set of at most $\sqrt{n}$ balls of radius $r = \Theta(\texttt{OPT}(X')/\sqrt{n})$ covering all received input points. Only when $\texttt{OPT}(X')$ doubles, we allow changing the radius r and resetting the set. Of course, we don't know the value of $\texttt{OPT}(X')$, but it turns out that we can successfully use the minimum spanning tree weight as a (polynomial-time computable) proxy. This proxy will be essential when showing that our covering only needs $\sqrt{n}$ balls.

We begin the formal presentation of our algorithm with the above mentioned proxy.

▶ **Definition 4.** Let (M, d) be a metric space and $X \subseteq M$ a finite subset of points. When the metric d is clear from context, we denote by $\texttt{MST}(X)$ the total weight of a minimum spanning tree of the complete graph on X, where edge weights are given by d.

Computing exact metric TSP length is NP-hard, as follows by a straightforward reduction from the undirected Hamiltonian cycle problem, one of Karp's 21 NP-complete problems [5]. Luckily, there is a simple 2-approximation based on minimum spanning trees [6]. This well-known argument yields the following lemma, which tells us that $\texttt{MST}(X)$ is a good proxy of $\texttt{OPT}(X)$. We provide a proof in the full version of the paper [4].

▶ **Lemma 5.** *Let (M, d) be a metric space and $X \subseteq M$ a finite subset of points. Then $\texttt{MST}(X) \leq \texttt{OPT}(X) \leq 2\,\texttt{MST}(X)$.*

As sketched, our algorithm will cover the input points by a dynamic set of balls. Formally, we will work with a set of *centers* forming a *net* in the following standard manner.

▶ **Definition 6.** Let (M, d) be a metric space and $X \subseteq M$ a set of points. Let $r \geq 0$ be a real number. A subset $C \subseteq X$ is an *r-net* of X if

- for every $x \in X$, there exists a $c \in C$ with $d(c, x) \leq r$, and
- for every $c, c' \in C$ with $c \neq c'$, we have $d(c, c') > r$.

If so, we sometimes refer to each point in C as a *center*, and to r as the *radius* of the net.

Nets will be useful to us, because they are small in terms of our proxy, in the sense of the following lemma.

▶ **Lemma 7.** *Let (M, d) be a metric space, in which C is an r-net of a set $X \subseteq M$. Then $(|C| - 1)r \leq 2\,\texttt{MST}(X)$.*

Proof. By the triangle inequality, we have $\texttt{OPT}(X) \geq \texttt{OPT}(C)$. Combining this with Lemma 5, we get $2\,\texttt{MST}(X) \geq \texttt{OPT}(X) \geq \texttt{OPT}(C) \geq \texttt{MST}(C)$. Every edge in the complete graph on C has length at least r, and a tree on $|C|$ points has $|C| - 1$ edges, so $\texttt{MST}(C) \geq (|C| - 1)r$. ◀

When receiving a new point x, we will generally update our r-net by the following simple subroutine.

Increase-Net(C, r, x)
1. If no $c \in C$ has $d(x, c) \leq r$ then add x to C.

This subroutine allows us to maintain an r-net, for a fixed radius r. We state this as the following lemma, which follows directly from the definition.

▶ **Lemma 8.** *Let (M, d) be a metric space, in which C is an r-net of a set $X \subseteq M$, and let $x \in X$. Then running* Increase-Net(C, r, x) *modifies C only by insertion, such that C becomes an r-net of $X \cup \{x\}$.*

To facilitate analysis, let us introduce two pieces of notation. For a partially filled array A, we write $c(A) = \sum_{i \in I} d(A[i], A[i+1])$ for the *cost* of A, where I is the set of indices i for which both $A[i]$ and $A[i+1]$ are non-empty. Note that this matches the usual cost, when A is full. Secondly, we write $G(A)$ for the number of *gaps* in A, i.e. the number of non-trivial maximal contiguous empty subarrays of A.

We now present "half" of our algorithm, namely an algorithm handling the first $\lceil n/2 \rceil$ input points. First, we give a quick overview. The algorithm will partition the array A into $2\lfloor \sqrt{n} \rfloor$ blocks, and maintain an r-net C with at most $\lfloor \sqrt{n} \rfloor$ centers. If we get too many centers, we will reset the net with a larger radius. Each block may be *assigned* to a center, such that every center is assigned at most one block. Initially, every block is unassigned. Generally, if a newly received point lies within the ball of a center, it will be placed inside the block assigned to that center. The exact algorithm is as follows.

Fill-Most-Blocks(n, A, X)

1. Let $N_1 = \lfloor \sqrt{n} \rfloor$ and $N_2 = 2N_1$.
2. Partition A into N_2 subarrays, called *blocks*, of length at least $\lfloor n/N_2 \rfloor$.
3. Initialize $r = 0$ and $C = \emptyset$.
4. For each point x in the stream X:
   a. Increase-Net(C, r, x).
   b. If $|C| > N_1$:
      I. Unassign all blocks.
      II. Set $r = 4\,\mathtt{MST}(X')/N_1$, where X' is the set of known input points.
      III. Set $C = \{x\}$.
   c. Let $c \in C$ be a center with $d(x, c) \leq r$.
   d. If a full block B is assigned to c:
      I. Unassign B from c.
   e. If no block is assigned to c:
      I. Assign an unassigned non-full block to c.
   f. Let B be the block assigned to c.
   g. Place x in the left-most empty cell of B.

▶ **Lemma 9.** *Let (M, d) be a metric space and X be an online stream of $\lceil n/2 \rceil$ points in M. Let A be an empty array of length n. The deterministic algorithm* Fill-Most-Blocks(n, A, X) *irrevocably places each point from X in an empty cell of A, such that when all points have been placed, we have $G(A) \leq 2\sqrt{n}$ and $c(A) \leq 11\sqrt{n}\,\mathtt{OPT}(X)$.*

Proof. The following two claims show that the algorithm is well-defined.

▷ Claim. After step 4b, C is an r-net of a set containing x.

Proof. Clearly, the claim holds if step 4(b)III just ran, so let us show, that it also holds before and between these resets of the net. This follows by induction, where the base case is step 3 producing an r-net, and the induction step is step 4a with Lemma 8. ◁

▷ Claim. In step 4(e)I, there is always an unassigned non-full block to assign.

Proof. Assume for the sake of contradiction, that no unassigned non-full block exists. Then every block is either assigned or full. There are at most $|C| \leq N_1$ assigned blocks, so at least $N_2 - |C| \geq N_2 - N_1 = N_1$ blocks are full. Note that every assigned block contains at least one point, as right after assigning a block in step 4(e)I, it is given a point in step 4g. So at least

N_1 blocks contain at least $\lfloor n/N_2 \rfloor$ points, and the remaining blocks contain at least one point. This means that the total number of points in A is at least $N_1\lfloor n/N_2 \rfloor + N_1 \geq N_1 n/N_2 = n/2$. This contradicts the stream containing only $\lceil n/2 \rceil$ points, as in step 4(e)I the point x has yet to be placed. ◁

The following two claims show that the algorithm is correct.

▷ Claim. At termination, $G(A) \leq 2\sqrt{n}$.

Proof. Since every point is placed in the left-most empty cell of a block, there is at most one gap per block, so $G(A) \leq N_2 \leq 2\sqrt{n}$. ◁

▷ Claim. At termination, $c(A) \leq 11\sqrt{n}\,\texttt{OPT}(X)$.

Proof. The cost between two neighboring blocks is at most $\texttt{OPT}(X)$, so the total cost between neighboring blocks is at most $N_2\,\texttt{OPT}(X)$. It remains to bound the cost between neighboring cells inside a block. So let x and x' be points placed in neighboring cells inside a block B.

Consider first the case where the net was not reset (steps 4(b)I to 4(b)III) between x and x' being placed. Then B was assigned to the same center c with the same radius r both when x and x' were placed. Let X' be the set of points read right after both x and x' were placed. By the triangle inequality, $\texttt{OPT}(X') \leq \texttt{OPT}(X)$, so by Lemma 5, $d(x,x') \leq d(x,c) + d(c,x') \leq 2r = 8\,\texttt{MST}(X')/N_1 \leq 8\,\texttt{OPT}(X')/N_1 \leq 8\,\texttt{OPT}(X)/N_1$. Since at most $\lceil n/2 \rceil$ points are placed, the total cost between such pairs of points is at most $(\lceil n/2 \rceil - 1)8\,\texttt{OPT}(X)/N_1 \leq 4n\,\texttt{OPT}(X)/N_1$.

Consider now the case where the net was reset between x and x' being placed. Then $d(x,x') \leq \texttt{MST}(X')$ where X' is the set of points read at any point after placing both x and x'. This is a worse bound, but this case can only appear once per block per reset of the net. So the total cost for such pairs of points is at most $N_2\,\texttt{MST}(X')$ for each reset, where X' is the set of points read when resetting. Let X_1 be the set of points read at the point of a reset, or at the beginning of the algorithm, and X_2 be the set of points read at the point of the following reset or when the algorithm terminates. Right before the latter reset, $|C| > N_1$ and $r = 4\,\texttt{MST}(X_1)/N_1$. This is true even if X_1 is the set at the beginning of the algorithm, as then $X_1 = \emptyset$ and $r = 0$. So Lemma 7 gives us that $2\,\texttt{MST}(X_2) \geq (|C| - 1)r \geq N_1 4\,\texttt{MST}(X_1)/N_1 = 4\,\texttt{MST}(X_1)$, which simplifies to $\texttt{MST}(X_2) \geq 2\,\texttt{MST}(X_1)$. The total cost of this type across all rebuilds is thus no more than the geometric series $N_2 \sum_{i=0} 2^{-i}\,\texttt{MST}(X) = 2N_2\,\texttt{MST}(X) \leq 2N_2\,\texttt{OPT}(X)$, where the last inequality uses Lemma 5.

The total cost $c(A)$ at termination is thus at most $(N_2 + 4n/N_1 + 2N_2)\,\texttt{OPT}(X)$. It can be checked that $N_2 + 4n/N_1 + 2N_2 \leq 11\sqrt{n}$, finishing the proof. ◁

The combination of the above claims finishes the proof. ◀

We are now ready to present our full algorithm, which recursively applies Fill-Most-Blocks, analogously to [1, 2]. The algorithm begins by placing the first half of the input points into the array A using Fill-Most-Blocks. It then treats the remaining empty cells of A as a contiguous array A_{empty}. The algorithm proceeds recursively on A_{empty}, treating it as a standard array; however, when a point is placed in A_{empty}, it is actually placed into the corresponding cell of A. The exact algorithm is given below as Recursively-Fill-Most-Blocks.

▶ **Theorem 10.** *Let (M,d) be a metric space and X be an online stream of n points in M. Let A be an empty array of length n. The deterministic algorithm* Recursively-Fill-Most-Blocks(n,A,X) *irrevocably places each point from X in an empty cell of A, such that when all points have been placed, we have $c(A) \leq 52\sqrt{n}\,\texttt{OPT}(X)$.*

Recursively-Fill-Most-Blocks(n, A, X)
1. If $n = 0$ then return.
2. Let X_{prefix} be the stream consisting the first $\lceil n/2 \rceil$ points of X.
3. Fill-Most-Blocks$(n, A, X_{\text{prefix}})$.
4. Consider the empty cells of A as one contiguous array A_{empty}.
5. Let X_{suffix} be the stream consisting the remaining $\lfloor n/2 \rfloor$ points of X.
6. Recursively-Fill-Most-Blocks$(\lfloor n/2 \rfloor, A_{\text{empty}}, X_{\text{suffix}})$.

Proof. We will show a cost of at most $15(2+\sqrt{2})\sqrt{n}\,\texttt{OPT}(X)$ using strong induction on n. The base case $n = 0$ is trivial, so let us handle the inductive step. Let A' denote the array A after step 3. Then $G(A') \leq 2\sqrt{n}$ and $c(A') \leq 11\sqrt{n}\,\texttt{OPT}(X_{\text{prefix}}) \leq 11\sqrt{n}\,\texttt{OPT}(X)$ by Lemma 9 and the triangle inequality. By induction, we have $c(A_{\text{empty}}) \leq 15(2+\sqrt{2})\sqrt{n/2}\,\texttt{OPT}(X)$. The final cost $c(A)$ is the sum of $c(A')$, $c(A_{\text{empty}})$, and the costs between neighboring cells in A where exactly one of the cells was empty in A'. The latter is at most $2G(A')\,\texttt{OPT}(X) \leq 4\sqrt{n}\,\texttt{OPT}(X)$, as there are at most two such pairs of neighboring cells per gap in A'. The total cost becomes $c(A) \leq (11 + 15(2+\sqrt{2})/\sqrt{2} + 4)\sqrt{n}\,\texttt{OPT}(X) = 15(2+\sqrt{2})\,\texttt{OPT}(X)$. ◀

From Theorem 10, we immediately get Theorem 1 and Corollary 2. We have thus generalized the optimal upper bound for online sorting [2] to online metric TSP, in particular improving the best-known upper bound for online TSP in d-dimensional Euclidean space. In Section 4 we show, that our algorithm is optimal for both general online metric TSP, online TSP in d-dimensional Euclidean space, and more.

4 Optimality

It follows directly from a known lower bound for online sorting [1, 2] that our algorithm is optimal. Recall that online sorting is equivalent to online TSP in 1-dimensional Euclidean space.

▶ **Theorem 11** (Theorem 1 in [2]). *The (deterministic and randomized) competitive ratio of online TSP in the Euclidean unit interval* $[0, 1]$ *is* $\Omega(\sqrt{n})$.

In particular, this gives a lower bound of $\Omega(\sqrt{n})$ for general online metric TSP, showing that our $O(\sqrt{n})$ algorithm Recursively-Fill-Most-Blocks from Section 3 is optimal. Perhaps surprisingly, online metric TSP is no harder than online sorting.

Since the Euclidean unit interval lies inside d-dimensional Euclidean space, Theorem 11 also shows that our $O(\sqrt{n})$ algorithm is optimal for online TSP in d-dimensional Euclidean space. We have thus closed the gap for this problem, improving upon the best known upper bound of $O(2^d\sqrt{dn\log n})$ [2], notably removing the dependence on dimension.

Studying the proof of the above lower bound, we find the following generalization. Intuitively, the generalization gives us a lower bound of $\Omega(\sqrt{n})$ for online TSP in any metric space where we can draw a straight line segment (of length ℓ with endpoints a_0 and a_1) and pick m evenly spaced points along it.

▶ **Corollary 12.** *Let* (M, d) *be a metric space, such that for every* $m \in \mathbb{N}$*, there exist two points* $a_0, a_1 \in M$*, and a set* $X \subseteq M$ *of* m *points, such that, for* $\ell = \texttt{OPT}(X \cup \{a_0, a_1\})$*,*
- $d(x, y) \geq \ell/m$ *for all distinct* $x, y \in X$*, and*
- $d(a_0, x) + d(x, a_1) \geq \ell$ *for all* $x \in X$*.*

Then the competitive ratio of online TSP in (M, d) *is* $\Omega(\sqrt{n})$.

Proof. This follows from the proof of Theorem 11 presented in Section 2 of [2]. They work in the Euclidean unit interval, but only use the points $a_0 = 0$, $a_1 = 1$, and the set of points $X = \{0, 1/\sqrt{n}, \ldots, (\sqrt{n}-1)/\sqrt{n}\}$. Ignoring the concrete values of the points a_0, a_1, and those in X, it can easily be checked, that they only use the properties stated above, where $m = \sqrt{n}$ and $\ell = 1$. The proof also follows through for any other $\ell > 0$, where all distances in their proof simply scale by ℓ. ◀

This tells us that our algorithm is also optimal for online TSP in e.g. (subgroups of) normed vector spaces and Riemannian manifolds.

4.1 Impossibility of optimality for every fixed metric space

We have seen, that our $O(\sqrt{n})$ algorithm is optimal for general online metric TSP, as well as for online TSP in many fixed metric spaces. It is not optimal for every fixed metric space, though, as an algorithm with competitive ratio $O(\log n)$ is known for the uniform metric [2]. Motivated by this gap, one might ask whether there exists a stronger algorithm, which optimally solves online TSP in (M, d), for every fixed metric space (M, d). In this subsection, we show that no such algorithm exists. This hints, that the weaker optimality of our algorithm is the best one can hope for.

Specifically, we show that no algorithm is optimal both for the uniform metric and for a general metric space. This is the content of Theorem 3, which we restate and prove below.

▶ **Theorem 3.** *No randomized algorithm for online metric TSP obtains both $O(\log n)$ expected competitive ratio for the uniform metric and $O(\sqrt{n})$ expected competitive ratio in general.*

Proof. Assume for the sake of contradiction that such an algorithm $\mathcal{A}$ exists. Let us first informally explain our basic idea. Let U be a set of $n^{4/5}$ points with pairwise distance 1, and let x be a point with distance $n^{4/5}$ to every point in U. Consider an input consisting $n^{1/5}$ consecutive copies of U. Then the optimal offline cost is $\texttt{OPT}(U) = |U| - 1 = n^{4/5} - 1$. The only way for $\mathcal{A}$ to match this optimal cost, would be to maintain $\Omega(|U|) = \Omega(n^{4/5})$ gaps until completion. Similarly, since $\mathcal{A}$ actually must obtain nearly-optimal cost $O(n^{4/5} \log n)$, it must have at least $n^{3/5}$ gaps at some point. But when this happens, we can trick $\mathcal{A}$ by changing the remaining input to be copies of x, filling up all these gaps. Then the cost becomes at least $n^{3/5}n^{4/5}$ which breaks the promise of competitive ratio $O(\sqrt{n})$, since $\texttt{OPT}(U \cup \{x\}) = 2n^{4/5} - 1$.

We now formally prove the theorem. Let $\mathcal{X}$ be the random input served by Oblivious-Random-Adversary(n), which we define below.

Oblivious-Random-Adversary(n)
1. Let U be a set of $n^{4/5}$ points with pairwise distance 1.
2. Let x be a point with distance $n^{4/5}$ to every point in U.
3. While less than n points have been served:
   a. With probability $n^{-3/5}$:
      I. Let m be the number of points served.
      II. Serve $n - m$ copies of x.
   b. Otherwise:
      I. Serve a copy of the points in U.

By closely following the proof of Theorem 11, replacing their random input by $\mathcal{X}$, we get that the expected cost of any deterministic algorithm on $\mathcal{X}$ is $\Omega(n)$. A full proof can be found in the full version of the paper [4].

Let $\mathcal{A}(\mathcal{X})$ denote the cost of $\mathcal{A}$ on $\mathcal{X}$. Considering $\mathcal{A}$ as a random variable over deterministic algorithms, we get $\mathbb{E}[\mathcal{A}(\mathcal{X})] \in \Omega(n)$. We will now derive an upper bound contradicting this lower bound. To utilize our assumed competitive ratio for the uniform metric, note that $\mathcal{X}$ follows the uniform metric when $x \notin \mathcal{X}$. From the definition of Oblivious-Random-Adversary, we have $\Pr[x \in \mathcal{X}] = 1 - \Pr[x \notin \mathcal{X}] = 1 - (1 - n^{-3/5})^{n^{1/5}} \leq n^{-2/5}$, using Bernoulli's inequality. Using this bound, we get

$$\begin{aligned}\mathbb{E}[\mathcal{A}(\mathcal{X})] &= \Pr[x \in \mathcal{X}]\,\mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \in \mathcal{X}] + \Pr[x \notin \mathcal{X}]\,\mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \notin \mathcal{X}] \\ &\leq n^{-2/5}\,\mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \in \mathcal{X}] + \mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \notin \mathcal{X}].\end{aligned}$$

By assumption, $\mathcal{A}$ has competitive ratio $O(\sqrt{n})$ in general, and competitive ratio $O(\log n)$ when x is not in the input. It's easy to see that $\texttt{OPT}(\mathcal{X}) \in O(n^{4/5})$, so we have $\mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \in \mathcal{X}] \in O(n^{4/5}\sqrt{n})$ and $\mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \notin \mathcal{X}] \in O(n^{4/5} \log n)$. In total, $\mathbb{E}[\mathcal{A}(\mathcal{X})] \in O(n^{-2/5} n^{4/5} \sqrt{n} + n^{4/5} \log n) \subseteq o(n)$, contradicting $\mathbb{E}[\mathcal{A}(\mathcal{X})] \in \Omega(n)$. ◀

5 Open questions

Both [1] and [2] additionally study a variant of the problem with extra space. In this variant, the array is of length γn for some $\gamma > 1$, and empty cells are ignored in the cost function. A significant gap between the best-known upper and lower bound remains, even for online sorting of reals with extra space [1]. For the uniform metric, a tight $\Theta(1 + \log(\gamma/(\gamma - 1)))$ competitive ratio is known [2]. We repeat it as an open question to tighten the gap for online sorting of reals with extra space, and suggest the introduction of an algorithm for online metric TSP with extra space.

References

1 Anders Aamand, Mikkel Abrahamsen, Lorenzo Beretta, and Linda Kleist. Online sorting and translational packing of convex polygons. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1806–1833. SIAM, 2023. doi:10.1137/1.9781611977554.CH69.

2 Mikkel Abrahamsen, Ioana O. Bercea, Lorenzo Beretta, Jonas Klausen, and László Kozma. Online sorting and online TSP: randomized, stochastic, and high-dimensional. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.5.

3 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001. doi:10.1007/S004530010071.

4 Christian Bertram. Online metric TSP. *CoRR*, abs/2504.17716, 2025. doi:10.48550/arXiv.2504.17716.

5 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

6 Daniel J. Rosenkrantz, Richard Edwin Stearns, and Philip M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6(3):563–581, 1977. doi:10.1137/0206041.

An $O(n \log n)$ Algorithm for Single-Source Shortest Paths in Disk Graphs

Mark de Berg
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Sergio Cabello
Faculty of Mathematics and Physics, University of Ljubljana, Slovenia
Institute of Mathematics, Physics and Mechanics, Slovenia

Abstract

We prove that the single-source shortest-path problem on disk graphs can be solved in $O(n \log n)$ expected time, and that it can be solved on intersection graphs of fat triangles in $O(n \log^3 n)$ time.

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

Keywords and phrases shortest path, geometric intersection graph, disk graph, fat triangles

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.81

Related Version *Full Version*: arXiv:2506.07571 [11]

Funding *Mark de Berg*: Supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.
Sergio Cabello: Funded in part by the Slovenian Research and Innovation Agency (P1-0297, N1-0218, N1-0285). Funded in part by the European Union (ERC, KARST, project number 101071836). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

Finding shortest paths in graphs is a classic topic covered in all basic courses and textbooks on algorithms. In the most basic setting, which is the setting we consider here, the graph is unweighted and the length of a path refers to the number of edges in the path. In the single-source shortest-path (SSSP) problem, the task is to compute shortest paths from a given vertex, the *source*, to all other vertices of the graph. The solution is represented using a so-called *shortest-path tree*.

The SSSP problem for a graph with n vertices and m edges can be solved in $O(n + m)$ time by breadth-first search, which is optimal if the graph is given as a collection of vertices and edges. In this paper we will consider the SSSP problem for implicitly defined graphs. In particular, we consider (planar) *intersection graphs*: graphs whose node set corresponds to a set $\mathcal{D}$ of n objects in the plane and that contain an edge (D, D') between two distinct objects $D, D' \in \mathcal{D}$ iff D and D' intersect. If the objects in $\mathcal{D}$ are disks then the intersection graph, which we denote by $\mathcal{G}^\times[\mathcal{D}]$, is called a *disk graph* – see Figure 1 – and if all disks have unit radius then it is called a *unit-disk graph*. Disk graphs are arguably the most popular and widely studied intersection graphs. One can solve the SSSP problem on intersection graphs by first constructing $\mathcal{G}^\times[\mathcal{D}]$ explicitly and then running breadth-first search. In the worst case, however, this requires $\Omega(n^2)$ time. This raises the question: is it possible to solve the SSSP problem on intersection graphs in subquadratic time in the worst case?

Given the basic nature of the SSSP problem and the prominence of (unit-)disk graphs, it is not surprising that this question has already been considered for such graphs. For unit disks, Roditty and Segal [24] noticed that the dynamic data structure of Chan [6] for


© Mark de Berg and Sergio Cabello;
licensed under Creative Commons License CC-BY 4.0

nearest-neighbor queries can be used to solve the problem in $O(n \operatorname{polylog} n)$ expected time. Cabello and Jejčič [4] gave an $O(n \log n)$ algorithm and showed that this is asymptotically optimal in the algebraic decision-tree model. They remarked that Efrat noted that the semi-dynamic data structure of Efrat, Itai and Katz [14] also gives an algorithm with running time $O(n \log n)$. Finally, Chan and Skrepetos [8] provided a simpler $O(n \log n)$ algorithm. The above algorithms either use Delaunay triangulations [4] or fixed-resolution grids [8, 14]; these approaches are not applicable for disks of arbitrarily different sizes.

For arbitrary disks graphs, Kaplan et al. [20] and Liu [22] presented algorithms running in $O(n \log^4 n)$ time. This was recently improved by Klost [21], who presented a general framework for solving SSSP problems in intersection graphs. She used the framework to obtain an $O(n \log^2 n)$ algorithm for the SSSP problem on disk graphs, and to obtain an $O(n \log n)$ algorithm for intersection graphs of axis-aligned squares.

Our results. Our main result in this paper is an algorithm for the SSSP problem in disk graphs that runs in $O(n \log n)$ expected time. We obtain this result using a version of the framework proposed by Klost [21]; our specialized framework and its relation to Klost's framework are discussed in detail in Section 2. A core ingredient in the framework is a *clique-based contraction* of the intersection graph $\mathcal{G}^\times[\mathcal{D}]$: a graph obtained from $\mathcal{G}^\times[\mathcal{D}]$ by contracting certain cliques to single nodes. To compute such a clique-based contraction in $O(n \log n)$ time, we combine shifted quadtrees [5] and skip quadtrees [15] – powerful and beautiful concepts that we believe deserve more attention – with standard techniques for stabbing fat objects.

Our algorithm for computing a clique-based contraction not only applies to disk graphs but also to intersection graphs of *fat triangles*, that is, triangles all of whose internal angles are lower bounded by a fixed constant $\alpha > 0$ [2]. By combining this with a novel intersecting-detection data structure for fat triangles, we obtain an $O(n \log^3 n)$ algorithm for the SSSP problem on intersection graphs of fat triangles. We are not aware of previous work on the SSSP problem for fat triangles. The all-pairs shortest-path (APSP) problem, however, has been considered for fat triangles, by Chan and Skrepetos [9]. The algorithm they present runs in $O(n^2 \log^4 n)$ time, under the condition that the fat triangles have roughly the same size. (They also present an $O(n^2 \log n)$ algorithm for the APSP problem for arbitrary disk graphs, and an $O(n^2)$ algorithm for unit-disk graphs [8].) Running our new SSSP algorithm n times, once for each input triangle as the source, we obtain an APSP algorithm for arbitrarily-sized fat triangles that runs in $O(n^2 \log^3 n)$ time. This is faster and more general than the algorithm of Chan and Skrepetos. (The improvement for the APSP problem is mainly because of our new intersection-detection data structure for fat triangles; using this data structures in the existing algorithm [9] would also give an $O(n^2 \log^3 n)$ algorithm.)

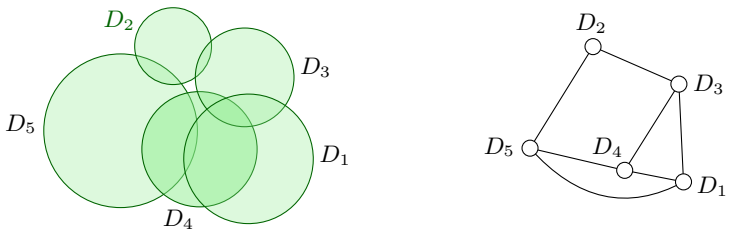


Figure 1 A set $\mathcal{D}$ of five disks (left) and their intersection graph (right).

In the full version of this work [11] we provide two enhancements: we obtain a *deterministic* $O(n \log n)$ algorithm for the SSSP problem in disk graphs and we reduce the running time for the SSSP problem on intersection graphs of fat triangles to $O(n \log^2 n)$. The basic structure of the algorithms is the same as here, but we improve some of the subroutines that we use.

On the model of computation. We use the real-RAM model – this is the standard model in computational geometry, which allows us for example to check in $O(1)$ time if two disks intersect – extended with an additional operation that allows us to compute a compressed quadtree on a set of n points in $O(n \log n)$ time. This extension is common when working with quadtrees; see the book by Har-Peled [19].

2 The framework

Our framework for computing a shortest-path tree on an intersection graph $\mathcal{G}^\times[\mathcal{D}]$ is an instantiation of the framework of Klost [21]. We describe it in detail to keep the paper self-contained and make the required subroutines explicit; at the end of this section we discuss the correspondence of our approach to that of Klost. For convenience, we will from now on not distinguish between the objects in the set $\mathcal{D}$ and the corresponding nodes in $\mathcal{G}^\times[\mathcal{D}]$.

Let $D_{\text{src}} \in \mathcal{D}$ be the given source node, and let $\mathcal{T}_{\text{sp}}$ be the shortest-path tree of $\mathcal{G}^\times[\mathcal{D}]$ that we want to compute for the given source. For an object $D \in \mathcal{D}$, let $\text{dist}[D]$ be the distance from D_{src} to D in $\mathcal{G}^\times[\mathcal{D}]$, and let $L_\ell := \{D \in \mathcal{D} : \text{dist}[\mathcal{D}] = \ell\}$. Thus, L_ℓ is the set of nodes at level ℓ in $\mathcal{T}_{\text{sp}}$. We follow the natural approach of computing $\mathcal{T}_{\text{sp}}$ level by level. To be able to go from one level to the next, we need a subroutine for the following problem.

> Bichromatic Intersection Testing. Given a set B of n_B blue objects and a set R of n_R red objects, report the blue objects that intersect at least one red object, and for each reported blue object, report a witness (a red object intersecting it).

A subroutine *BIT-Subroutine* that solves Bichromatic Intersection Testing allows us to compute $L_{\ell+1}$ by setting $R := L_\ell$ and $B := \mathcal{D}_{\text{cand}}$, where $\mathcal{D}_{\text{cand}}$ is a set of *candidate objects* that should contain all objects from $L_{\ell+1}$ and no objects from $L_{\leqslant \ell}$, where $L_{\leqslant \ell} := \bigcup_{i=0}^{\ell} L_i$. We could simply set $\mathcal{D}_{\text{cand}} := \mathcal{D} \setminus L_{\leqslant \ell}$, but this will not be efficient; we must keep the total size of the candidates sets over all levels ℓ under control. We do this using a so-called clique-based contraction, as defined next.

Let $\mathcal{G} = (V, E_{\mathcal{G}})$ be an (undirected) graph. We say that a graph $\mathcal{H} = (\mathcal{C}, E_{\mathcal{H}})$ is a *clique-based contraction* of $\mathcal{G}$ if

- each node $C \in \mathcal{C}$ corresponds to a clique in $\mathcal{G}$ and the cliques in $\mathcal{C}$ form a partition[1] of the nodes in V;
- there is an edge $(C, C') \in E_{\mathcal{H}}$ iff there are nodes $v \in C$ and $v' \in C'$ such that $(v, v') \in E_{\mathcal{G}}$.

Note that for each edge $(v, v') \in E_{\mathcal{G}}$ there exists a clique $C \in \mathcal{C}$ that contains both v and v', or there are cliques $C \ni v$ and $C' \ni v'$ such that $(C, C') \in E_{\mathcal{H}}$.

A clique-based contraction $\mathcal{H}$ of the intersection graph $\mathcal{G}^\times[\mathcal{D}]$ helps us to find good candidate sets. To see this, let $\mathcal{C}_\ell \subset \mathcal{C}$ denote the set of cliques[2] that contain at least one object from L_ℓ. Then any object $D \in L_{\ell+1}$ must be part of a clique C that is either in $\mathcal{C}_\ell$

[1] In our application, it would also be sufficient to require that each node in V appears in $O(1)$ cliques, but we do not need this extra flexibility.

[2] With a slight abuse of terminology, we not only use the term *clique* to refer a complete graph, but also to refer to a set of geometric objects whose intersection graph is a clique.

itself or that is a neighbor in $\mathcal{H}$ of a clique in $\mathcal{C}_\ell$. In other words, if $D \in L_{\ell+1}$ then $D \in C$ for some clique $C \in N_{\mathcal{H}}[\mathcal{C}_\ell]$, where $N_{\mathcal{H}}[\mathcal{C}_\ell]$ denotes the closed neighborhood of $\mathcal{C}_\ell$ in $\mathcal{H}$. Moreover, the objects in a clique are in at most two levels in the shortest-path three $\mathcal{T}_{\text{sp}}$, which helps to control the total size of the candidate sets. The following pseudocode describes our framework in detail. It does not explicitly construct the shortest-path tree itself (it only constructs the levels), but this is easy to do using the witnesses reported by *BIT-Subroutine*.

Algorithm 1 *SSSP-for-Geometric-Intersection-Graphs*$(\mathcal{D}, D_{\text{src}})$.

1: Construct a clique-based contraction $\mathcal{H} = (\mathcal{C}, E_{\mathcal{H}})$ of $\mathcal{G}^\times[\mathcal{D}]$
2: $L_0 \leftarrow \{D_{\text{src}}\}$; $\mathcal{C}_0 \leftarrow \{$the clique $C \in \mathcal{C}$ that contains $D_{\text{src}}\}$ ▷ D_{src} is the source
3: Label D_{src} as *ready*; $\ell \leftarrow 0$; *done* $\leftarrow$ FALSE.
4: **while** not *done* **do**
5: $\quad\mathcal{D}_{\text{cand}} \leftarrow \{D \in \mathcal{D} : D \in C$ for a clique $C \in N_{\mathcal{H}}[\mathcal{C}_\ell]$ and D is not *ready*$\}$
6: $\quad L_{\ell+1} \leftarrow$ *BIT-Subroutine*$(\mathcal{D}_{\text{cand}}, L_\ell)$
7: $\quad$**if** $L_{\ell+1} = \emptyset$ **then**
8: $\quad\quad L_\infty \leftarrow \mathcal{D} \setminus L_{\leqslant \ell}$; *done* $\leftarrow$ TRUE ▷ nodes in L_∞ are unreachable from D_{src}
9: $\quad$**else**
10: $\quad\quad \mathcal{C}_{\ell+1} \leftarrow \{C \in \mathcal{C} : C$ contains an object $D \in L_{\ell+1}\}$
11: $\quad\quad$ Label all objects in $L_{\ell+1}$ as *ready*; $\ell \leftarrow \ell + 1$

We obtain the following theorem.

▶ **Theorem 1.** *Let $\mathcal{D}$ be a set of n constant-complexity objects in the plane. Suppose that*
(i) *we can compute a clique-based contraction for $\mathcal{G}^\times[\mathcal{D}]$ in $T_{\text{ccg}}(n)$ time,*
(ii) *for any two subsets $B, R \subset \mathcal{D}$ we can solve* Bichromatic Intersection Testing *in $T_{\text{bit}}(n_B, n_R)$ time, where $n_B := |B|$ and $n_R := |R|$.*
Then we can compute a shortest-path tree in $\mathcal{G}^\times[\mathcal{D}]$ for a given source node $D_{\text{src}} \in \mathcal{D}$ in $O(T_{\text{ccg}}(n) + T_{\text{bit}}(n, n))$ time.

Proof. It is straightforward to prove by induction on ℓ that our algorithm correctly computes the levels L_ℓ of the shortest-path tree. To prove the bound on the running time, observe that Steps 1–3 take $O(T_{\text{ccg}}(n))$ time. It remains to bound the runtime of the while-loop.

For a clique $C \in \mathcal{C}$, let $\text{dist}[C] := \min_{D \in C} \text{dist}[D]$. Note that for objects D, D' in cliques $C, C' \in \mathcal{C}$ that are neighbors in $\mathcal{H}$, we have $|\,\text{dist}[D] - \text{dist}[D']\,| \leqslant 3$. Hence, D can only be in the candidate set $\mathcal{D}_{\text{cand}}$ in iterations of the while-loop where $\text{dist}[D] - 3 \leqslant \ell \leqslant \text{dist}[C] - 1$. Thus, if we denote the size of $\mathcal{D}_{\text{cand}}$ in iteration ℓ by n_ℓ, then the total time needed for Step 6 over all iterations is $\sum_\ell T_{\text{bit}}(n_\ell, |L_\ell|)$, where $\sum_\ell n_\ell \leqslant 3n$ and $\sum_\ell |L_\ell| \leqslant n$. Since $T_{\text{bit}}(n_B, n_R)$ is at least linear in n_B and n_R (and at most quadratic), we have $\sum_\ell T_{\text{bit}}(n_\ell, |L_\ell|) = O(T_{\text{bit}}(n, n))$. This also bounds the total runtime of Steps 7–11 over all iterations.

To bound the total time for Step 5, note that a clique C is only considered in iterations where $\text{dist}[C] - 2 \leqslant \ell \leqslant \text{dist}[C] + 2$. This implies that the total time to find the relevant cliques in Step 5 is $O(|E_{\mathcal{H}}|)$. The total time to inspect those cliques in order to find the candidate sets, is $O(\sum_{C \in \mathcal{C}} |C|)$. Thus, the total time for Step 5 over all iterations is $O(|E_{\mathcal{H}}| + \sum_{C \in \mathcal{C}} |C|)$, which can be bounded by $O(T_{\text{ccg}}(n))$. ◀

▶ Remark (relation to the framework of Klost). The framework presented above is an instantiation of the framework of Klost [21]: she also computes the shortest-path tree level by level as described above – this natural approach was also taken by earlier papers on

the SSSP problem on (unit-)disk graphs [4, 8, 9, 21] – and she also keeps the size of the candidate sets under control using an auxiliary graph. Klost uses a so-called *shortcut graph* $\mathcal{G}_{sc}$ as auxiliary graph. The nodes in $\mathcal{G}_{sc}$ correspond to the objects in $\mathcal{D}$, and the edge set E_{sc} is a superset of the edge set of $\mathcal{G}^{\times}[\mathcal{D}]$ such that for any edge $(D, D') \in E_{sc}$ we have that the distance between D and D' in $\mathcal{G}^{\times}[\mathcal{D}]$ is $O(1)$. As E_{sc} is a superset of the edge set of $\mathcal{G}^{\times}[\mathcal{D}]$, the size of E_{sc} can be quadratic, so $\mathcal{G}_{sc}$ needs to be constructed implicitly. Our clique-based contraction $\mathcal{H} = (\mathcal{C}, E_{\mathcal{H}})$ provides such an implicit shortcut graph, by defining $E_{sc} := \{ (D, D') : D \text{ and } D' \text{ are part of the same clique in } \mathcal{C} \text{ or of neighboring cliques} \}$.

3 The algorithm for disks

In this section we implement the framework presented above for the case where the input $\mathcal{D}$ is a set of n disks in the plane. We first explain how to create the cliques in our clique-based contraction $\mathcal{H}$ of $\mathcal{G}^{\times}[\mathcal{D}]$, and how to compute the edge set of $\mathcal{H}$. Using known results on Bichromatic Intersection Testing for disks we then obtain our final result.

Finding the cliques. Our algorithm to create the set of cliques is related to a recent construction by Chan and Huang [7, Section 3] of 3-hop spanners for disk graphs; we comment more on the similarities and differences later. The construction is based on so-called shifted quadtrees, introduced by Chan [5], which we describe next.

We start by defining hierarchical grids, a concept closely related to quadtrees; the connection will be made clear below. Let $o = (o_x, o_y)$ be any point in the plane. A *cell* in the hierarchical grid centered at o is any square of the form

$$\sigma(o, \ell, i, j) := \big[o_x + i \cdot 2^{\ell}, o_x + (i+1) \cdot 2^{\ell}\big) \times \big[o_y + j \cdot 2^{\ell}, o_y + (j+1) \cdot 2^{\ell}\big)$$

for integers $i, j, \ell \in \mathbb{Z}$. Note that $\sigma(o, \ell, i, j)$ is a translation of the square $[0, 2^{\ell}) \times [0, 2^{\ell})$ by the vector $o + (i \cdot 2^{\ell}, j \cdot 2^{\ell})$, and that the set $\Sigma(o, \ell) := \{\sigma(o, \ell, i, j) : i, j \in \mathbb{Z}\}$ forms a regular grid. Moreover, the grid $\Sigma(o, \ell - 1)$ is a refinement of the grid $\Sigma(o, \ell)$, obtained by partitioning each cell of $\Sigma(o, \ell)$ into four quadrants. We define the *hierarchical grid centered at* o, denoted by $\Gamma(o)$, to be the (infinite) collection $\{\Sigma(o, \ell) : \ell \in \mathbb{Z}\}$ of nested grids.

Define the *size* of any object D, denoted by $\text{size}(D)$, to be the side length of a smallest enclosing (axis-parallel) square of D. Thus $\text{size}(\sigma) = 2^{\ell}$ for any cell $\sigma \in \Sigma(o, \ell)$. We say that an object D is *c-aligned* with a hierarchical grid Γ, for a given constant $c > 0$, if there exists a cell σ in Γ such that $D \subset \sigma$ and $\text{size}(\sigma) \leqslant c \cdot \text{size}(D)$. Note that whether or not D is c-aligned with Γ only depends on the choice of the center o of the hierarchical grid. The definition of being c-aligned trivially implies the following.

▶ **Observation 2.** *Let D be an object that is c-aligned with a hierarchical grid Γ, and let σ be a cell of Γ such that D intersects the boundary $\partial\sigma$ of σ. Then* $\text{size}(D) \geqslant (1/c) \cdot \text{size}(\sigma)$.

Let $\mathcal{D}$ be a finite set of objects. In general, it is impossible to choose the center of a hierarchical grid Γ such that each object $D \in \mathcal{D}$ is c-aligned with Γ. However, Chan [5] has shown that, surprisingly, it *is* possible to pick a small number of different centers such that each object $D \in \mathcal{D}$ is c-aligned with at least one of the resulting hierarchical grids, for a suitable constant c. Even more surprisingly, we can select these centers independently of $\mathcal{D}$. The following lemma follows directly from Lemma 3.2 in Chan's paper.[3]

[3] Chan uses the term *shifted quadtree* in his lemma. To stress the fact that the shifts are independent of any input set on which one might build a quadtree, we prefer to use the term *hierarchical grid*.

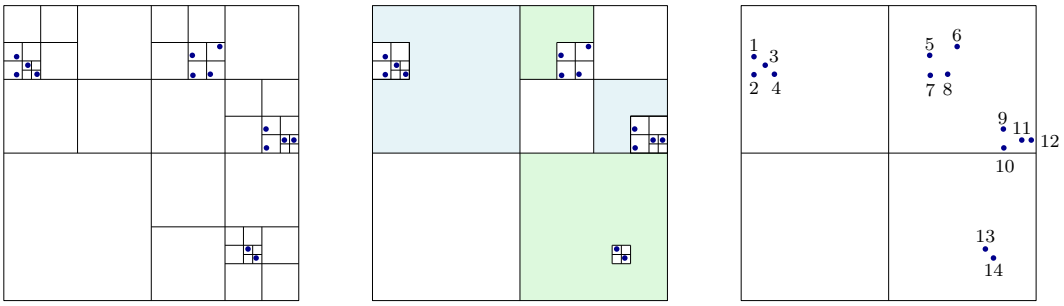


Figure 2 (i) A quadtree subdivision for a set P of points. (ii) The compressed-quadtree subdivision for P, with its donut cells marked. (iii) The ordering of the points to construct a skip quadtree.

▶ **Lemma 3** ([5]). *There is a collection Ξ of three hierarchical grids, each centered at a different point, with the following property: for any object D contained in the square $[0,1) \times [0,1)$ there is a hierarchical grid $\Gamma \in \Xi$ such that D is 6-aligned with Γ.*

Let P be a set of points, let Γ be a hierarchical grid centered at a given center o, and let σ_0 be a cell of Γ that contains P. Then we can construct a *quadtree* on P – see Figure 2(i) – whose root node corresponds to σ_0. Any node in this quadtree corresponds to a cell of Γ. Let $\mathcal{Q} = \mathcal{Q}(\sigma_0, P)$ be the corresponding *compressed quadtree* [19]. Each internal node ν of $\mathcal{Q}$ corresponds to a cell σ_ν of Γ and each leaf node μ corresponds to a region σ_μ that is either a cell of Γ or a donut cell; see Figure 2(ii). (A *donut cell* is the difference $\sigma_1 \setminus \sigma_2$ of two cells σ_1 and σ_2 of Γ with $\sigma_2 \subset \sigma_1$.) The *subdivision* defined by the compressed quadtree $\mathcal{Q}$ is the set of regions corresponding to the leaves of $\mathcal{Q}$; this subdivision is a decomposition of σ_0. Our algorithm to efficiently compute the cliques in the clique-based contraction combines the hierarchical grids of Lemma 3 with skip quadtrees [15], as explained next.

Skip quadtrees, defined by Eppstein, Goodrich, and Sun [15] are defined as follows. Let P be a set of n points inside a given starting cell σ_0 of a hierarchical grid Γ. We start by sorting the points in P in Z-order, as follows. First, order the points according to the quadrants from σ_0 they fall in: put the points in the NW-quadrant first, then those in the NE-quadrant, then those in the SW-quadrant, and finally those in the SE-quadrant. Next, recursively sort the points in each quadrant. Let $p_1, \ldots, p_n$ be the resulting sorted list of points; see Figure 2(iii). We now create a sequence $P = P_0 \supset P_1 \supset \cdots \supset P_t$, where P_i is obtained from P_{i-1} by deleting every other element (starting with the first element) and $|P_t| = 1$. From the sequence $P_0 \supset P_1 \supset \cdots \supset P_t$ we construct a sequence of compressed-quadtree *subdivisions* $\mathcal{Q}_0, \mathcal{Q}_1, \ldots, \mathcal{Q}_t$, each based on the same starting cell σ_0. The sequence $\mathcal{Q}_0, \mathcal{Q}_1, \ldots, \mathcal{Q}_t$ has the following properties, as illustrated in Figure 3:

- $\mathcal{Q}_t$ is equal to the square σ_0.
- $\mathcal{Q}_i$ is a refinement of $\mathcal{Q}_{i+1}$ for all $0 \leqslant i < t$, that is, each region in $\mathcal{Q}_i$ is contained in a unique region in $\mathcal{Q}_{i+1}$. Moreover, each region in $\mathcal{Q}_{i+1}$ contains $O(1)$ regions from $\mathcal{Q}_i$.
- Each region in $\mathcal{Q}_0$ contains at most one point from P.

We can view $\mathcal{Q}_t, \ldots, \mathcal{Q}_1, \mathcal{Q}_0$ as the levels of a tree $\mathcal{T}$, where a node ν at level i corresponds to a region R_ν of the subdivision $\mathcal{Q}_i$. In particular, the root of $\mathcal{T}$ corresponds to the starting square σ_0 and the leaves of $\mathcal{T}$ correspond to the regions in $\mathcal{Q}_0$ (which is the compressed-quadtree subdivision for the whole point set P with root σ_0). We call this tree structure[4] a *skip quadtree* for P with root σ_0. It can be constructed in $O(n \log n)$ time [15].

[4] The skip quadtree defined by Eppstein, Goodrich, and Sun is slightly more complicated, as it stores

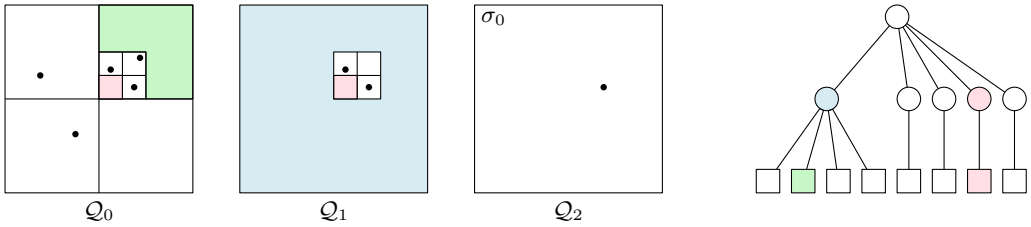


Figure 3 Example of the three compressed-quadtree subdivisions for a set of five points, and the resulting skip quadtree. Colors indicate the correspondence between regions and nodes.

Without loss of generality, we will assume henceforth that all the disks of $\mathcal{D}$ are fully contained in the unit square $[0,1) \times [0,1)$. Let Ξ be the collection of three hierarchical grids given by Lemma 3. Each disk D of $\mathcal{D}$ is 6-aligned with some hierarchical grid $\Gamma \in \Xi$, and we can find such a hierarchical grid for D in constant time.

Now consider a hierarchical grid Γ from the collection Ξ. Let $\mathcal{D}(\Gamma) \subset \mathcal{D}$ be the set of disks that are 6-aligned with Γ, where we put a disk that is 6-aligned with multiple grids into an arbitrary one of the corresponding sets $\mathcal{D}(\Gamma)$. We create a set $\mathcal{C}(\Gamma)$ of cliques, as follows.

1. Construct a skip quadtree $\mathcal{T}(\Gamma)$ on the set $P(\Gamma)$ of centers of the disks in $\mathcal{D}(\Gamma)$, where the starting square σ_0 of the skip quadtree is a cell of Γ that contains all the disks in $\mathcal{D}(\Gamma)$.
2. For each disk $D \in \mathcal{D}(\Gamma)$, follow a path in $\mathcal{T}(\Gamma)$ consisting of nodes μ whose region R_μ contains the center of D, until a node ν is reached such that D intersects ∂R_ν or a leaf is reached. Assign D to that node or leaf.
3. Let $\mathcal{D}_\nu \subset \mathcal{D}(\Gamma)$ be the set of disks assigned to an (internal or leaf) node ν in $\mathcal{T}(\Gamma)$. For each node ν, partition $\mathcal{D}_\nu$ into a set $\mathcal{C}_\nu$ of $O(1)$ cliques, as follows. If ν is a leaf and R_ν contains a disk – note that a leaf can contain at most one disk – then we create a singleton clique for that disk. Now suppose ν is an internal node. First, suppose R_ν is a square region. Because the disks in $\mathcal{D}_\nu$ are 6-aligned with Γ, we know that $\text{size}(D) \geqslant \frac{1}{6} \cdot \text{size}(R_\nu)$ for all $D \in \mathcal{D}_\nu$ intersecting ∂R_ν. Hence, we can create a set of $O(1)$ points such that each disk D is stabbed by at least one of them, which we use to create $\mathcal{C}_\nu$. If R_ν is a donut we proceed similarly, where we treat the disks intersecting the boundary of the hole of the donut – this hole can be much smaller than the donut – separately from the disks that only intersect the outer boundary of the donut. Finally, set $\mathcal{C}(\Gamma) := \bigcup_{\nu \in \mathcal{T}(\Gamma)} \mathcal{C}_\nu$.

Each clique that is generated by our algorithm is a so-called *stabbed clique*, that is, for each clique $C \in \mathcal{C}(\Gamma)$ there is a point stabbing all disks in C. We refer to the union of all the disks in C as a *flower*, which we denote by $F(C)$, and we let $\mathcal{F}(\Gamma) := \{F(C) : C \in \mathcal{C}(\Gamma)\}$ be the set of flowers corresponding to the cliques in $\mathcal{C}(\Gamma)$. Define $\text{ply}(S)$, the *ply* of a set S of objects in the plane, to be the maximum, over all points $q \in \mathbb{R}^2$, of the number of objects in S containing q. The following lemma states a property of our algorithm that will be crucial to efficiently compute the edges in the clique-based contraction.

▶ **Lemma 4.** *The algorithm above constructs in $O(n \log n)$ time a partition of $\mathcal{D}(\Gamma)$ into a set $\mathcal{C}(\Gamma)$ of stabbed cliques such that $\text{ply}(\mathcal{F}(\Gamma)) = O(\log n)$.*

more information than just the compressed-quadtree subdivisions. Since we do not need our structure to be dynamic, the simple hierarchy described above suffices.

Proof. Step 1 of the algorithm, constructing the skip quadtree $\mathcal{T}(\Gamma)$ on the set $P(\Gamma)$ of the centers of the at most n disks in $\mathcal{D}(\Gamma)$, can be done in $O(n \log n)$ time [15]. The depth of the skip quadtree is $O(\log n)$, and so Step 2 takes $O(\log n)$ time per disk and $O(n \log n)$ time in total. Step 3 takes $\sum_{v \in \mathcal{T}(\Gamma)} O(1 + |\mathcal{D}_v|)$ time, which is $O(n)$ because each disk is assigned to exactly one node. Hence, the total time for the algorithm is $O(n \log n)$.

To bound $\text{ply}(\mathcal{F}(\Gamma))$, consider an arbitrary point $q \in \mathbb{R}^2$. Recall that for each node ν in $\mathcal{T}(\Gamma)$ we created $O(1)$ stabbed cliques and, hence, $O(1)$ flowers. Thus, it suffices to prove the following claim: at each level of the skip quadtree $\mathcal{T}(\Gamma)$ there are only $O(1)$ nodes ν such that $q \in D$ for some disk D assigned to ν. To prove this claim, follow the search path of q in $\mathcal{T}(\Gamma)$, that is, the path consisting of nodes μ such that $q \in R_\mu$. Let $D \in \mathcal{D}$ be a disk that contains q, let ν be the node to which D is assigned, and let μ be the parent of ν. Then μ must lie on the search path of q. Indeed, D is fully contained in R_μ – otherwise D would have been assigned to μ – and so $q \in D \subset R_\mu$. Since any node in a skip quadtree has $O(1)$ children, this implies the claim. ◀

▶ Remark (relation to the spanner construction of Chan and Huang). In their algorithm for constructing a 3-hop spanner for $\mathcal{G}^\times[\mathcal{D}]$, Chan and Huang [7] create a collection of cliques in a similar way as we create our cliques. As they are only concerned with the size of the spanner, however, they do not analyze the time needed to construct it. The differences between our approach and theirs mainly stem from the necessity to construct the clique-based contraction $\mathcal{H}$ in $O(n \log n)$ time, as discussed next.

A first difference is that Chan and Huang use a centroid decomposition of a compressed quadtree instead of a skip quadtree. This is not a crucial difference – possibly we could have used a centroid decomposition as well, although working with a skip quadtree is more convenient. Another difference is that they work with a collection Ξ of hierarchical grids – they call them shifted quadtrees – such that, for each pair $D, D' \in \mathcal{D}$, there is at least one hierarchical grid $\Gamma \in \Xi$ such that both D and D' are aligned with Γ. This allows them to construct their spanner by applying the following procedure to each quadtree $\mathcal{Q}$ defined by the hierarchical grid Γ: (i) take a centroid node ν in a compressed quadtree $\mathcal{Q}$ and partition the set $\mathcal{D}_\nu$ of disks intersecting $\partial\sigma_\nu$ into $O(1)$ cliques, (ii) for each clique C, add a star graph on that clique to the spanner, (iii) for each disk D and each clique C intersected by D, add an edge (D, D') to the spanner, where $D' \in C$ is an arbitrary disk intersected by D, and (iv) recursively construct 3-hop spanners for the disks inside and outside σ_ν. The crucial difference lies in step (iii), where they add an edge from every disk D to a neighboring disk in each clique (if such a neighboring disk exists). It seems difficult to do this in $O(n \log n)$ time in total over all recursive calls. We therefore proceed differently: we first recursively construct cliques on the disks not intersecting $\partial\sigma_\nu$, and then we only add a single edge between two cliques if the corresponding flowers intersect. Such an approach would increase the spanning ratio if we were to use it to construct a spanner, but this is of no concern to us. The advantage of being more conservative is that we can compute the edges in our clique-based contraction in $O(n \log n)$ time, as explained next.

Computing the edges in the clique-based contraction. Let $\mathcal{C} := \bigcup_{\Gamma \in \Xi} \mathcal{C}(\Gamma)$ be the set of cliques generated by applying the procedure above to each hierarchical grid Γ in the collection Ξ given by Lemma 3. Let $\mathcal{F} := \bigcup_{\Gamma \in \Xi} \mathcal{F}(\Gamma)$ be the corresponding flower set. Note that $\text{ply}(\mathcal{F}) = O(\log n)$ by Lemma 4. To construct the edge set $E_\mathcal{H}$ of the clique-based contraction $\mathcal{H} = (\mathcal{C}, E_\mathcal{H})$, we must find all pairs $F, F' \in \mathcal{F}$ that intersect. Next we show how to do this in $O(n \log n)$ time.

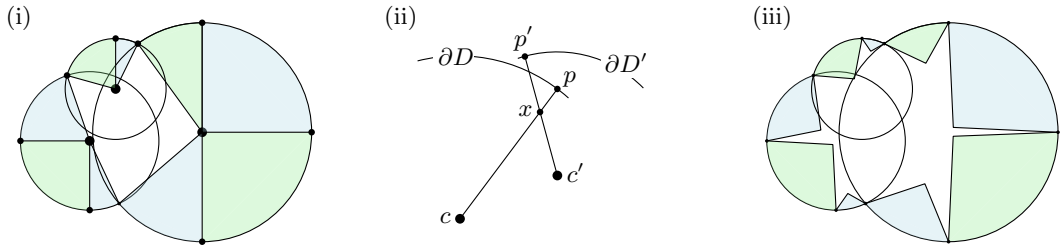


Figure 4 (i) The set Z_F of circular sectors defined by the boundary arcs of a flower F. Some boundary arcs have been cut into sub-arcs to ensure they span an angle of at most $\pi/2$. (ii) If $|px| < |p'x|$ then $p \in D'$; otherwise $p' \in D$. (iii) The modified sectors.

Consider a flower F defined by some clique C. The boundary ∂F of F is comprised of maximal pieces of the boundaries of the disks in C that show up on ∂F. We call these pieces *boundary arcs* and we denote the set of boundary arcs of a flower F by $\mathcal{B}(F)$. For a set $\mathcal{F}$ of flowers, we define $\mathcal{B}(\mathcal{F}) := \bigcup_{F\in\mathcal{F}} \mathcal{B}(F)$ to be the set of boundary arcs of the flowers in $\mathcal{F}$.

▶ **Lemma 5.** *Let $\mathcal{F}$ be a set of flowers that consist of n disks in total. Then the number of intersections between the boundary arcs in $\mathcal{B}(\mathcal{F})$ is $O(n \cdot \mathrm{ply}(\mathcal{F}))$.*

Proof. First, observe that $|\mathcal{B}(F)| = O(n_F)$, where n_F is the number of disks defining a flower $F \in \mathcal{F}$, since the union of a set of disks has linear complexity. Hence, $|B(\mathcal{F})| = O(n)$.

Now consider a boundary arc $\beta \in \mathcal{B}(F)$ of some flower $F \in \mathcal{F}$. We assume for simplicity that the angle spanned by the circular arc β is at most $\pi/2$; if this is not the case we can split β into at most four sub-arcs, and work with the sub-arcs instead. Let D be the disk that contributes the arc β, let c be the center of D, and let s_1 and s_2 be the segments that connect c to the endpoints of β. Together with β, these two segments bound a (convex) circular sector. Let Z_F be the set of such circular sectors created for the arcs in $\mathcal{B}(F)$; see Figure 4(i). We say that two sectors z, z' are *non-overlapping* if their interiors are disjoint.

▷ Claim. The sectors in the set Z_F are pairwise non-overlapping.

Proof. Consider two sectors $z, z' \in Z_F$. Let D, D' be the disks whose boundaries define z and z', respectively, and let c and c' be their centers. If $D = D'$ then obviously z and z' do not overlap, so assume that $D \neq D'$. Since a sector in Z_F cannot be fully contained in another sector in Z_F by construction, the sectors z, z' can only overlap if there is a proper intersection between their boundaries. Such an intersection can only happen between a segment s connecting c to an endpoint p of some boundary arc $\beta \subset \partial D$ and a segment s' connecting c' to an endpoint p' of some boundary arc $\beta' \subset \partial D'$. But then, by the triangular inequality, either $p \in D'$ or $p' \in D$ – see Figure 4(ii) for an illustration – contradicting that p and p' are both points on ∂F. ◁

Now consider a circular sector z, defined by some arc β. Let x be the apex of z (which coincides with the center of the disk D contributing β). If the angle at apex x is large enough then z is fat, but if the angle is very small then this is not the case. We therefore proceed as follows: we move the apex x towards the midpoint of β until the angle at x is slightly larger than $\pi/2$; see Figure 4(iii). It is easy to see that the modified region (which is no longer a circular sector) is fat and convex. Moreover, no two modified regions share an apex anymore. With a slight abuse of notation, from now on we use Z_F to denote the set of modified regions created for a flower $F \in \mathcal{F}$.

Let $Z(\mathcal{F}) := \bigcup_{F \in \mathcal{F}} Z_F$. Note that $|Z(\mathcal{F})| \leqslant 4 \cdot |\mathcal{B}(\mathcal{F})| = O(n)$ and that $\text{ply}(Z(\mathcal{F})) \leqslant 2 \cdot \text{ply}(\mathcal{F}) = O(\log n)$; the latter is true since the regions created for a single flower R are pairwise disjoint except at shared endpoints of boundary arcs. For each arc $\beta \in \mathcal{B}(\mathcal{F})$, there is a region $z \in Z(\mathcal{F})$ such that $\beta \subset \partial z$. Hence, the number of intersections between the arcs in $\mathcal{B}(\mathcal{F})$ is upper bounded by the number of intersections between the regions in $Z(\mathcal{F})$.

▷ Claim. The number of intersections between the regions in $Z(\mathcal{F})$ is $O(|Z(\mathcal{F})| \cdot \text{ply}(Z(\mathcal{F})))$.

Proof. The proof is standard, but we sketch it for completeness. We charge the intersection between two regions z, z' to the smaller of the two regions, and claim that each region is charged $O(\text{ply}(Z(\mathcal{F})))$ times. To see this, consider a region $z \in Z(\mathcal{F})$ and let b be the smallest enclosing disk of Z. Let b' be the disk with the same center as b such that $\text{radius}(b') = 2 \cdot \text{radius}(b)$. Due to the fatness of the regions in $Z(\mathcal{F})$, any region z' that intersects z and is at least as large as z, will cover a constant fraction of the area of b'. Hence, there can be at most $O(\text{ply}(Z(\mathcal{F})))$ such regions z'. ◁

This finishes the proof. ◀

Observe that two flowers F, F' intersect iff the following holds: a boundary arc of F intersects a boundary arc of F', or $F \subset F'$, or $F' \subset F$. Thus we can find all intersecting pairs of flowers – and, hence, all edges of the clique-based contraction $\mathcal{H}$ – as follows. (The algorithm below may report a pair of intersecting flowers multiple times, but this is not a problem.)

First, we construct the arrangement $\mathcal{A}(\mathcal{F})$ induced by the flower set $\mathcal{F}$. Next, for each vertex of $\mathcal{A}(\mathcal{F})$ that is the intersection of the boundaries of some pair of flowers F, F, we report that pair. Finally, we need to find the pairs of flowers F, F' such that $F \subset F'$. To this end, we traverse the dual graph of $\mathcal{A}(\mathcal{F})$, maintaining a list $\mathcal{L}$ of flowers containing the face we are currently in. Whenever we enter a flower F for the first time, we report the pairs (F, F') for the faces F' that are currently in the list $\mathcal{L}$.

We obtain the following lemma.

▶ **Lemma 6.** *Let $\mathcal{D}$ be a set of n disks in the plane. Then we can compute a clique-based contraction for $\mathcal{G}^{\times}[\mathcal{D}]$ in $O(n \log n)$ expected time.*

Proof. Computing the collection $\mathcal{C}$ of cliques takes $O(n \log n)$ time by Lemma 4. It remains to analyze the time needed to compute the edge set $E_{\mathcal{H}}$ of the clique-based contraction.

For each clique $C \in \mathcal{C}$, we can compute the corresponding flower $F(C)$ in $O(|C| \log |C|)$ time using a simple divide-and-conquer algorithm. Since each disk is part of only clique, this implies that the flower set $\mathcal{F}$ can be computed in $O(n \log n)$ time. Moreover, the total number of boundary arcs of the flowers is $O(n)$ because the complexity of a single flower is linear. Hence, we can construct the arrangement $\mathcal{A}(\mathcal{F})$ in $O(n \log n + k)$ expected[5] time, where k is the number of intersections between the boundary arcs, using a randomized incremental algorithm by Mulmuley [23]. Since $k = O(n \log n)$ by Lemmas 4 and 5, constructing the arrangement thus takes $O(n \log n)$ expected time. The traversal of the dual graph, including the maintenance of $\mathcal{L}$ takes $O(n \log n)$ time. Since the size of the list $\mathcal{L}$ is $O(\log n)$ at any time – this is because the ply of the flower set is $O(\log n)$ – we report at most $O(n \log n)$ pairs in total during the traversal. Thus, the total time to compute $E_{\mathcal{H}}$ is $O(n \log n)$. ◀

[5] We cannot use the deterministic algorithm for line-segment intersection by Chazelle and Edelsbrunner [10] because it does not work for curves. The deterministic algorithm for line-segment intersection by Balaban [3] also works for curves, but that algorithm does not seem to give enough information to construct the arrangement.

Putting it all together. Lemma 6 gives us the first ingredient we need to apply Theorem 1. The second ingredient is an algorithm that solves Bichromatic Intersection Testing for disks. As observed in previous papers [9, 21] this can be done in $O((n_B + n_R) \log n_R)$ time, as follows. First, compute the additively weighted Voronoi diagram $\mathrm{Vor}(R)$ on the centers of the set R of red disks, where the weight of a center is equal to the radius of its corresponding disk. Next, query with the center of each blue disk D in $\mathrm{Vor}(R)$ to find the red disk D' closest to D. Now D intersects $\bigcup R$, the union of the red disks, iff D intersects D'. Since $\mathrm{Vor}(R)$ can be computed in $O(n_R \log n_R)$ time [17], after which it can be preprocessed for logarithmic-time point location in $O(n_R \log n_R)$ time [13], we can solve Bichromatic Intersection Testing in $O((n_B + n_R) \log n_R)$ time. We thus obtain our main result.

▶ **Theorem 7.** *Let $\mathcal{D}$ be a set of n disks in the plane. Then we can compute a shortest-path tree in $\mathcal{G}^{\times}[\mathcal{D}]$ for a given source disk $D_{\mathrm{src}} \in \mathcal{D}$ in expected $O(n \log n)$ time.*

In the full version [11] we present a (slightly more complicated) deterministic algorithm to compute the edges of the clique-based contraction for $\mathcal{G}^{\times}[\mathcal{D}]$ in $O(n \log n)$ time. Using that algorithm instead of Lemma 6, we obtain an algorithm to compute shortest-path trees on disks graphs deterministically in $O(n \log n)$ time.

4 Extension to fat triangles

A triangle Δ is called *α-fat* if its minimum angle is at least α. Let $\mathcal{D} = \{\Delta_1, \ldots, \Delta_n\}$ be a set of α-fat triangles, where $\alpha > 0$ is some fixed absolute constant. From now on, we simply refer to the triangles in $\mathcal{D}$ as *fat triangles* and we refer to α as the *fatness constant*.

Constructing the clique-based contraction. Our approach to construct a clique-based contraction for fat triangles is similar to the algorithm for disks. We now go over the various ingredients and explain how to adapt them, if necessary.

We first observe that Lemma 3.2 from Chan's paper [5] actually holds for any type of objects. Hence, our Lemma 3 also holds for any type of objects. To construct the skip quadtree, the set $P(\Gamma)$ of disk centers is replaced by a set $P(\Gamma)$ that contains an arbitrary point in each object. Constructing the cliques can therefore be done in exactly the same way as before. Indeed, the crucial property was as follows: any set of disks intersecting the boundary of a region R_ν and whose size is at least $\frac{1}{6} \cdot \mathrm{size}(R_\nu)$, can be stabbed by $O(1)$ points. This property also holds for fat objects.

Now let C be a stabbed clique of fat triangles. We refer to the union of the triangles in C as a *spiky flower*, and we denote it by $F(C)$. As before, we let $\mathcal{C}(\Gamma)$ denote the set of cliques created by our algorithm, and we define $\mathcal{F}(\Gamma) := \{F(C) : C \in \mathcal{C}(\Gamma)\}$ to be the corresponding set of spiky flowers. Then Lemma 4 also holds for fat triangles – we can follow the proof verbatim, only replacing occurrences of "disk" by "fat triangle."

It remains to prove the equivalent of Lemma 5 for fat triangles. To keep the terminology similar to the case of disks, we will refer to the edges of a spiky flower F as *boundary segments*, and we denote the set of boundary segments of a flower F by $\mathcal{B}(F)$. Furthermore, we let $\mathcal{F} := \bigcup_{\Gamma \in \Xi} \mathcal{F}(\Gamma)$ denote the set of spiky flowers created by our algorithm, and we define $\mathcal{B}(\mathcal{F}) := \bigcup_{F \in \mathcal{F}} \mathcal{B}(F)$.

▶ **Lemma 8.** *Let $\mathcal{F}$ be a set of spiky flowers that consist of n fat triangles in total. Then the number of intersections between the boundary segments in $\mathcal{B}(\mathcal{F})$ is $O(n \cdot \mathrm{ply}(\mathcal{F}))$.*

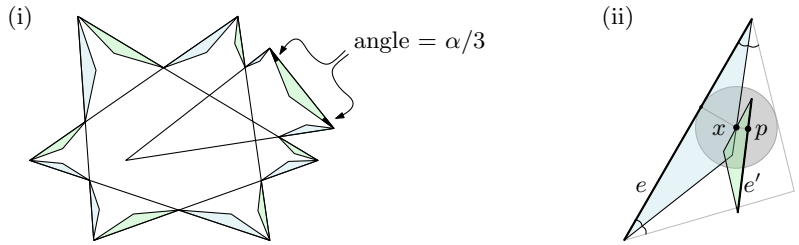


Figure 5 (i) The set Z_F of sectors defined by the boundary segments of a spiky flower F. (ii) The point p lies in D, which in turn lies inside the expanded sector.

Proof. We apply the same proof technique as in the proof of Lemma 5: we cover the boundary segments of each spiky flower F by a set Z_F of non-overlapping fat "sectors" contained in the flower, as explained below, from which the lemma follows.

The number of boundary segments of a spiky flower is $O(n_F)$, where n_F is the number of triangles defining F [1, Section 3.2]. We create the set Z_F for a spiky flower F as follows. For each boundary segment β of F, we create an isosceles triangle $z \subset F$ whose angles at the endpoints of β are $\alpha/3$, where α is the fatness constant of the triangles; see Figure 5. With a slight abuse of terminology, we refer to these isosceles triangles as sectors.

▷ Claim. The sectors in the set Z_F are pairwise non-overlapping.

Proof. Each sector z has one edge that corresponds to a boundary segment of F; we call this the *external edge* of z. The two other edges, which make an angle $\alpha/3$ with the external edge, are called *internal edges*.

Now assume for a contradiction that two sectors z, z' overlap. Since the external edge of any sector cannot properly intersect an (external or internal) edge of any other sector, there must be a crossing between an internal edge of z and an internal edge of z'. Let x be this crossing, and let e and e' be the external edges of z and z', respectively; see Figure 5(ii). Assume without loss of generality that the distance from x to e is at least the distance from x to e'. Let p be the point on e' nearest to x, and let D be the disk centered at x and touching e. Then p is contained in D. Moreover, D is contained in the triangle $\Delta \in \mathcal{D}$ contributing e to the boundary of the spiky flower F. Indeed, if we were to expand the sector z by making the angles at the endpoints of e equal to $2\alpha/3$ then $D \subset z$, and this expanded sector is contained in Δ. Thus, $p \in \Delta$, which contradicts that p lies on an external edge. ◁

As in the proof of Lemma 5, it now follows that the number of intersections between the sectors in $Z(\mathcal{F})$ is $O(|Z(\mathcal{F})| \cdot \text{ply}(Z(\mathcal{F})))$, which proves the lemma. ◀

As before, we have the following corollary. Its proof is the same as the proof of Lemma 6; the only difference is that we can now use the algorithm by Chazelle and Edelsbrunner [10] to construct the arrangement defined by the flowers, thus avoiding the use of randomization.

▶ **Corollary 9.** *Let $\mathcal{D}$ be a set of n fat triangles in the plane. Then we can compute a clique-based contraction for $\mathcal{G}^{\times}[\mathcal{D}]$ in $O(n \log n)$ time.*

Bichromatic Intersection Testing for fat triangles. The final ingredient that we need is an efficient algorithm for Bichromatic Intersection Testing for a set B of n_B blue triangles and a set R of n_R red triangles, where all triangles in $B \cup R$ are fat. Such an algorithm can be developed using standard machinery, as described next.

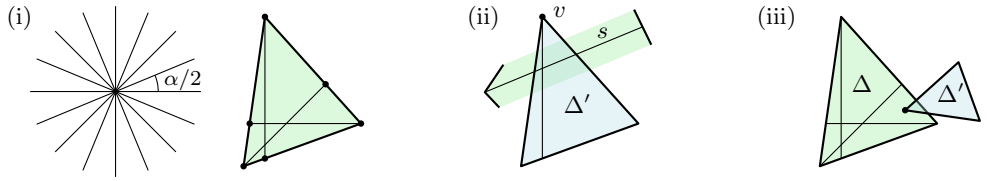


Figure 6 (i) The set A of canonical directions and the canonical chords of a triangle. (ii) Chord s intersects two sides of Δ' so it intersects a canonical chord of Δ'. (iii) All canonical chords of Δ miss Δ', so Δ' must have a vertex inside Δ.

Let $A := \{i \cdot (\alpha/2) : 0 \leqslant i \leqslant \lfloor 4\pi/\alpha \rfloor\}$ be a set of $O(1/\alpha) = O(1)$ *canonical directions*, where α is the fatness constant of the triangles. Define a *canonical segment* to be a segment whose direction is canonical, and define a *canonical chord* of a fat triangle Δ to be a canonical segment that connects a vertex of Δ to its opposite side; see Figure 6(i). Note that each vertex of Δ admits a canonical chord. Let $S(\Delta)$ be a set of three canonical chords of Δ, one per vertex of Δ, and let $P(\Delta)$ be the endpoints of these chords (including the vertices of Δ).

▶ **Observation 10.** *If a fat triangle Δ intersects a fat triangle Δ' then at least one of the following conditions holds:*

(i) *Δ contains a vertex of Δ';*
(ii) *a point from $P(\Delta)$ lies inside Δ';*
(iii) *a canonical chord from $S(\Delta)$ intersects a canonical chord from $S(\Delta')$.*

Proof. Suppose that condition (ii) does not hold. Then there is a canonical chord $s \in S(\Delta)$ that intersects two edges of Δ', or all canonical chords in $S(\Delta)$ are disjoint from Δ'. In the former case, s separates a vertex v of Δ' from its opposite side, which implies that s intersects the canonical chord of v; see Figure 6(ii). Thus, condition (iii) holds in this case. In the latter case, Δ' must have a vertex inside Δ and condition (i) holds; see Figure 6(iii). ◀

Observation 10 implies the following lemma.

▶ **Lemma 11.** *Let R be a set of n fat triangles. There exists a data structure of $O(n \log^2 n)$ size such that we can decide in $O(\log^3 n)$ time if a fat query triangle Δ intersects any triangle in R and, if so, report a witness triangle. The data structure can be built in $O(n \log^3 n)$ time.*

Proof. The data structure consists of three separate data structures, each handling one of the conditions in Observation 10.

For condition (i) we need a data structure for range searching with fat triangles in a set V of $3n$ points, namely the vertices of the triangles in R. As observed by Gray [18, page 14], such queries can be answered in $O(\log^3 n)$ time with a structure of size $O(n \log^2 n)$ that can be built in $O(n \log^3 n)$ time.

For condition (ii) we use a *union tree* on the triangles in R. This is a binary tree $\mathcal{T}$ whose leaves correspond to the triangles in R, and whose internal nodes store the union of the triangles in their subtree, preprocessed for point-location queries. Checking if a query point q is contained in any triangle in R can be done in $O(\log n)$ by performing a point location at the root of $\mathcal{T}$. If this is the case, then we can find a witness triangle in $O(\log^2 n)$ time by walking down $\mathcal{T}$, always proceeding to a child whose union contains q. Since the union complexity of n fat triangles is $O(n \log^* n)$ [2], the total amount of storage is $O(n \log^* n \log n)$. Note that the union at a node ν can be computed in $O(n_\nu \log^* n_\nu \log n_\nu)$ time from the unions of its two children by a simple plane-sweep algorithm, where n_ν is the number of triangles in the subtree of ν. Hence, the total preprocessing time is $O(n \log^* n \log^2 n)$.

To handle condition (iii) we proceed as follows. Let $S_R : \{s \in S(\Delta) : \Delta \in R\}$ be the set of canonical chords of the triangles in R. We will build $O(1/\alpha)$ different data structures on S_R, one for each of the $O(1/\alpha)$ possible directions of the query chord. To construct a data structure for a fixed direction of the query chord, we partition S_R into $O(1/\alpha)$ classes, one for each canonical direction in A, and we build a separate substructure for each class A_i. Assume wlog that the query chord is vertical and that the chords in A_i are horizontal. Then we can use an interval tree [12, Section 10.1] to answer queries in $O(\log n)$, using $O(n)$ space and $O(n \log n)$ preprocessing. (Using a point-location structure on the vertical decomposition of the chords in A_i is another option.) Thus the total amount of storage is $O(n/\alpha) = O(n)$, the total preprocessing is $O((n/\alpha) \log n) = O(n \log n)$, and the total query time is $O((1/\alpha) \log n) = O(\log n)$.

Since all data structure work within the claimed bounds, the lemma follows. ◀

We can now solve Bichromatic Intersection Testing for fat triangles in $O((n_B + n_R) \log^3(n_B + n_R))$ time, by querying with each triangle $\Delta \in B$ in the data structure provided by Lemma 11. This leads to the following theorem.

▶ **Theorem 12.** *Let $\mathcal{D}$ be a set of n fat triangles in the plane. Then we can compute a shortest-path tree in $\mathcal{G}^{\times}[\mathcal{D}]$ for a given source triangle $\Delta_{\text{src}} \in \mathcal{D}$ in $O(n \log^3 n)$ time.*

In the full version [11] we provide a better data structure for the range-query problem considered in Lemma 11: its construction time is $O(n \log^2 n)$ and each query is then answered in $O(\log^2 n)$ time. Using this data structure, we obtain an $O(n \log^2 n)$ algorithm to compute shortest-path trees on the intersection graph of fat triangles.

5 Concluding remarks

We presented the first algorithm for the SSSP problem in disk graphs that runs in $O(n \log n)$ (expected) time. We extended the algorithm to intersection graphs of fat triangles, where we obtain a worst-case running time of $O(n \log^3 n)$. In the full version [11] we provide a deterministic $O(n \log n)$ algorithm for the SSSP problem in disk graphs and improve the running time for intersection graphs of fat triangles to $O(n \log^2 n)$.

A natural question is whether the algorithm for fat triangles can be further improved. Another natural question is whether an $O(n \operatorname{polylog} n)$ algorithm is possible for intersection graphs of non-fat objects such as line segments. This is unlikely, however, because Hopcroft's problem – given a set L of n lines and a set P of n points, decide if any of the points lies on any of the lines – can be reduced to the SSSP problem for segments, and Hopcroft's problem has an $\Omega(n^{4/3})$ lower bound [16] in a somewhat restricted model of computation.

References

1 Pankaj K. Agarwal, Matthew J. Katz, and Micha Sharir. Computing depth orders for fat objects and related problems. *Comput. Geom.*, 5:187–206, 1995. doi:10.1016/0925-7721(95)00005-8.

2 Boris Aronov, Mark de Berg, Esther Ezra, and Micha Sharir. Improved bounds for the union of locally fat objects in the plane. *SIAM J. Comput.*, 43(2):543–572, 2014. doi:10.1137/120891241.

3 Ivan J. Balaban. An optimal algorithm for finding segments intersections. In *Proc. 11th Symposium on Computational Geometry (SoCG)*, pages 211–219, 1995. doi:10.1145/220279.220302.

4 Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Comput. Geom.*, 48(4):360–367, 2015. doi:10.1016/j.comgeo.2014.12.003.

5 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.

6 Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. *J. ACM*, 57(3):16:1–16:15, 2010. doi:10.1145/1706591.1706596.

7 Timothy M. Chan and Zhengcheng Huang. Constant-hop spanners for more geometric intersection graphs, with even smaller size. In *Proc. 39th International Symposium on Computational Geometry (SoCG)*, volume 258 of *LIPIcs*, pages 23:1–23:16, 2023. doi:10.4230/LIPICS.SoCG.2023.23.

8 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In Seok-Hee Hong, editor, *Proc. 27th International Symposium on Algorithms and Computation (ISAAC)*, volume 64 of *LIPIcs*, pages 24:1–24:13, 2016. doi:10.4230/LIPICS.ISAAC.2016.24.

9 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in geometric intersection graphs. *J. Comput. Geom.*, 10(1):27–41, 2019. doi:10.20382/JOCG.V10I1A2.

10 Bernard Chazelle and Herbert Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992. doi:10.1145/147508.147511.

11 Mark de Berg and Sergio Cabello. An $O(n \log n)$ algorithm for single-source shortest paths in disk graphs, 2025. doi:10.48550/arXiv.2506.07571.

12 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications (3rd Edition)*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

13 Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986. doi:10.1137/0215023.

14 Alon Efrat, Alon Itai, and Matthew J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001. doi:10.1007/S00453-001-0016-8.

15 David Eppstein, Michael T. Goodrich, and Jonathan Z. Sun. Skip quadtrees: Dynamic data structures for multidimensional point sets. *Int. J. Comput. Geom. Appl.*, 18:131–160, 2008. doi:10.1142/S0218195908002568.

16 Jeff Erickson. New lower bounds for Hopcroft's problem. *Discret. Comput. Geom.*, 16(4):389–418, 1996. doi:10.1007/BF02712875.

17 Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987. doi:10.1007/BF01840357.

18 Chris Gray. *Algorithms for Fat Objects: Decompositions and Applications*. PhD thesis, TU Eindhoven, 2008. URL: https://pure.tue.nl/ws/portalfiles/portal/3256691/200811208.pdf.

19 Sariel Har-Peled. *Geometric approximation algorithms*, volume 173 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2011. doi:10.1090/surv/173.

20 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discret. Comput. Geom.*, 64(3):838–904, 2020. doi:10.1007/S00454-020-00243-7.

21 Katharina Klost. An algorithmic framework for the single source shortest path problem with applications to disk graphs. *Comput. Geom.*, 111:101979, 2023. doi:10.1016/j.comgeo.2022.101979.

22 Chih-Hung Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. *SIAM Journal on Computing*, 51(3):723–765, 2022. doi:10.1137/20M1388371.

23 Ketan Mulmuley. A fast planar partition algorithm, II. *J. ACM*, 38(1):74–103, 1991. doi:10.1145/102782.102785.

24 Liam Roditty and Michael Segal. On bounded leg shortest paths problems. *Algorithmica*, 59(4):583–600, 2011. doi:10.1007/S00453-009-9322-3.

Buffered Partially-Persistent External-Memory Search Trees

Gerth Stølting Brodal
Aarhus University, Denmark

Casper Moldrup Rysgaard
Aarhus University, Denmark

Rolf Svenning
Aarhus University, Denmark

Abstract

We present an optimal partially-persistent external-memory search tree with amortized I/O bounds matching those achieved by the non-persistent B^ε-tree by Brodal and Fagerberg [SODA 2003]. In a partially-persistent data structure, each update creates a new version. All past versions can be queried, but only the current version can be updated. Operations should be efficient with respect to the size N_v of the accessed version v. For any parameter $0 < \varepsilon < 1$, our data structure supports insertions and deletions in amortized $\mathcal{O}\big(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N_v\big)$ I/Os, where B is the external-memory block size. It also supports successor and range reporting queries in amortized $\mathcal{O}\big(\frac{1}{\varepsilon} \log_B N_v + K/B\big)$ I/Os, where K is the number of keys reported. The space usage of the data structure is linear in the total number of updates. We make the standard and minimal assumption that the internal memory has size $M \geq 2B$. The previous state-of-the-art external-memory partially-persistent search tree by Arge, Danner and Teh [JEA 2003] supports all operations in worst-case $\mathcal{O}(\log_B N_v + K/B)$ I/Os, matching the bounds achieved by the classical B-tree by Bayer and McCreight [Acta Informatica 1972]. Our data structure successfully combines buffering updates with partial persistence. The I/O bounds can also be achieved in the worst-case sense, by slightly modifying our data structure and under the requirement that the memory size $M = \Omega\big(B^{1-\varepsilon} \log_2(\max_v N_v)\big)$. For updates, where the I/O bound is $o(1)$, we assume that the I/Os are performed evenly spread out among the updates (by performing buffer-overflows incrementally). The worst-case result slightly improves the memory requirement over the previous ephemeral external-memory dictionary by Das, Iacono, and Nekrich (ISAAC 2022), who achieved matching worst-case I/O bounds but required $M = \Omega(B \log_B N)$, where N is the size of the current dictionary.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases B-tree, buffered updates, partial persistence, external memory

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.82

Related Version *Full Version*: `https://arxiv.org/abs/2503.08211` [13]

Funding All authors are funded by Independent Research Fund Denmark, grant 9131-00113B.

1 Introduction

Developing data structures for storing a set of keys from a totally ordered set subject to insertions, deletions, successor and predecessor queries, and range reporting queries is a fundamental problem in computer science. The classical solution in external-memory is the B-tree by Bayer and McCreight [5] which supports all the operations in worst-case $\mathcal{O}(\log_B N + K/B)$ I/Os, where N is the current size of the set, K is the number of reported keys, and B is the external-memory block size. While the B-tree achieves the optimal number of I/Os for queries, for any $0 < \varepsilon < 1$, the B^ε-tree by Brodal and Fagerberg [10] significantly improves update efficiency by attaching buffers to the internal nodes of a B-tree. This design supports updates with amortized $\mathcal{O}\big(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N\big)$ I/Os. The $\varepsilon B^{1-\varepsilon}$ factor improvement


© Gerth Stølting Brodal, Casper Moldrup Rysgaard, and Rolf Svenning;
licensed under Creative Commons License CC-BY 4.0

over traditional B-trees is significant when considering typical parameters of, e.g., $\varepsilon = 1/2$ and $B = 1000$ [4] and the B^ε-tree has found important applications in high-performance industry software such as TokuDB [9] and BetrFS [25].

Although the B^ε-tree optimizes update efficiency, it is *ephemeral*, like most dynamic data structures, meaning that each update overwrites the previous version, and only the current version can be queried. In many applications, maintaining access to past versions is beneficial or even essential. A *persistent* data structure supports such accesses, and in their seminal 1989 paper, Driscoll, Sarnak, Sleator, and Tarjan introduced general techniques for making ephemeral data structures persistent [20]. A *partially-persistent* data structure supports queries in all past versions of the data structure but only the current version can be updated. Multiple authors have adapted these techniques to the external-memory model, developing partially-persistent B-trees that support updates and queries in worst-case $\mathcal{O}(\log_B N_v + K/B)$ I/Os, where N_v is size of the accessed version v, matching the performance of classical B-trees [3, 6, 32].

In this paper, we present the first buffered partially-persistent external-memory search tree that retains the optimal update and query performance of the (ephemeral) B^ε-tree. Our approach combines buffering techniques, which are essential for efficient updates in external memory, with a geometric view of persistence.

1.1 The External-Memory Model

For problems on massive amounts of data that do not fit in internal memory, the standard model of computation is the I/O-model by Aggarwal and Vitter [1]. In this model, all computation occurs in an internal memory of size M, while an infinite external memory is used for storage. Data is transferred between internal and external memory in blocks of B consecutive elements, with each transfer counting as an I/O. The I/O complexity of an algorithm is defined as the total number of I/Os it performs, and the space usage is the maximum number of external-memory blocks used at any given time. The only operation we allow on stored keys are comparisons and we follow the standard assumption that the parameters $B \geq 2$ and $M \geq 2B$. Aggarwal and Vitter proved that the optimal bound for sorting in external memory is $\mathit{sort}(N) = \Theta\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os [1]. An algorithm is called *cache oblivious* if it is designed without explicit knowledge of B and M but is still analyzed in the I/O model for arbitrary values of these parameters, assuming an optimal offline cache replacement strategy [22]. Some authors make stronger assumptions on the size of the internal memory, such as the *tall-cache assumption* $M \geq B^{1+\delta}$, for some constant $\delta > 0$. For cache-oblivious algorithms, a tall-cache assumption is necessary to achieve optimal comparison-based external-memory sorting [11].

Considering the I/O-behavior of algorithms can be crucial in practice, as demonstrated by Streaming B-trees [8], the generation of massive graphs for the LFR benchmark [24, 27, 28], and the FlashAttention algorithm used in Transformer models [15].

1.2 Interface of a Partially-Persistent Search Tree

A partially-persistent search tree stores an ordered set of keys supporting the interface below (in our examples we use integers, but our data structure works for any totally ordered set). Each version is identified by a unique integer version identifier v, with zero being the initial version and the *current* version denoted by v_c. Further, we let $\mathcal{S}_v$ denote the set of keys contained in version v, and N_v the size of $\mathcal{S}_v$. Initially $v_c = 0$ and $\mathcal{S}_{v_c} = \emptyset$. Updates

(insertions and deletions) can only be performed on the current set $\mathcal{S}_{v_c}$, and any update advances the current version identifier, i.e., each version of the set $\mathcal{S}_v$ only differs from the previous version $\mathcal{S}_{v-1}$ by at most a single key. Queries can be performed on any version.

- INSERT(x) Creates $\mathcal{S}_{v_c+1} = \mathcal{S}_{v_c} \cup \{x\}$, increments v_c, and returns v_c.
- DELETE(x) Creates $\mathcal{S}_{v_c+1} = \mathcal{S}_{v_c} \setminus \{x\}$, increments v_c, and returns v_c.
- RANGE(v, x, y) Reports all keys in $\mathcal{S}_v \cap [x, y]$ in increasing order.
- SEARCH(v, x) Returns the successor of x in $\mathcal{S}_v$, i.e., $\min\{y \in \mathcal{S}_v \mid x \leq y\}$.

1.3 Previous Work

In internal memory, the *fat node* and *node copying* techniques can make any ephemeral linked data structure partially-persistent with constant overhead in both time and space, if the in-degree of each node in the ephemeral structure is constant [20]. Becker, Gschwind, Ohler, Seeger, and Widmayer [6] and Varman and Verma [32] adapted these techniques to B-trees in external-memory. An elegant application of partial persistence appears in the design of linear space planar point location data structures [31]. In this setting, the underlying set consists of segments which are partially ordered (only a pair of segments intersected by a vertical line can be compared). To adapt this approach to the external-memory setting, Arge, Danner, and Teh strengthened the partially-persistent B-tree to require only a total order on keys alive at any given version, leading to a static external-memory point-location structure [3].

A different approach to persistence is to interpret it geometrically, see Figure 1(left), modeling it as a data structure problem on a dynamic set of vertical segments. Kolovson and Stonebraker explored this perspective [26], though their reliance on R-trees led to poor performance guarantees [23]. More recently, Brodal, Rysgaard, and Svenning [12] leveraged this geometric approach to develop *fully persistent* B-trees, which allow both queries and modifications to all past versions in $\mathcal{O}(\log_B N_v)$ I/Os. In a fully persistent data structure, updating a version corresponds to cloning it and then applying the modification to the newly cloned version, ensuring that existing versions remain unaffected. Such behavior contrasts with *retroactive data structures* [17], where updates recursively propagate to cloned versions.

Concurrently with the work on persistent data structures in external-memory, there were significant improvements to external-memory data structures by leveraging buffering techniques to always process multiple updates and/or queries together. These include the Buffer Tree by Arge [2] which can form the basis for external-memory sorting, priority queues and batched dynamic algorithms [21] in amortized $\mathcal{O}\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os per operation. For a batched operation the answer might not be immediately returned, which is often sufficient, e.g., in many geometric plane-sweep algorithms where only the end result matters. For standard (non-batched) data structures, a line of work has investigated the update-query trade-off, beginning with the Buffered Repository Tree [14] performing updates in amortized $\mathcal{O}\left(\frac{1}{B}\log_B N\right)$ I/Os and queries in $\mathcal{O}(\log_2 N)$ I/Os. This was later generalized by the B^ε-tree [10] which corresponds to the Buffered Repository Tree for $\varepsilon \approx 0$ and to the standard B-tree for $\varepsilon \approx 1$. The amortized performance of the B^ε-tree was improved to high-probability [7] and worst-case [16] I/O bounds using stronger assumptions on the size of B and M (see Table 1).

1.4 Contribution

Combining the two lines of research on persistence and buffered data structures has remained an open challenge for the past 20 years, likely due to their seemingly conflicting principles. Persistence requires maintaining access to past versions without affecting their structure,

Table 1 Overview of results on the I/O complexity of ephemeral and partially-persistent search trees. Results marked by "am." hold amortized, and results marked by "rand." are randomized and hold with high probability. All other bounds are worst case. The parameter ε must satisfy $0 < \varepsilon < 1$. All results assume $M = \Omega(B)$, further † assumes $B = \Omega(\log N)$ and $M = \Omega\big(\max\{B \log^{\Theta(1)} N, B^2\}\big)$; ‡ assumes $M = \Omega(B \log_B N)$; and $*$ assumes $M = \Omega\big(B^{1-\varepsilon} \log_2(\max_v N_v)\big)$. For both queries and updates in [7, 16], we include the multiplicative dependency on $\frac{1}{\varepsilon}$ (that can be omitted when treating ε as a constant), allowing, for example, setting $\varepsilon = \frac{1}{\log_2 B}$. All ephemeral results use space linear in N and all partial persistence results use space linear in the total number of updates.

	Range Query	Update
Ephemeral		
Bayer and McCreigh [5]	$\mathcal{O}(\log_B N + K/B)$	$\mathcal{O}(\log_B N)$
Brodal and Fagerberg [10]	$\mathcal{O}\big(\frac{1}{\varepsilon} \log_B N + K/B\big)$ am.	$\mathcal{O}\big(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N\big)$ am.
Bender, Das, Farach-Colton, Johnson, and Kuszmaul† [7]	$\mathcal{O}\big(\frac{1}{\varepsilon} \log_B N + K/B\big)$	$\mathcal{O}\big(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N\big)$ rand.
Das, Iacono, and Nekrich‡ [16]	$\mathcal{O}\big(\frac{1}{\varepsilon} \log_B N + K/B\big)$	$\mathcal{O}\big(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N\big)$
Partial Persistent		
Becker, Gschwind, Ohler, Seeger, and Widmayer [6]	$\mathcal{O}(\log_B N_v + K/B)$	$\mathcal{O}(\log_B N_v)$
Varman and Verma [32]		
Arge, Danner, and Teh [3]		
This paper (Theorem 1)	$\mathcal{O}\big(\frac{1}{\varepsilon} \log_B N_v + K/B\big)$ am.	$\mathcal{O}\big(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N_v\big)$ am.
This paper (Theorem 2)*	$\mathcal{O}\big(\frac{1}{\varepsilon} \log_B N_v + K/B\big)$	$\mathcal{O}\big(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N_v\big)$
This paper (Theorem 3)	$\mathcal{O}\big(\frac{1}{\varepsilon} \log_B N_v + \gamma + K/B\big)$	$\mathcal{O}\big(\frac{1}{B^{1-\varepsilon}} \big(\frac{1}{\varepsilon} \log_B N_v + \gamma\big)\big)$
	$\gamma = sort\big(B^{1-\varepsilon} \log_2 N_v\big) = \frac{\log_2 N_v}{B^\varepsilon} \log_{M/B} \frac{\log_2 N_v}{B^\varepsilon}$	

while buffers essentially hold updates to past versions before applying them. Our work demonstrates that these two ideas can be effectively unified by developing partially-persistent external-memory search trees that achieve bounds matching those of ephemeral B^ε-trees. In Section 2 we prove the following theorem.

▶ **Theorem 1.** *Given any parameter $0 < \varepsilon < 1$ and $M \geq 2B$, there exist partially-persistent external-memory search trees over any totally ordered set, that support* Insert *and* Delete *in amortized $\mathcal{O}\big(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N_v\big)$ I/Os,* Search *in amortized $\mathcal{O}\big(\frac{1}{\varepsilon} \log_B N_v\big)$ I/Os, and* Range *in amortized $\mathcal{O}\big(\frac{1}{\varepsilon} \log_B N_v + K/B\big)$ I/Os. Here N_v denotes the number of keys contained in version v, and K the number of keys reported by* Range*. The space usage is linear in the total number of updates.*

Our construction is essentially a B^ε-tree with buffers of $\mathcal{O}(B)$ delayed updates at each internal node, leaves storing $\Theta\big(\frac{1}{\varepsilon} B \log_B N_v\big)$ updates, and where partially persistence is obtained using path copying of nodes, where an internal node is only copied when its buffer is empty.

The query Search can trivially also answer a member query "$x \in \mathcal{S}_v$?" by checking if Search(v, x) returns x. Our data structure also supports predecessor queries along with successor queries, as well as strict predecessor and successor queries, i.e., the returned key should be strictly smaller or larger than the query key x. The structure can also handle the case when $\mathcal{S}_0 \neq \emptyset$, where the initial structure can be constructed using $\mathcal{O}(1 + |\mathcal{S}_0|/B)$ I/Os (essentially this is Section 2.6, where a structure is constructed for a given sorted set of keys). Our data structure is stated as maintaining a set of keys, but it can easily be extended to

support dictionaries storing key-value pairs (each vertical segment in Figure 1 now stores a key-value pair, where the first axis is the key. Changing the value for key x at version v starts a new vertical segment at (x, v) with the new value).

In Section 3 we describe how to convert the amortized I/O bounds of Theorem 1 to worst-case bounds under the assumption that $M = \Omega\big(B^{1-\varepsilon}\log_2(\max_v N_v)\big)$ (Theorem 2), a weaker or equal assumption on the memory size than used in [7] and [16] for high-probability and worst-case bounds for B^ε-trees, respectively. Under the weakest assumption that $M \geq 2B$, we achieve the worst-case bounds in Theorem 3 with an additional term of at most $\mathcal{O}\big(sort\big(B^{1-\varepsilon}\log_2 N_v\big)\big)$ I/Os, where $B^{1-\varepsilon}\log_2 N_v$ is an upper bound on the number of buffered updates on a root-to-leaf path in our buffered B^ε-tree that should be flushed to a leaf. For updates, where the I/O bound can be $o(1)$, we assume that the I/Os are performed evenly spread out among the updates.

▶ **Theorem 2.** *Given any parameter $0 < \varepsilon < 1$ and $M = \Omega\big(B^{1-\varepsilon}\log_2(\max_v N_v)\big)$, there exist partially-persistent external-memory search trees over any totally ordered set, that support* Insert *and* Delete *in worst-case $\mathcal{O}\big(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B N_v\big)$ I/Os,* Search *in worst-case $\mathcal{O}\big(\frac{1}{\varepsilon}\log_B N_v\big)$ I/Os, and* Range *in worst-case $\mathcal{O}\big(\frac{1}{\varepsilon}\log_B N_v + K/B\big)$ I/Os. Here N_v denotes the number of keys contained in version v, and K the number of keys reported by* Range*. The space usage is linear in the total number of updates.*

▶ **Theorem 3.** *Given any parameter $0 < \varepsilon < 1$ and $M \geq 2B$, there exist partially-persistent external-memory search trees over any totally ordered set, that support* Insert *and* Delete *in worst-case $\mathcal{O}\big(\frac{1}{B^{1-\varepsilon}}\big(\frac{1}{\varepsilon}\log_B N_v + \gamma\big)\big)$ I/Os,* Search *in worst-case $\mathcal{O}\big(\frac{1}{\varepsilon}\log_B N_v + \gamma\big)$ I/Os, and* Range *in worst-case $\mathcal{O}\big(\frac{1}{\varepsilon}\log_B N_v + \gamma + K/B\big)$ I/Os, where $\gamma = sort\big(B^{1-\varepsilon}\log_2 N_v\big)$. Here N_v denotes the number of keys contained in version v, and K the number of keys reported by* Range*. The space usage is linear in the total number of updates.*

Note that, for example, when $N_v = 2^{\mathcal{O}(B^\varepsilon)}$ then $B^{1-\varepsilon}\log_2 N_v = \mathcal{O}(B)$ and $\gamma = \mathcal{O}(1)$ and the I/O bounds of Theorem 3 match those of Theorem 2, with only the assumption $M \geq 2B$. This observation can be further strengthened, as when $\gamma = \mathcal{O}\big(\frac{1}{\varepsilon}\log_B N_v\big)$ the I/O bounds match similarly, which holds when $N_v = 2^{B^\varepsilon\left(\frac{M}{B}\right)^{\mathcal{O}\left(\frac{B^\varepsilon}{\varepsilon\log_2 B}\right)}}$.

Outline of Data Structure. Previous work on partially-persistent search trees in external memory directly adapted the general pointer-based transformations for persistence [20]. In contrast, our approach embraces the geometric interpretation of partial persistence like in [12], where the state of the data structure is embedded in a two-dimensional plane with keys on the first axis and versions on the second axis, see Figure 1(left). Under this interpretation, each update corresponds to the start or end of a vertical segment in the plane. Since partially-persistent updates are applied to the current version, it always affects the top of the plane. Successor and predecessor queries correspond to horizontal ray shooting to the right and left, respectively, and range queries to reporting the intersections between a horizontal query segment among vertical segments.

To efficiently update and query the geometric view, we partition the plane into rectangles, each containing $\Theta\big(\frac{1}{\varepsilon}B\log_B \overline{N}\big)$ vertical segments in lexicographic order. For now, we assume that all versions have size $\Theta\big(\overline{N}\big)$, for a fixed $\overline{N}$. This assumption is lifted using global rebuilding, see Section 2.6. In the geometric persistent view, a vertical segment crossing multiple rectangles is split into multiple smaller segments, one for each rectangle, and each smaller segment is inserted into one rectangle.

At a high level, our data structure is divided into two parts, see Figure 1(right). The top part consists of all the open rectangles containing the current set $\mathcal{S}_{v_c}$, which may still be updated. The entry point of this data structure is a B^ε-tree T on the key axis to facilitate

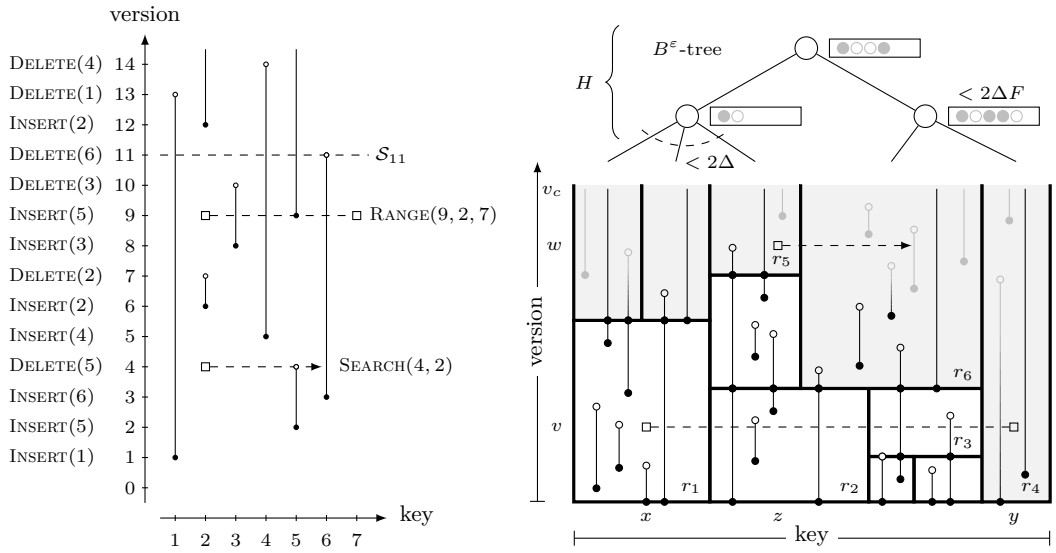


Figure 1 (Left) A list of updates performed on an initially empty set and the geometric interpretation of the updates. Vertical lines illustrate the half-open intervals of versions containing a key. Note that the key 3 is contained in versions $[8, 10[$, i.e., versions 8 and 9, whereas the key 2 is contained in version 6 and versions $[12, \infty[$. The topmost dashed line shows that version 11 of the set is $\mathcal{S}_{11} = \{1, 4, 5\}$, the dashed line segment at version 9 shows that the result of the query $\textsc{Range}(9, 2, 7)$ is $\{3, 4, 5, 6\}$, and the bottommost dashed arrow shows that the result of the successor search $\textsc{Search}(4, 2)$ is 6. (Right) A B^ε-tree of the open rectangles and below the geometric interpretation of the updates, split into multiple rectangles, where gray rectangles are open rectangles. The black endpoints represent updates present in the rectangles and the gray endpoints represent buffered updates present in the buffers at the internal nodes of the B^ε-tree. A query $\textsc{Range}(v, x, y)$ is represented as the dashed line between two square endpoints, spanning rectangles r_1, r_2, r_3, and r_4, and a successor query $\textsc{Search}(w, z)$ is represented as the dashed arrow from a square endpoint, spanning two rectangles r_5 and r_6. Black dots on vertical segments correspond to the upper endpoint of the segment in the rectangle below and the lower endpoint of the segment in the rectangle above.

buffered updates and to find the relevant rectangle(s) for updates and queries. Since updates are buffered, the geometric view stored in the rectangles may be incomplete, since buffered updates (segment endpoints) will first be added to the rectangle when buffers are flushed. The bottom part consists of all the finalized rectangles, i.e., rectangles which can be queried but not updated. The entry point to the bottom part is a point location data structure P to find the relevant rectangle(s) for a query. We implement P as an external-memory adaption of the classical planar point location solution using partial persistence [3, 31], more specifically a B-tree with path copying during updates.

2 The Buffered Persistent Data Structure

In this section, we describe our partially-persistent B^ε-tree structure. Versions are identified by the integers $0, 1, 2, \ldots$, where v_c denotes the identifier of the current version. We let $\mathcal{S}_v$ denote the set at version v, where keys are from some totally ordered set. The initial set $\mathcal{S}_0 = \emptyset$, and $\mathcal{S}_{v+1} = \mathcal{S}_v \cup \{x\}$ if the $v+1$'th update is $\textsc{Insert}(x)$, and $\mathcal{S}_{v+1} = \mathcal{S}_v \setminus \{x\}$ if the $v+1$'th update is $\textsc{Delete}(x)$. Note that $\mathcal{S}_{v+1} = \mathcal{S}_v$ if the $(v+1)$'th update inserts a key already in $\mathcal{S}_v$ or is deleting a key not in $\mathcal{S}_v$.

2.1 Geometric Interpretation of Partial Persistence

The problem has a natural geometric interpretation in a two-dimensional space, with the first dimension representing the keys and the second dimension representing the versions, see Figure 1(left). On this two-dimensional plane, a key x existing in a half-open interval of versions $[v, w[$, is represented by the vertical line segment $\{x\} \times [v, w[$, i.e., x is inserted in version v and deleted in version w. If $x \in \mathcal{S}_{v_c}$, then $w = +\infty$ (x has not been deleted yet).

2.2 Partitioning the Plane into Rectangles

We consider the sequence of versions partitioned into intervals $[v_0, v_1[, \ldots, [v_{k-1}, v_k[, [v_k, \infty[$, for some versions $0 = v_0 < v_1 < \cdots < v_k \leq v_c$. In Section 2.6 we show how to maintain the version intervals. We let $\overline{v} = v_k$ and $\overline{N} = |\mathcal{S}_{\overline{v}}|$. In the following, we consider the interval $[\overline{v}, \infty[$, which contains the current version v_c of the set. We allow up to $\alpha \cdot \overline{N}$ partially-persistent updates during this interval of versions for a constant $0 < \alpha < 1$, i.e., all versions v, $\overline{v} \leq v \leq v_c$, satisfy $(1-\alpha) \cdot \overline{N} \leq |\mathcal{S}_v| \leq (1+\alpha) \cdot \overline{N}$.

Our data structure is built around four parameters:

$$\Delta = \lceil B^{\varepsilon} \rceil \qquad H = 1 + \lceil \log_{\Delta} \overline{N} \rceil \qquad F = \lceil B^{1-\varepsilon} \rceil \qquad R = H \cdot 2\Delta \cdot F$$

The basic idea is to have a B^{ε}-tree T of degree at most $2\Delta - 1$ (and degree at least Δ, if only insertions can be performed, and degree at least 1 if deletions are allowed), where leaves (open rectangles) store between $4R$ and $10R$ updates (see Section 2.4) and all leaves are at the same level of T. In Section 2.5 we prove that H is an upper bound on the height of T (number of nodes on a root-to-leaf path, excluding the leaves). Each internal node of T will have a buffer of at most $2\Delta F = \Theta(B)$ updates yet to be applied to the leaves of the subtree rooted at the node. Note that R is an upper bound on the total number of buffered updates along a root-to-leaf path in T. The essential property of the parameters is that $R/B = \Theta(H) = \Theta(\frac{1}{\varepsilon} \log_B \overline{N})$.

The geometric plane defined in Section 2.1 is partitioned into a set of axis aligned rectangles $[x, y[\times [v, w[$, such that the number of updates in each rectangle is $\Theta(R)$. For each rectangle we store a list of the updates in the rectangle in lexicographical order by first the key and secondly the version of that update. Note that equal keys are group consecutively in the list. To allow efficiently locating a rectangle for a given version and key, we store a list indexed by version identifier, where we for each version v store a pointer to the root of a B-tree P_v over the rectangles left-to-right containing $\mathcal{S}_v$ (see Section 2.4). The tree P_v has degree $\Theta(B)$ and does not store buffered updates. Further, we require that each rectangle contains $\Omega(R)$ keys which are present in all versions the rectangle spans. We denote such a key as *spanning*. If $\overline{N} = \mathcal{O}(R)$, all updates are stored in a single list.

New updates are buffered, to achieve I/O efficient update bounds. The topmost rectangles, which cover the current version v_c, are all *open*, with all other rectangles being *closed*. Crucially, new updates are always performed in the current version. We maintain the invariant that for a buffered update, i.e., an update not having been flushed to the corresponding rectangle yet, the rectangle must be open.

For the open rectangles, we store a B^{ε}-tree T, such that recent updates to the open rectangles are buffered. We let the maximum degree of an internal node in T be $2\Delta - 1$. Each internal node of T contains a buffer of up to $2\Delta F$ updates, sorted lexicographically by key and version. Additionally, each update stores if the update is an insertion or deletion. Consider a full buffer, i.e., it contains at least $2\Delta F$ updates, where each update should be *flushed* to one of the at most $2\Delta - 1$ children. Then, there must exist a subset of least F updates that should be flushed to the same child.

The setup is illustrated on Figure 1(right). The vertical black and gray lines represent the version intervals containing a key. The black lines represent updates present in the list of updates contained in that rectangle, while the gray lines and endpoints represent updates contained in buffers of T, which are illustrated at the top of the figure.

2.3 Handling Queries and Updates

When performing $\textsc{Search}(v, x)$, first the rectangle r covering point (x, v) in the plane must be found. By using the B-tree P_v associated with version v, r can be found using $\mathcal{O}(H)$ I/Os. If r is closed, then all updates inside r are contained in the sorted list of updates stored in r, and these can be scanned in $\mathcal{O}(R/B)$ I/Os. If r is open, then the result of the successor query may be affected by buffered updates, which are not stored in r. The rectangle r must therefore be *actualized*, by merging all updates in buffers on the path from the root of T to r with the updates in r. The details of this operation are described in Section 2.4, where the actualize operation is shown to take amortized $\mathcal{O}(H)$ I/Os. After r is actualized, the query continues by scanning the updates of r. If the result of the successor query is not contained in the rectangle r, then by the spanning requirement, the result of the query must be in the neighboring rectangle to the right, that similarly is actualized if it is open. In total, the operation spends amortized $\mathcal{O}(H + R/B) = \mathcal{O}\big(\frac{1}{\varepsilon}\log_B \overline{N}\big)$ I/Os. Note that the operation easily can be modified to support member, predecessor, and strict predecessor or successor queries.

A $\textsc{Range}(v, x, y)$ query is performed very similarly to a $\textsc{Search}$ query. The query may however touch more than two rectangles. Note that for the at most two rectangles containing the endpoints of the query we do not necessarily report all keys they contain at version v. These rectangles can be found using amortized $\mathcal{O}(H + R/B)$ I/Os by the argument above. For each intermediate rectangle accessed (and possibly actualized if it is open), then by the spanning requirement, a constant fraction of the updates scanned result in reported keys. Since accessing a rectangle takes amortized $\mathcal{O}(H + R/B)$ I/Os, and each rectangle contains $\Theta(R) = \Theta(B \cdot H)$ keys at version v, then amortized $\mathcal{O}(1/B)$ I/Os are spent for each key reported for an intermediate rectangle. In total, a $\textsc{Range}$ query reporting K keys takes amortized $\mathcal{O}(H + R/B + K/B) = \mathcal{O}\big(\frac{1}{\varepsilon}\log_B \overline{N} + K/B\big)$ I/Os.

Each update, either an $\textsc{Insert}$ or $\textsc{Delete}$, is applied to the current version v_c of the set. The B^ε-tree T contains all buffered updates to the open rectangles, which cover the current version v_c. For an update operation, a tuple with the update and v_c is added to the root buffer of T, which is stored in internal memory. In Section 2.4 it is shown that adding the update to the root buffer and handling possible *buffer overflows* takes amortized $\mathcal{O}(H/F) = \mathcal{O}\big(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B \overline{N}\big)$ I/Os.

2.4 Flushing Buffers

To argue about the amortized cost of flushing the content of buffers down the tree T, we let the potential of each buffered update be $1/F$ multiplied by the height of the buffer the update is stored in, with the root buffer being at the largest height. One unit of released potential can cover $\mathcal{O}(1)$ I/Os. When adding an update to the tree, the root buffer is always stored in internal memory, and therefore no I/Os are needed to access it. However, the potential is increased by at most $1/F \cdot H$, and the operation therefore uses amortized $\mathcal{O}(H/F)$ I/Os.

Buffer Overflows. Each buffer at an internal node of T contains at most $2\Delta F$ updates. If a buffer contains more than $2\Delta F$ updates, then a *buffer overflow* is performed. Since each node of T has at most $2\Delta - 1$ children, at least F updates from the buffer must be to the same child. These updates can be moved together to the buffer of that child.

A buffer overflow can happen in two cases. Either when an update is placed into the root buffer as the result of an update operation, or when updates are placed into a buffer because the parent buffer is overflowing. Moving exactly F updates out of a buffer, is always sufficient to make an overflowing buffer non-overflowing again. A buffer overflow therefore only moves down a single path of T.

An overflowing buffer can be stored in $\mathcal{O}(1)$ blocks, since $2\Delta F + F = \mathcal{O}(B)$, and therefore the F updates to move can be found in $\mathcal{O}(1)$ I/Os. If the overflowing updates are moved to a child buffer, these can be inserted via a merge in $\mathcal{O}(1)$ I/Os. As F updates are moved one level down the tree, then the potential decreases by 1, which is enough to cover the $\mathcal{O}(1)$ I/O cost of the overflow operation.

If the child is not an internal node of T, but an open rectangle, then merging the F overflowing updates into the list of updates in the rectangle takes $\mathcal{O}(R/B) = \mathcal{O}(H)$ I/Os. As buffer overflows only move down a single path of the tree, then $\Omega(H)$ overflows must have occurred before the overflow reaches the open rectangle. Merging the overflow into the list of updates in the open rectangle therefore does not increase the asymptotic amortized number of I/Os performed.

Actualizing. When *actualizing* an open rectangle, all buffered updates to that open rectangle must be moved into the rectangle. Note that all relevant updates are in the buffers on the root-to-leaf path in T to the open rectangle. Each of these buffers contains at most $2\Delta F = \mathcal{O}(B)$ buffered updates. The total number of buffered updates on the path is at most $H \cdot 2\Delta \cdot F = R$. For each node on the path from the root down to the rectangle the following is done. Let U be the updates on the path from the levels above in sorted order. Initially, U is empty. To extend U for each level top-down, U is merged with the relevant updates of the next buffer. This requires $\mathcal{O}(1 + |U|/B)$ I/Os, by scanning U and the buffer. Since the U updates are moved one level down, they release potential $|U|/F \geq |U|/B$, that can cover $\Theta(|U|/B)$ I/Os, i.e., the amortized cost for actualizing one level of the tree is $\mathcal{O}(1)$ I/Os. As there are $\mathcal{O}(H)$ levels of the tree, the at most R relevant updates can be found in sorted order in amortized $\mathcal{O}(H)$ I/Os. They can then be merged with the updates in the open rectangle in $\mathcal{O}(R/B)$ I/Os. In total, the actualize operation requires amortized $\mathcal{O}(H + R/B) = \mathcal{O}(H)$ I/Os.

Finalizing. Each open rectangle is allowed to receive between R and $2R$ updates before it is *finalized*, converting it into a closed rectangle. When finalizing an open rectangle, all buffered updates to the rectangle are removed from T and merged with the rectangle, to ensure that all buffered updates in T are only to open rectangles. We now argue that open rectangles receive at most $2R$ updates in total by finalizing the rectangle as soon as R updates have been added to it. A finalize operation can be triggered from an actualize operation or from a buffer overflow. In both cases the number of updates in the rectangle before the operation is at most $R - 1$. An actualize operation may add at most R buffered updates to a rectangle, i.e., at most $2R - 1$ total updates are placed in a finalized rectangle. If the rectangle receives an update from a buffer overflow, then the overflow must have been triggered by an update in the root buffer. Buffered updates to add to the rectangle can only be the R updates in buffers on the path, and the new update, which in total is at most $(R - 1) + R + 1 = 2R$ updates to add to the open rectangle. Thus, by finalizing a rectangle as soon as it receives at least R updates, it will receive between R and $2R$ updates.

Spanning Requirement. We require that the first version of a rectangle contains $[4R, 8R[$ keys. When finalizing a rectangle, $[R, 2R]$ updates have been performed and therefore at least $2R$ of the initial keys are still present, that is, span all versions of the rectangle. This ensures that the $\Omega(R)$ spanning keys requirement is met. When finalizing a rectangle, $[2R, 10R[$ keys are contained in the rectangle at version v_c. New open rectangles must be created to span the key range of the closed rectangle, containing the keys present at version v_c. If $[4R, 8R[$ keys are present at version v_c, then a single rectangle suffices. If $[8R, 10R[$ keys are present at version v_c, then the range is *split* in two rectangles at the median key, both with $[4R, 5R]$ keys, and T must be updated as described below. Otherwise, at version v_c the rectangle contains $[2R, 4R[$ keys. In this case a sibling rectangle is finalized, to allow for a *merge* of two rectangles. The sibling rectangle holds $[2R, 10R[$ keys at version v_c. Concatenation the keys present at version v_c from the two closed rectangles gives a sorted list of $[4R, 14R[$ keys for a new open rectangle. A split may need to be performed if there are $8R$ or more keys, i.e., the result is one or two new open rectangles.

Updating the B^ε-Tree T. After finalizing open rectangles, the B^ε-tree must be updated accordingly. If an open rectangle was split, then a new child is added to the parent node in T of the updated rectangle. If this increases the degree of the node to 2Δ, it is split by distributing its children into two new nodes, each with degree Δ. Its buffer is also partitioned so that each buffered update is placed in the buffer containing its relevant child. Splitting the node further introduces a new child to the parent of the node. Note that this may cascade up the tree, but only on the path towards the root. If a merge of the rectangle occurs, then a child is deleted from the parent. When merging rectangles, the merged rectangles must be siblings in the tree. The rectangle is merged with the left or right neighbor rectangle, which has a nearest common ancestor with the rectangle in T, to ensure that the key range of existing nodes only increase. This may cause the degree of nodes to be below Δ. Notably, we do not merge internal nodes of T, as this could create a large buffer that requires multiple flushes in different directions, known as *flushing cascades* [7]. We instead allow nodes to have a degree down to one, where deleting the last child results in deleting the path of consecutive degree-one from the child towards the root. As we show in Section 2.5, this does not affect the asymptotic height of the tree.

Updating the rectangles and the B^ε-tree T upon finalizing therefore requires scanning $\mathcal{O}(R)$ keys and traversing a constant number of paths of length $\mathcal{O}(H)$ in T, which takes $\mathcal{O}(R/B + H) = \mathcal{O}(R/B)$ I/Os. As $\Omega(R)$ updates must be applied to a rectangle before finalizing it, this does not increase the asymptotic amortized cost of an update operation. Queries may also finalize rectangles, but already use amortized $\mathcal{O}(H)$ I/Os, causing no asymptotic query overhead.

Recall, that we maintain a list that for any version v stores a pointer to the root of a B-tree P_v over the rectangles left-to-right for version v. If the set of rectangles is unchanged from version $v_c - 1$ to v_c, we have $P_{v_c} = P_{v_c-1}$. If a rectangle is finalized during version v, the set of rectangles changes and we need to create a new P_{v_c} from P_{v_c-1}. Since only $\mathcal{O}(1)$ rectangles are removed and created in the new version, we can create P_{v_c} from P_{v_c-1} partially persistently by standard B-tree updates and rebalancing using naive *path copying* [31], using $\mathcal{O}(\log_B(N_v/R)) = \mathcal{O}(H)$ I/Os and $\mathcal{O}(H) = \mathcal{O}(R/B)$ blocks of space.

Space Usage. When finalizing a rectangle, $\Omega(R)$ updates must have occurred in that rectangle. New rectangles are then created, which in total copies $\mathcal{O}(R)$ updates, and one path of the tree is copied. As the height of the tree is at most $H = \mathcal{O}(R/B)$, and the updates

of the rectangles are stored in lists, the newly allocated space is $\mathcal{O}(R/B)$, which can be amortized over the $\Omega(R)$ updates required for the finalization to happen. In addition to the updates, initially $\overline{N}$ keys are stored across $\mathcal{O}\big(\overline{N}/R\big)$ rectangles in lists, and an initial balanced B^ε-tree is built on these initial rectangles, causing an initial space of $\mathcal{O}\big(\overline{N}/B\big)$ blocks. As the structure allows for at most $\alpha \cdot \overline{N}$ updates, the total space usage is $\mathcal{O}\big(\overline{N}/B\big)$ blocks.

2.5 Bounding the Tree Height

In this section we show that H is an upper bound on the height of the B^ε-tree T.

We define the *weight* of a node at height i in T to be the number of updates on keys in the key range of the node, and let w_i be a lower bound for the weight of a node at height i. The rectangles are at height 0 of the tree, with the nodes of the tree starting at height 1. The updates are both the $\overline{N}$ initial keys as well as the up to $\alpha \cdot \overline{N}$ additional updates. It holds that the weight of a node is the sum of the weights of its children. By induction on the number of updates we show that $w_i \geq B\Delta^i$ for all nodes at all heights, except for the root. First note that the inequality holds for $i = 0$, as any rectangle contains at least $4R \geq B$ updates when it was created. Initially, $\overline{N}$ updates are distributed into at most $\overline{N}/(4R)$ rectangles, where the number of updates in each rectangle is at least $4R \geq B$. Each internal node initially has degree $[\Delta, 2\Delta[$, except for the root that has degree $[2, 2\Delta[$. By induction on the tree height i, it holds that the initial tree satisfies $w_i \geq B\Delta^i$, except for the root. Each update affects a root-to-leaf path of the tree. If the tree is not updated, then the weights of the nodes on the path can only grow, and therefore the inequality holds. If rectangles are merged, then one rectangle disappears together with all the ancestors having only this single rectangle as a leaf. The other rectangle and its ancestors up to the least common ancestor of the two merged leaves get their key ranges expanded. It follows that the surviving nodes after merging rectangles only can have their key range increase, and therefore the inequality holds. If a split occurs in any node at height i, then the degree of the node before the split is 2Δ. The node is split into two nodes at height i, each with Δ children. The weight of each of the two nodes is therefore at least $\Delta \cdot w_{i-1} \geq \Delta \cdot B\Delta^{i-1} = B\Delta^i$. Therefore, it holds that $w_i \geq B\Delta^i$.

Since the number of updates is at most $(1+\alpha) \cdot \overline{N}$, we have $B\Delta^i \leq (1+\alpha) \cdot \overline{N}$ for all nodes at height i, except for the root. Since by definition $2 \leq \Delta \leq B$ and $\alpha < 1$, we have $\Delta^{i+1} \leq 2\overline{N}$, i.e., $i \leq \log_\Delta\big(2\overline{N}\big) - 1 \leq \log_\Delta \overline{N}$. The height of T is then at most the largest value of i satisfying this inequality, plus one for the root, i.e., the height of T is at most $1 + \log_\Delta \overline{N} \leq 1 + \lceil \log_\Delta \overline{N} \rceil = H$.

2.6 Global Rebuilding

The data structure above allows for an initial set of $\overline{N}$ keys to receive up to $\alpha \cdot \overline{N}$ persistent updates, for a constant $0 < \alpha < 1$. For any version v, we have $(1-\alpha) \cdot \overline{N} \leq N_v \leq (1+\alpha) \cdot \overline{N}$, i.e., $N_v = \Theta\big(\overline{N}\big)$. Therefore, the asymptotic costs of all operations also hold with $\overline{N}$ replaced by N_v.

To allow for more than $\alpha \cdot \overline{N}$ updates, we create multiple copies of the data structure above using global rebuilding [29, 30]. Whenever the current data structure reaches $\alpha \cdot \overline{N}$ updates, a new data structure is created with initial set $\mathcal{S}_{v_c}$ and $\overline{N}_{\text{new}} = |\mathcal{S}_{v_c}|$ (and new H and R parameters), with a new set of rectangles and a new B^ε-tree T, where all buffers are empty. We compute $\mathcal{S}_{v_c}$ by performing $\textsc{Range}(v_c, -\infty, \infty)$ in amortized $\mathcal{O}\big(\frac{1}{\varepsilon}\log_B \overline{N} + K/B\big) = \mathcal{O}\big(\overline{N}/B\big)$ I/Os. This actualizes all open rectangles which now can be closed. The new data

structure can be build using $\mathcal{O}(\overline{N}/B)$ I/Os by a single scan of the sorted list containing $\mathcal{S}_{v_c}$. In the old data structure $\alpha \cdot \overline{N}$ updates have been performed before this rebuild is performed. By amortizing the rebuild cost over these updates, the amortized cost of each update is increased by $\mathcal{O}(1/B)$ I/Os, i.e., the asymptotic amortized cost of an update is not increased. As the space usage of the new data structure is $\mathcal{O}(\overline{N}/B)$ blocks, a similar argument can be used to amortize the space usage over the updates, maintaining a linear space usage in the total number of updates. This concludes the proof of Theorem 1.

3 Worst-Case Bounds

In this section, we describe how to achieve worst-case I/O guarantees instead of amortized under progressively weaker assumptions on the internal memory size M. Previous approaches to improving the amortized performance of ephemeral B^ε-trees, both in the randomized [7] and worst-case [16] setting, assumed at least that $M = \Omega(\frac{1}{\varepsilon} B \log_B \overline{N}) = \Omega(HB)$, which allows all buffers on a path to be sorted in internal memory, i.e., $\mathcal{O}(sort(HB)) = \mathcal{O}(H)$ I/Os. First, in Section 3.1, we show that if $M = \Omega(HB)$, our persistent structure can be deamortized without asymptotic overhead, by applying incremental global rebuilding for handling changing $\overline{N}$ and performing the flushing along a root-to-leaf path incrementally. Then, in Section 3.2, we relax the assumption to $M = \Omega(B^{1-\varepsilon} \log_2 \overline{N})$ (Theorem 2). We apply the *subtracting game* studied by Dietz and Raman [18] to argue that the buffer of a node can at most store $\mathcal{O}(F \log_2 \Delta)$ updates towards each child, if we always recursively flush F updates to the child with most updates when a node receives F updates from its parent. This represents an improvement by a factor $B^\varepsilon / \log_2 B$ on the assumption for the size of the internal memory. Finally, in Section 3.3, we show worst-case results when only assuming $M \geq 2B$, which introduces a small additive overhead on all operations (Theorem 3). We employ *the zeroing* game by Dietz and Sleator [19, Theorem 5] to avoid a multiplicative overhead for RANGE queries, by incrementally splitting and merging the open rectangle with the most updates.

3.1 Large Internal Memory Assumption

We first consider the case when $M = \Omega(HB)$. When actualizing a rectangle (see Section 2.3), the buffers at the $\mathcal{O}(H)$ nodes along the root-to-leaf path, with a total size of $\mathcal{O}(HB)$, are merged to produce a sorted list of updates to apply to the rectangle. As shown in Section 2.4, this can be done in amortized $\mathcal{O}(H)$ I/Os, by merging the buffers top-down. In the worst case, this requires $\mathcal{O}(H^2)$ I/Os. By instead merging the buffers using an external memory sorting algorithm, the worst-case number of I/Os can be improved to $\mathcal{O}(sort(HB))$. Previous approaches to improving the amortized performance of B^ε-trees in the randomized [7] and worst-case [16] settings both assumed at least that $M = \Omega(HB)$, in which case the sorting term trivially disappears by performing the sorting internally after reading the H buffers into internal memory. The remaining challenge was handling flushing cascades, which occur when merging internal nodes of T results in large buffers requiring many flushes in different directions. For our structure, we avoid this issue, by never merging internal nodes, and instead maintain the height of T using global rebuilding. For the remainder of this section, we assume a large internal memory of size $M = \Omega(HB)$ and describe how to achieve worst-case guarantees by incrementally performing amortized work.

Queries. Finding and actualizing a relevant rectangle for a query takes $\mathcal{O}(H + sort(HB)) = \mathcal{O}(H)$ I/Os when $M = \Omega(HB)$. The worst case for a SEARCH and RANGE query is therefore $\mathcal{O}(H)$ and $\mathcal{O}((1 + K/R)\,H) = \mathcal{O}(H + K/B)$ I/Os, respectively. Note that for a RANGE query, for each rectangle that intersects the query, except for the leftmost and rightmost ones, $\Omega(R)$ keys are reported due to the $\Omega(R)$ spanning keys in each rectangle.

Updates. When performing an update, it may be the $\lceil \alpha \cdot \overline{N} \rceil$'th update, which triggers a global rebuild of the structure based on a new $\overline{N}$, which uses $\mathcal{O}(\overline{N}/B)$ I/Os, as described in Section 2.6. However, by performing the global rebuilding incrementally [29, 30] over the next $\Theta(\overline{N})$ updates, this does not increase the asymptotic worst-case number of I/Os of each update. While initializing the new structure there are still updates happening which must then be applied before it can take over. By performing updates to the new structure at a sufficiently fast rate compared to the live structure this ensures that they stay within a constant factor of each other in size until the new structure takes over. Therefore, only the I/O cost of an update without global rebuilding needs to be considered.

Updates are inserted in the root buffer of the B^ε-tree, as described in Section 2.4. By keeping the root buffer in internal memory this uses no I/Os. If the root buffer overflows, it may cause buffer overflows along a root-to-leaf path down to an open rectangle, which may then be finalized by performing an actualize operation followed by a path copy. The update therefore requires $\mathcal{O}(H)$ I/Os in total under the large internal memory assumption. However, each time the root buffer overflows, F updates are removed from it, meaning this occurs at most every Fth update. Thus, when the root buffer overflows, we incrementally apply the update to the structure over the next F updates, ensuring that $\mathcal{O}(H/F)$ I/Os are performed per update in the worst case. To not interfere with the incremental work, we place new updates in a separate buffer while it is in progress and merge them with the root buffer when it is finished. If a path copy has occurred, the root pointers of the F most recent versions must be updated to the new root. Since they are stored together in an array indexed by their version identifier this takes $\mathcal{O}(1)$ I/Os. If a query occurs while an update is being performed incrementally, we complete the update before executing the query. This does not increase the asymptotic worst-case number of I/Os for queries.

3.2 Smaller Buffers on All Paths Using the Subtraction Game

In the previous section, we showed that there is no overhead on worst-case queries and updates compared to the amortized bounds, if the internal memory can hold all keys on a root-to-leaf path towards the same open rectangle. To lower the possible number of such keys, we slightly change the flushing strategy described in Section 2.4 where we only performed a flush when a buffer overflowed. Instead, for every F'th update, we flush along an entire root-to-leaf path, always flushing towards the child where most of the updates are going. We still flush at most F keys, which preserves the property that internal nodes of T contain at most $2\Delta F$ updates. In the following, we show that this flushing strategy guarantees that all buffers contain $\mathcal{O}(F \log_2 \Delta)$ updates going towards the same child, and therefore also the same leaf. This implies that the assumption $M = \Omega(HF \log_2 \Delta) = \Omega(B^{1-\varepsilon} \log_2 \overline{N})$ is sufficient to achieve no overhead for worst-case queries and updates. This is a factor $\Theta(B^\varepsilon / \log_2 B)$ improvement over the previous smallest assumption on M [16].

We can view each node as playing the *subtracting game* studied by Dietz and Raman [18] for the number of updates $x_1, x_2, x_3, ..., x_{2\Delta-1}$ going towards each of its at most $2\Delta - 1$ children. When at most F updates are flushed towards a node, and δ_i new updates are going

towards the ith child, then variable x_i is increased by δ_i. We flush towards the child j where most of the updates are going which sets the variable $x_j = \max\{x_j - F, 0\}$. Following [18, Theorem 3], scaled by a factor of F, this guarantees $x_i = \mathcal{O}(F \log_2 \Delta)$ for any i.

We also need to consider how merging and splitting nodes in T impacts the subtraction games. Only leaves of T, corresponding to open rectangles, are merged. When an internal node is split, it corresponds to evenly distributing the x_i variables from one game to two new games, except for one variable that is split into two new variables, each with a smaller or equal value. When a leaf, i.e., an open rectangle, is merged or split, the one or two rectangles involved are first actualized, which sets their variables to zero, a stronger operation than subtracting. Thus, the variable for a new rectangle is always zero and variables on root-to-leaf paths to actualized rectangles may be decremented. In all cases and for all games, variables are either decremented without adding to the game, or a copy of an existing game is created, where all variables in the copy are equal or smaller in value than before. This concludes the proof of Theorem 2.

3.3 Improving Worst-Case Range Queries Using the Zeroing Game

In this section, we consider the small-memory setting with $M \geq 2B$, to overcome the theoretical limitation of the memory assumptions made in Sections 3.1 and 3.2. Actualizing a rectangle by merging the relevant updates on a root-to-leaf path to a rectangle requires $\mathcal{O}(H + \gamma)$ total I/Os, where $\gamma = \mathit{sort}\big(B^{1-\varepsilon} \log_2 \overline{N}\big)$. The construction from the previous section directly results in worst-case Search queries using $\mathcal{O}(H + \gamma)$ I/Os and updates using $\mathcal{O}\big(\frac{1}{F}(H + \gamma)\big)$ I/Os. However, since Range queries are performed by repeatedly searching for the $\Theta(1 + K/R)$ rectangles intersecting the query, the worst-case number of I/Os is $\Theta((1 + K/R)(H + \gamma)) = \Theta\Big(H + \gamma + \frac{K}{B}\Big(1 + \frac{\varepsilon \log_2 B}{B^\varepsilon} \log_{M/B}\big(B^{-\varepsilon} \log_2 \overline{N}\big)\Big)\Big)$, notably with a multiplicative non-constant overhead on the reporting term. In this section, we describe how to guarantee Range queries in worst-case $\Theta\big(H + \gamma + \frac{K}{B}\big)$ I/Os when $M \geq 2B$.

The worst-case I/O cost of a Range query can be improved by merging all the buffered updates to the open rectangles intersecting the query in a top-down, level-by-level fashion. That is, by essentially actualizing all the open rectangles intersected by the Range query simultaneously. We denote the updates already present in the rectangles the *partial output list*. Rather than applying the buffered updates to the rectangles, we merge them with the partial output list to obtain the final output. A given query Range(v, x, y) reports $\Omega(R)$ keys from each intermediate rectangles, i.e., all rectangles intersecting the query except for the two that contain the endpoints x and y. Thus, in $\mathcal{O}((1 + K/R)H + (R + K)/B) = \mathcal{O}(H + K/B)$ I/Os we can find all the relevant rectangles and compute the partial output list. To collect the buffered updates in T for the intermediate open rectangles in sorted order, we merge the updates down level-by-level. We only move down the updates to versions earlier or equal to v since only these can affect the query result. Once obtained, these updates are merged with the partial output list using linear I/Os to report the output of the Range query.

The buffered updates are stored in T, which contains the open rectangles at version v_c, however, the Range query is on the rectangles present at version v. Let T_v denote the B^ε-tree on open rectangles at version v, i.e., the state of T when version v was created. From T_v to T the tree may have changed, but no later updates are relevant for the query. Thus, the total number of relevant updates does not increase from T_v to T, and each update remains on the root-to-leaf path towards the open rectangle to which the update is relevant. The relevant updates in T can be collected in sorted lists ordered by level by traversing each root-to-leaf path in T towards open rectangles intersecting the query using $\mathcal{O}((1 + K/R)H)$

I/Os. To bound the I/Os to move the updates down level-by-level, we show that the total number of updates is $\mathcal{O}\big(B^{1-\varepsilon}\log_2 \overline{N} + K/H\big)$. To this end, we need the additional invariant that the buffer of a degree one node is empty, which we show how to obtain below. Let T'_v be the subtree of T_v consisting of all nodes on root-to-leaf paths to rectangles intersecting the query. Then split T'_v into two root-to-leaf paths p_x and p_y to x and y, respectively, along with all the subtrees hanging off p_x or p_y. For a node on p_x (symmetrically p_y) of degree at least two there may be one or more subtrees T_{sub} hanging off the node. Since only the nodes of T_{sub} with degree at least two have non-empty buffers, if T_{sub} has ℓ leaves, the number of buffered updates in T_{sub} is at most $\mathcal{O}((\ell-1)B)$. Thus, the number of buffered updates in T'_v excluding degree one nodes on p_x and p_y is $\mathcal{O}(K/H)$. A degree one node on p_x and p_y may have a large degree in T_v, but since it only has one child in the direction of the query, due to the subtracting game, it stores at most $\mathcal{O}(F\log_2\Delta)$ relevant updates. The number of degree one nodes on p_x and p_y is at most $2H$, and they together contribute $\mathcal{O}\big(B^{1-\varepsilon}\log_2\overline{N}\big)$ buffered updates, which we locate and sort separately using $\mathcal{O}(H+\gamma)$ I/Os. For the remaining $\mathcal{O}(K/H)$ buffered updates, we merge them level-by-level using $\mathcal{O}(H+K/B)$ I/Os. In total, the worst-case number of I/Os to perform a RANGE query is $\mathcal{O}(H+\gamma+K/B)$ I/Os.

Empty Buffers for Degree one Nodes. To ensure that each node of degree one has an empty buffer, we alter the buffer capacity of nodes to scale with the degree. Let the capacity of the buffer of a node with degree $d \geq 2$ be at most $F\cdot\min\{2d, 2\Delta\}$, with nodes of degree one having a buffer capacity of 0. When flushing a node, as the maximum number of updates in the buffer scales with the degree, then at least F updates going to the same child can be found when overflowing. When splitting a node, it must have degree 2Δ, resulting in the two new nodes having degree Δ, which therefore does not decrease the buffer capacity, and flushing is not needed. When a child of a node is removed due to merging rectangles, the degree of the node is decreased by one. If the degree remains at least two, at most two flushes are required to get the buffer capacity within bounds. Otherwise, if the degree drops from two to one, at most four flushes are needed. To avoid cascading merges of rectangles, we do not finalize a rectangle once it receives a certain number of updates. Instead, we finalize the rectangle that has received the most updates, provided it has received at least R updates. This last condition ensures a bound on the space usage. Including the initial flush from the root buffer, then when a rectangle is finalized, at most $5F$ updates have been flushed into open rectangles.

Let U_i denote the number of updates to the ith rectangle, excluding the initial insertions. We extend the data structure to include an array over all open rectangles, where index j stores a blocked-linked-list of all rectangles where the number of updates is $U_i = j$. Each rectangle has a double linked pointer between its location in the array of lists and the rectangle. This allows for moving a rectangle to a new entry in the array, when it receives updates, as well as finding a rectangle which has received the most updates, by scanning the list.

To show that U_i is bounded by $\mathcal{O}(R)$, we apply the *zeroing game* of Dietz and Sleator [19], using the variables $x_i = \max\left\{0, \frac{U_i - R}{5F}\right\}$ if rectangle i is open and $x_i = 0$ if it is closed. For open rectangles, x_i counts the number of units of $5F$ updates received beyond the first R updates. This ensures that the variables are incremented by at most 1 in total for each round, when at most 5 flushes of size at most F are flushed into the open rectangles. When finalizing a rectangle, it becomes closed, which ensures that $x_i = 0$, matching the zeroing step. We bound the total number of rectangles by $\overline{N}$ and therefore also the number of variables. Following [19, Theorem 5] and the refined analysis in [13, Theorem 4], we have that for

all i, $x_i \leq \log_2 \overline{N} + 1$ at all times. Consequently, it follows that $U_i \leq 5F(\log_2 \overline{N} + 1) + R$. It can be shown that $5F(\log_2 \overline{N} + 1) \leq 2R$ for $\overline{N} \geq 4$, by simplifying the inequality using $R = 2\Delta F\left(1 + \left\lceil \log_\Delta \overline{N} \right\rceil\right)$ and showing $\frac{5}{4}\left(1 + \frac{1}{\log_2 \overline{N}}\right) \leq \frac{\Delta}{\log_2 \Delta}$, for all $\Delta \geq 2$ and $\overline{N} \geq 4$. It therefore holds that each rectangle receives at most $U_i \leq 3R$ updates, due to the zeroing game. Thus, when finalizing a rectangle, at least R updates have been performed. Including the updates from the buffers on the path towards the rectangle, the total number of updates applied is between R and $4R$. By ensuring that each rectangle contains $[8R, 16R[$ initial keys, the rebalancing operations are possible, and the spanning requirement remains satisfied.

An update therefore performs at most 5 flushes using $\mathcal{O}(H)$ I/Os, along with locating and finalizing a single rectangle in respectively $\mathcal{O}(R/B) = \mathcal{O}(H)$ and $\mathcal{O}(H + \gamma)$ I/Os. Performing this operation incrementally allows for updates to spend worst-case $\mathcal{O}\left(\frac{1}{F}(H + \gamma)\right)$ I/Os. If a query happens while an incremental update is being performed, the incremental update is completed, using at most $\mathcal{O}(H + \gamma)$ I/Os, which does not increase the total cost of the query. When a SEARCH query happens, similar to the new RANGE query, we do not apply the relevant updates on the path to the open rectangle to avoid queries interfering with the zeroing game. This concludes the proof of Theorem 3.

4 Discussion and Open Problems

Global rebuilding, as described in Section 2.6, allows for constructing a partially-persistent set of any sorted set in a linear number of I/Os, without recreating the set by a sequence of insertions. Symmetrically, it is possible to *purge* all versions of the set older than some threshold without performing all updates again. This problem was first motivated by Becker, Gschwind, Ohler, Seeger, and Widmayer [6]. As our data structure consists of multiple independent data structures covering disjoint version intervals, then all data structures which only cover versions to be purged can be removed efficiently by a linear number of I/Os. For both use cases, space usage is asymptotically linear in the size of the oldest stored set and the number of updates performed.

Further, global rebuilding allows for a crude fully persistent data structure, which supports efficient buffered updates and queries, but where cloning past versions requires a linear number of I/Os. The fully persistent data structure by Brodal, Rysgaard, and Svenning [12] allows cloning past versions in worst case $\mathcal{O}(1)$ I/Os. They do, however, not buffer updates, which therefore are amortized and a factor $\mathcal{O}\left(\varepsilon B^{1-\varepsilon}\right)$ slower than our data structure. Our data structure is therefore better when there are many updates, but few clones of past versions happening. Further, our data structure is simpler. It remains an open problem to design buffered fully-persistent search-trees that remain efficient for clone operations.

In Section 3 we showed how to achieve worst-case bounds matching those of ephemeral B^ε-trees, when $M = \Omega\left(B^{1-\varepsilon} \log_2 N\right)$. This is an improvement by a factor $\Theta(B^\varepsilon / \log_2 B)$ on the required lower bound on M over the worst-case results of B^ε-trees by Das, Iacono, and Nekrich [16]. It remains an open problem to show a worst-case I/O lower bound dependency on M, or alternatively, to find a structure with worst-case I/O guarantees matching the amortized I/O bounds for $M = 2B$.

References

1 Alok Aggarwal and Jeffrey S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, September 1988. doi:10.1145/48529.48535.

2 Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, September 2003. doi:10.1007/s00453-003-1021-x.

3 Lars Arge, Andrew Danner, and Sha-Mayn Teh. I/O-efficient point location using persistent B-trees. *ACM Journal of Experimental Algorithmics*, 8:1.2–es, December 2004. doi:10.1145/996546.996549.

4 Lars Arge, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory algorithms for processing line segments in geographic information systems. *Algorithmica*, 47:1–25, 2007. doi:10.1007/s00453-006-1208-z.

5 Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972. doi:10.1007/BF00288683.

6 Bruno Becker, Stephan Gschwind, Thomas Ohler, Bernhard Seeger, and Peter Widmayer. An asymptotically optimal multiversion B-tree. *The VLDB Journal*, 5(4):264–275, 1996. doi:10.1007/s007780050028.

7 Michael A. Bender, Rathish Das, Martín Farach-Colton, Rob Johnson, and William Kuszmaul. Flushing without cascades. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 650–669. Society for Industrial and Applied Mathematics, 2020. doi:10.1137/1.9781611975994.40.

8 Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Yonatan R. Fogel, Bradley C. Kuszmaul, and Jelani Nelson. Cache-oblivious streaming B-trees. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 81–92. Association for Computing Machinery, 2007. doi:10.1145/1248377.1248393.

9 Michael A. Bender, Martin Farach-Colton, William Jannen, Rob Johnson, Bradley C. Kuszmaul, Donald E. Porter, Jun Yuan, and Yang Zhan. An introduction to B-trees and write-optimization. *;login:*, 40(5), 2015. URL: https://www.usenix.org/publications/login/oct15/bender.

10 Gerth Stølting Brodal and Rolf Fagerberg. Lower bounds for external memory dictionaries. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 546–554. Society for Industrial and Applied Mathematics, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644201.

11 Gerth Stølting Brodal and Rolf Fagerberg. On the limits of cache-obliviousness. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 307–315. Association for Computing Machinery, 2003. doi:10.1145/780542.780589.

12 Gerth Stølting Brodal, Casper Moldrup Rysgaard, and Rolf Svenning. External memory fully persistent search trees. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1410–1423. Association for Computing Machinery, 2023. doi:10.1145/3564246.3585140.

13 Gerth Stølting Brodal, Casper Moldrup Rysgaard, and Rolf Svenning. Buffered partially-persistent external-memory search trees. *CoRR*, abs/2503.08211, 2025. doi:10.48550/arXiv.2503.08211.

14 Adam L. Buchsbaum, Michael Goldwasser, Suresh Venkatasubramanian, and Jeffery R. Westbrook. On external memory graph traversal. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 859–860. Society for Industrial and Applied Mathematics, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338650.

15 Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359. Curran Associates, Inc., 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf.

16 Rathish Das, John Iacono, and Yakov Nekrich. External-memory dictionaries with worst-case update cost. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 21:1–21:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.21.

17 Erik D. Demaine, John Iacono, and Stefan Langerman. Retroactive data structures. *ACM Trans. Algorithms*, 3(2):13–es, May 2007. doi:10.1145/1240233.1240236.

18 Paul F. Dietz and Rajeev Raman. Persistence, randomization and parallelization: On some combinatorial games and their applications (abstract). In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, Nicola Santoro, and Sue Whitesides, editors, *Algorithms and Data Structures, Third Workshop, WADS '93, Montréal, Canada, August 11-13, 1993, Proceedings*, volume 709 of *Lecture Notes in Computer Science*, pages 289–301. Springer, 1993. doi:10.1007/3-540-57155-8_256.

19 Paul F. Dietz and Daniel Dominic Sleator. Two algorithms for maintaining order in a list. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 365–372. Association for Computing Machinery, 1987. doi:10.1145/28395.28434.

20 James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989. doi:10.1016/0022-0000(89)90034-2.

21 Herbert Edelsbrunner and Mark H. Overmars. Batched dynamic solutions to decomposable searching problems. *Journal of Algorithms*, 6(4):515–542, 1985. doi:10.1016/0196-6774(85)90030-6.

22 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. *ACM Transactions on Algorithms*, 8(1), January 2012. doi:10.1145/2071379.2071383.

23 Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57. Association for Computing Machinery, 1984. doi:10.1145/602259.602266.

24 Michael Hamann, Ulrich Meyer, Manuel Penschuck, Hung Tran, and Dorothea Wagner. I/O-efficient generation of massive graphs following the LFR benchmark. *ACM Journal of Experimental Algorithmics*, 23, August 2018. doi:10.1145/3230743.

25 William Jannen, Jun Yuan, Yang Zhan, Amogh Akshintala, John Esmet, Yizheng Jiao, Ankur Mittal, Prashant Pandey, Phaneendra Reddy, Leif Walsh, Michael A. Bender, Martin Farach-Colton, Rob Johnson, Bradley C. Kuszmaul, and Donald E. Porter. BetrFS: Write-optimization in a kernel file system. *ACM Transactions on Storage*, 11(4), November 2015. doi:10.1145/2798729.

26 Curtis P. Kolovson and Michael Stonebraker. Indexing techniques for historical databases. In *Proceedings of the Fifth International Conference on Data Engineering, February 6-10, 1989, Los Angeles, California, USA*, pages 127–137. IEEE Computer Society, 1989. doi:10.1109/ICDE.1989.47208.

27 Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, 80:016118, July 2009. doi:10.1103/PhysRevE.80.016118.

28 Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78:046110, October 2008. doi:10.1103/PhysRevE.78.046110.

29 Mark H. Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1983. doi:10.1007/BFb0014927.

30 Mark H. Overmars and Jan van Leeuwen. Dynamization of decomposable searching problems yielding good worsts-case bounds. In Peter Deussen, editor, *Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings*, volume 104 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 1981. doi:10.1007/BFB0017314.

31 Neil Sarnak and Robert E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, July 1986. doi:10.1145/6138.6151.

32 Peter J. Varman and Rakesh M. Verma. An efficient multiversion access structure. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):391–409, 1997. doi:10.1109/69.599929.

Fault-Tolerant Matroid Bases

Matthias Bentert
University of Bergen, Norway
Technische Universität Berlin, Germany

Fedor V. Fomin
University of Bergen, Norway

Petr A. Golovach
University of Bergen, Norway

Laure Morelle
LIRMM, Univ Montpellier, CNRS, Montpellier, France

Abstract

We investigate the problem of constructing *fault-tolerant bases* in matroids. Given a matroid $\mathcal{M}$ and a redundancy parameter k, a k-fault-tolerant basis is a minimum-size set of elements such that, even after the removal of any k elements, the remaining subset still spans the entire ground set. Since matroids generalize linear independence across structures such as vector spaces, graphs, and set systems, this problem unifies and extends several fault-tolerant concepts appearing in prior research.

Our main contribution is a fixed-parameter tractable (FPT) algorithm for the k-fault-tolerant basis problem, parameterized by both k and the rank r of the matroid. This two-variable parameterization by $k + r$ is shown to be tight in the following sense. On the one hand, the problem is already NP-hard for $k = 1$. On the other hand, it is Para-NP-hard for $r \geq 3$ and polynomial-time solvable for $r \leq 2$.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Theory of computation → Fixed parameter tractability

Keywords and phrases Parameterized Complexity, matroids, robust bases

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.83

Related Version *Full Version*: `https://arxiv.org/abs/2506.22010` [4]

Funding The research leading to these results have been supported by the Research Council of Norway via the Franco-Norwegian AURORA project (grant no. 349476).
Matthias Bentert: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416) and ERC Consolidator grant AdjustNet (agreement No. 864228).
Fedor V. Fomin: Supported by the Research Council of Norway under BWCA project (grant no. 314528).
Petr A. Golovach: Supported by the Research Council of Norway under BWCA project (grant no. 314528).

1 Introduction

A basis in a d-dimensional vector space $\mathcal{V}$ is a set of exactly d linearly independent vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$. Any vector $\mathbf{x} \in \mathcal{V}$ then has a *unique* representation $\mathbf{x} = \alpha_1\mathbf{v}_1 + \cdots + \alpha_d\mathbf{v}_d$. However, if certain basis vectors are lost or corrupted – for instance, in a distributed storage system where vectors are stored across multiple servers that can malfunction – it may be challenging or impossible to accurately recover $\mathbf{x}$. The algorithmic question we address in this paper is: *For an expected number k of failures, what is the minimum-size k-fault-tolerant set of vectors that still guarantees data reconstructability?*


© Matthias Bentert, Fedor V. Fomin, Petr A. Golovach, and Laure Morelle;
licensed under Creative Commons License CC-BY 4.0



A natural way to capture the notions of basis and independence is through *matroids*, which generalize linear independence to a variety of combinatorial settings (like graphs, set systems, vector spaces). Let $\mathcal{M} = (E, \mathcal{I})$ be a matroid, and let k be a nonnegative integer. We say that $B \subseteq E$ is a *k-fault-tolerant basis* of $\mathcal{M}$ if B is a minimum-size set such that, for every subset $F \subseteq B$ of size at most k, we have $\mathsf{rank}(B \setminus F) = \mathsf{rank}(\mathcal{M})$. (All necessary matroid definitions appear in the Preliminaries section.) Equivalently, this condition means that $\mathsf{cl}(B \setminus F) = E$, where $\mathsf{cl}(\cdot)$ denotes matroid closure. Note that this concept generalizes the standard basis definition: a 0-fault-tolerant basis is simply a basis of $\mathcal{M}$. However, while every matroid has a basis, it may not admit a k-fault-tolerant basis for $k \geq 1$. We investigate the following problem.

Fault-Tolerant Basis

Input:	A matroid $\mathcal{M} = (E, \mathcal{I})$ and nonnegative integer k.
Task:	Find a k-fault-tolerant basis $B \subseteq E$ or correctly decide that none exists.

Let us give some examples of Fault-Tolerant Basis.

Linear Matroids. A *linear matroid* over a field $\mathbb{F}$ can be represented by a set of vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ in a vector space $\mathcal{V} \subseteq \mathbb{F}^d$. The rank of the matroid corresponds to $\dim(\mathcal{V})$. A basis here is simply a set of d linearly independent vectors. A set of vectors $\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$ is k-fault-tolerant if, after removing up to k vectors, the remaining set still spans the entire subspace $\mathcal{V}$. Such concepts naturally arise in distributed storage – where vectors might be stored on different servers [11] – and in coding theory, where losing some symbols or vectors should not destroy the ability to recover the entire space [13].

Graphic Matroids. For a connected graph G, its *graphic matroid* $\mathcal{M}(G)$ has ground set $E(G)$, and a set $I \subseteq E(G)$ is independent if it forms a forest. A basis is thus a spanning tree of G. In this matroid, a set of edges $B \subseteq E(G)$ is k-fault-tolerant if, upon removing any k edges, the subgraph remains connected. Thus, a k-fault-tolerant basis is a $(k+1)$-edge connected spanning subgraph with minimum number of edges. This problem is known in the literature as k-Edge-Connected Spanning Subgraph [8, 14, 17]. It is motivated by network applications as its task is to compute a minimum-cost sub-network that is resilient against up to k link failures (see also [5]).

Gammoids. A *gammoid* is derived from a directed (or undirected) graph D. Let X and Y be two distinguished sets of vertices of $V(D)$. Within the set X, define a subset $U \subseteq X$ to be *independent* if there exist $|U|$ vertex-disjoint paths in G originating from vertices in Y and ending in the vertices of U. The basis of the gammoid is the maximum set of vertices from X that can be reached by vertex-disjoint paths from Y. In this case, a vertex set $Z \subseteq X$ is a k-fault-tolerant basis if upon removing any k vertices from Z, the remaining vertices are still linkable from sources in Y without losing rank. This problem is related to the problem of finding fault-tolerant disjoint paths studied by Adjiashvili et al. [1].

Transversal Matroids. A *transversal matroid* $\mathcal{M}(U, \mathcal{S})$ is described via a bipartite graph between a ground set U and a set of "target positions" $\mathcal{S}$. A set $I \subseteq U$ is independent if it can be matched injectively to distinct elements in $\mathcal{S}$. The basis of the transversal matroid is a maximum-size subset $I \subseteq U$ that can be perfectly matched into $\mathcal{S}$. Finding a k-fault-tolerant basis for a transversal matroids is naturally related to robust matching problems, where one wants to preserve a perfect (or maximum) matching under the loss of a few vertices or edges [12, 18].

1.1 Our Contribution

We investigate the parameterized complexity of Fault-Tolerant Basis under natural parameterizations by k and the rank r of the input matroid. (We refer to the textbook by Cygan et al. [10] for an introduction to parameterized complexity.) Our main result is that the problem is FPT when parameterized by both k and r.

▶ **Theorem 1.** *Fault-Tolerant Basis can be solved in* $(kr)^{\mathcal{O}(kr^4)} \cdot n^{\mathcal{O}(1)}$ *time on* n*-element matroids of rank at most* r *given by independence oracles.*

To prove Theorem 1, we design an algorithm that recursively decomposes the input matroid and identifies a small *core set* containing any possible k-fault-tolerant basis, thereby restricting the problem to a bounded search space. More precisely, the algorithm locates a set W of *important elements* with bounded size such that, if the input matroid $\mathcal{M}$ admits a k-fault-tolerant basis, then there is one contained entirely in W. The construction of W is guided by the observation (see Observation 7) that if X is a rank-r set with $r + k$ elements such that *every* r-element subset of X is independent, then X is already a k-fault-tolerant basis. Hence, finding such an X would immediately solve the problem. Otherwise, we identify an inclusion-maximal rank-r set X (of bounded size) such that for every r-element subset $Y \subseteq X$, the rank of Y is r. The crucial insight here is that the closure $\mathsf{cl}(X)$ of X can be expressed as the union of $\mathsf{cl}(S)$ over all $(r-1)$-element subsets S of X. We then proceed recursively, selecting elements of W by searching for analogous sets in the closures of each S, applying the same approach used to choose X. Once we have computed W, we determine whether a k-fault-tolerant basis exists within W by enumerating all candidate subsets via a brute-force algorithm. If such a basis is found, it is also a valid solution for the original matroid.

We then show that the result of Theorem 1 is tight. First, by adapting the results of Fomin et al. [15] for our purposes, we observe the following lower bound.

▶ **Proposition 2.** *It is* W[1]*-hard for the parameterization by* k *to decide whether a given linear matroid has a* k*-fault-tolerant basis.*

For the parameterization of Fault-Tolerant Basis by k, we remind that Fault-Tolerant Basis for graphic matroids is equivalent to finding a $(k+1)$-edge connected spanning subgraph with the minimum number of edges. As was shown by Fernandes [14], an n-vertex graph G has a 2-edge connected spanning subgraph with at most n edges if and only if G has a Hamiltonian cycle. Thus, Fault-Tolerant Basis is intractable already for $k = 1$. Taking into account the inapproximability lower bounds for higher connectivities established by Gabow et al. [17], we obtain the following observation.

▶ **Observation 3.** *For every integer* $k \geq 1$*, it is* NP*-hard to decide whether a graphic matroid* $\mathcal{M}$ *given with an integer* $b \geq 1$ *has a* k*-fault-tolerant basis of size at most* b*.*

For the parameterization by the rank, we establish a dichotomy – for any fixed $r \geq 3$, Fault-Tolerant Basis is intractable, but for $r \leq 2$, the problem can be solved in polynomial time. In fact, for $r \leq 2$, we solve the more general weighted variant of Fault-Tolerant Basis. In Weighted Fault-Tolerant Basis, we are given a matroid $\mathcal{M}$ together with a weight function $w \colon E(\mathcal{M}) \to \mathbb{Z}_{\geq 0}$, and the task is to find a set $B \subseteq E(\mathcal{M})$ of minimum total weight such that for any set $F \subseteq B$ of size at most k, $\mathsf{rank}(B \setminus F) = \mathsf{rank}(\mathcal{M})$. Our result is summarized in the following theorem.

▶ **Theorem 4.** *For any integer $r \geq 3$, it is* NP*-hard to decide whether a linear matroid $\mathcal{M}$ of rank r over rationals given together with integers k and b has a k-fault-tolerant basis of size at most b. For $r \leq 2$,* Weighted Fault-Tolerant Basis *can be solved in $\mathcal{O}(n^2)$ on matroids given by an independence oracle.*

In particular, one can find a k-fault-tolerant basis of vectors in $\mathbb{R}^2$ in polynomial time but the problem becomes NP-hard in $\mathbb{R}^3$. To obtain the hardness for $r \geq 3$, we use the result of Froese et al. [16] stating that, given a set P of points on the plane and a positive integer k, it is NP-hard to decide whether P contains at least k points in general position. To establish the claim for $r \leq 2$, it is convenient to show a more general result for partition matroids which may be of independent interest.

▶ **Proposition 5.** *Given an n-element partition matroid $\mathcal{M}$ with unit capacities together with a weight function $w\colon E(\mathcal{M}) \to \mathbb{Z}_{\geq 0}$ and integers $r, k \geq 0$, one can find in $\mathcal{O}(n^2)$ time either a set $X \subseteq E(\mathcal{M})$ of minimum weight such that, for any set $F \subseteq X$ of size at most k, it holds that $\mathsf{rank}(X \setminus F) \geq r$ or correctly decide that such a set does not exist.*

1.2 Related Work

Adjiashvili, Stiller, and Zenklusen [2] introduced a *Bulk-Robustness* model of combinatorial optimization. They studied several instances of this framework, including the Bulk-Robust Minimum Matroid Basis problem, which is most relevant to our work. In this setting, for a matroid $\mathcal{M} = (E, \mathcal{I})$, we are given a collection of *interdiction sets* $\Omega = \{F_1, F_2, \ldots, F_m\}$, where $F_i \subseteq E(\mathcal{M})$ for each $i \in \{1, \ldots, m\}$. The goal is to find a cheapest subset $X \subseteq E(\mathcal{M})$ such that $X \setminus F_i$ contains a basis of $\mathcal{M}$ for every $i \in \{1, \ldots, m\}$. Adjiashvili, Stiller, and Zenklusen provided an $\mathcal{O}(\log m + \log r)$-approximation algorithm for Bulk-Robust Minimum Matroid Basis, where r is the rank of the matroid. The problem Fault-Tolerant Basis can be viewed as a special case of Bulk-Robust Minimum Matroid Basis when Ω consists of all subsets of E of size k.

There are several prior works investigating fault-tolerance for classic optimization problems. The model of s-t-Path and s-t-Flow problems with safe and vulnerable edges was introduced by Adjiashvili et al. [1], who studied the approximability of these problems. Subsequently, generalizations were also studied [3, 7, 9]. Bentert et al. [5] studied the parameterized complexity of computing a fault-tolerant spanning tree. Approximation algorithms and inapproximability lower bounds for k-Edge-Connected Spanning Subgraph have been considered in [8, 14, 17, 19].

More generally, Fault-Tolerant Basis belongs to *robust optimization*, a branch of optimization that adapts classic optimization-theoretic tools to settings with uncertainty. For a comprehensive introduction, see the survey by Bertsimas et al. [6].

2 Preliminaries

We refer to the book of Oxley [21] for a detailed introduction to matroids. A pair $\mathcal{M} = (E, \mathcal{I})$, where E is a finite *ground* set and $\mathcal{I}$ is a family of subsets of the ground set, called *independent sets* of $\mathcal{M}$, is a *matroid* if it satisfies the following conditions, called *independence axioms*:

(I1) $\emptyset \in \mathcal{I}$.

(I2) If $A \subseteq B \subseteq E$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$.

(I3) If $A, B \in \mathcal{I}$ and $|A| < |B|$, then there is $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$.

We use $E(\mathcal{M})$ and $\mathcal{I}(\mathcal{M})$ to denote the ground set and the set of independent sets, respectively. An inclusion-maximal independent set B is called a *basis* of $\mathcal{M}$. We use $\mathcal{B}(\mathcal{M})$ to denote the set of bases of $\mathcal{M}$. All the bases of $\mathcal{M}$ have the same size called the *rank* of $\mathcal{M}$ and

denoted by $\mathsf{rank}(\mathcal{M})$. The *rank* of a subset $A \subseteq E(\mathcal{M})$, denoted by $\mathsf{rank}(A)$, is the maximum size of an independent set $X \subseteq A$; the function $\mathsf{rank}\colon 2^{E(\mathcal{M})} \to \mathbb{Z}$ is the *rank* function of M. A set $A \subseteq E(\mathcal{M})$ *spans* an element $x \in E(\mathcal{M})$ if $\mathsf{rank}(A \cup \{x\}) = \mathsf{rank}(A)$. The *closure* (or *span*) of A is the set $\mathsf{cl}(A) = \{x \in E(M) \mid A \text{ spans } x\}$. Closures satisfy the following properties, called *closure axioms*:

(CL1) For every $A \subseteq E(\mathcal{M})$, $A \subseteq \mathsf{cl}(A)$.
(CL2) If $A \subseteq B \subseteq E(\mathcal{M})$, then $\mathsf{cl}(A) \subseteq \mathsf{cl}(B)$.
(CL3) For every $A \subseteq E(\mathcal{M})$, $\mathsf{cl}(A) = \mathsf{cl}(\mathsf{cl}(A))$.
(CL4) For every $A \subseteq E(\mathcal{M})$, every $x \in E(\mathcal{M}) \setminus A$, and every $y \in \mathsf{cl}(A \cup \{x\}) \setminus \mathsf{cl}(A)$, $x \in \mathsf{cl}(A \cup \{y\})$.

For $S \subseteq E(M)$, the matroid obtained by *deleting* S, denoted by $\mathcal{M}' = \mathcal{M} - S$, is the matroid such that $E(\mathcal{M}') = E(\mathcal{M}) \setminus S$ and $\mathcal{I}(\mathcal{M}') = \{X \in \mathcal{I}(\mathcal{M}) \mid S \cap X = \emptyset\}$. Let $r \geq 0$ be an integer. The *r-truncation* of a matroid $\mathcal{M}$ is the matroid $\mathcal{M}'$ with $E(\mathcal{M}') = E(\mathcal{M})$ such that $X \in \mathcal{I}(\mathcal{M}')$ if and only if $X \in \mathcal{I}(\mathcal{M})$ and $|X| \leq r$. A matroid $\mathcal{M}$ is a *partition* matroid if there is a partition $(P_1, \ldots, P_d)$ of the ground set and a d-tuple $(c_1, \ldots, c_d)$ of positive integers, called capacities, such that a set $X \subseteq E(G)$ is independent if and only if $|X \cap P_i| \leq c_i$ for every $i \in \{1, \ldots, d\}$. A $d \times n$-matrix $\mathbf{A}$ over a field $\mathbb{F}$ is a *representation of a matroid $\mathcal{M}$ over $\mathbb{F}$* if there is a one-to-one correspondence f between $E(\mathcal{M})$ and the columns of $\mathbf{A}$ such that for any $X \subseteq E(\mathcal{M})$, it holds that $X \in \mathcal{I}(\mathcal{M})$ if and only if the columns in $\{f(x) \mid x \in X\}$ are linearly independent (as vectors of $\mathbb{F}^d$). If $\mathcal{M}$ has such a representation, then $\mathcal{M}$ has a *representation over* $\mathbb{F}$. A matroid $\mathcal{M}$ admitting a representation over $\mathbb{F}$ is said to be *linear*; a matroid is *binary* if it has a representation over $\mathsf{GF}[2]$.

In our algorithms for general matroids, we assume that input matroids are given by *independence oracles*. Such an oracle for a matroid $\mathcal{M}$, takes as input a set $X \subseteq E(\mathcal{M})$ and correctly answers in a constant time whether $X \in \mathcal{I}(\mathcal{M})$. We note that some matroids could be given explicitly, for example, by their representations.

3 Basic Observations

In this section, we prove a couple of simple results that will be helpful in the later discussion. Recall that for an integer $k \geq 0$, $B \subseteq E(M)$ is a k-fault-tolerant basis of a matroid $\mathcal{M}$ if B is a subset of minimum cardinality such that $\mathsf{rank}(B \setminus F) = \mathsf{rank}(\mathcal{M})$ for every subset $F \subseteq B$ of size at most k. We use the following bounds on the size of a k-fault-tolerant basis.

▶ **Proposition 6.** *Let $\mathcal{M}$ be a matroid with $\mathsf{rank}(\mathcal{M}) \geq 1$ and let $k \geq 0$ be an integer. Then for a k-fault-tolerant basis B of $\mathcal{M}$, it holds that*

$$\mathsf{rank}(\mathcal{M}) + k \leq |B| \leq (k+1)\mathsf{rank}(\mathcal{M}).$$

Proof. The lower bound immediately follows from the definition of a k-fault-tolerant basis. To show the upper bound, assume for the sake of contradiction that $|B| > (k+1)\mathsf{rank}(\mathcal{M})$. We iteratively construct sets $X_0, \ldots, X_k$ where $X_0 \subseteq B$ is an inclusion-maximal independent set in B, and for each $i \in \{1, \ldots, k\}$, X_i is an inclusion-maximal independent set in $B \setminus \left(\bigcup_{j=0}^{i-1} X_j\right)$. Since $\mathsf{rank}(B) = \mathsf{rank}(\mathcal{M})$ and $|B| > (k+1)\mathsf{rank}(\mathcal{M})$, such sets $X_0, \ldots, X_k$ exist. Set $B' = \bigcup_{i=0}^{k} X_i$. Consider an arbitrary $F \subseteq B$ of size at most k. By the pigeonhole principle, there is an index $i \in \{0, \ldots, k\}$ such that $X_i \cap F = \emptyset$. By construction, it holds that $B \setminus B' \subseteq B \setminus \left(\bigcup_{j=0}^{i-1} X_j\right) \subseteq \mathsf{cl}(X_i)$. Hence, $B \setminus F \subseteq (B' \setminus F) \cup (B \setminus B') \subseteq \mathsf{cl}(B' \setminus F)$. Because B is a k-fault-tolerant basis, $\mathsf{cl}(B \setminus F) = E(\mathcal{M})$. Thus,

$$E(\mathcal{M}) = \mathsf{cl}(B \setminus F) \subseteq \mathsf{cl}(\mathsf{cl}(B' \setminus F)) = \mathsf{cl}(B' \setminus F).$$

Since F was chosen arbitrarily, we obtain that B' is a k-fault-tolerant basis of $\mathcal{M}$. However, since $|B'| < |B|$, this contradicts the choice of B. ◀

We next argue that the bounds in Proposition 6 are tight. First, note that the lower bound is tight as the matroid $\mathcal{I}(\mathcal{M}) = \{X \subseteq E(\mathcal{M})\colon |X| \leq k\}$ proves. We next argue that the upper bound is also tight. Let $e_1, \ldots, e_r$ be a basis of $\mathbb{R}^r$. Let k and n be integers such that $0 \leq k < n$. Consider the linear matroid $\mathcal{M}$ with $E(\mathcal{M}) = \{je_i \mid 1 \leq i \leq r \text{ and } 1 \leq j \leq n\}$. It is easy to see that any k-fault-tolerant basis of $\mathcal{M}$ has to contain at least $k+1$ vectors of $\{je_i \mid 1 \leq j \leq n\}$ for each $i \in \{1, \ldots, r\}$. Thus, the size of a k-fault-tolerant basis is at least $(k+1)r$. From the other side, we have that, for $B = \{je_i \mid 1 \leq i \leq r \text{ and } 1 \leq j \leq k+1\}$, $\mathsf{rank}(B \setminus F) = r$ for any $F \subseteq E(\mathcal{M})$ of size at most k. Thus, B is a k-fault-tolerant basis of size $(k+1)r$ of $\mathcal{M}$.

Let $\mathcal{M}$ be a matroid. For a positive integer h, we say that a set $X \subseteq E(\mathcal{M})$ is *h-uniform* if $\mathsf{rank}(X) = h$ and $\mathsf{rank}(Y) = h$ for every subset $Y \subseteq X$ of size h. The definition and Proposition 6 yield the following.

▶ **Observation 7.** *Let $\mathcal{M}$ be a matroid of rank $r \geq 1$, $k \geq 0$ be an integer, and $B \subseteq E(\mathcal{M})$ be a set of size $k + r$. Then, B is a k-fault-tolerant basis of $\mathcal{M}$ if and only if B is r-uniform.*

Proof. If B is a k-fault-tolerant basis, then, for every set $F \subseteq B$ of size at most k, $\mathsf{rank}(B \setminus F) = r$. Since $|B| = k + r$, this implies that, for every set $X \subseteq B$ of size r, $\mathsf{rank}(X) = \mathsf{rank}(B) = r$, that is, B is r-uniform. For the opposite direction, if B is r-uniform, then, for any $X \subseteq B$ of size r, $\mathsf{rank}(X) = \mathsf{rank}(B) = \mathsf{rank}(\mathcal{M})$. Since $|B| = k + r$, it holds for any $F \subseteq B$ of size at most k that $\mathsf{rank}(B \setminus F) = \mathsf{rank}(\mathcal{M})$. This completes the proof. ◀

We conclude this section by proving Proposition 2 using the results of Fomin et al. [15]. In particular, they studied Rank h-Reduction. Here, the input is a binary matroid $\mathcal{M}$ given by its representation over $\mathsf{GF}[2]$ and two positive integers h and k. The task is to decide whether there is a set $X \subseteq E(\mathcal{M})$ of size at most k such that $\mathsf{rank}(\mathcal{M}) - \mathsf{rank}(\mathcal{M} - X) \geq h$.

▶ **Proposition 2.** *It is* W[1]*-hard for the parameterization by k to decide whether a given linear matroid has a k-fault-tolerant basis.*

Proof. We reduce from Rank h-Reduction parameterized by k, which is known to be W[1]-hard [15]. Let $(\mathcal{M}, h, k)$ be an instance of Rank h-Reduction where $\mathcal{M}$ is a binary matroid of rank r. We define $\mathcal{M}'$ to be the $(r - h + 1)$-truncation of $\mathcal{M}$. Notice that $\mathcal{M}'$ is a linear matroid and its representation (over a different field) can be constructed in polynomial time by the result of Lokshtanov et al. [20]. We claim that $\mathcal{M}'$ has no k-fault-tolerant basis if and only if $(\mathcal{M}, h, k)$ is a yes-instance of Rank h-Reduction.

To this end, note that $\mathcal{M}'$ has no k-fault-tolerant basis if and only if there is a set $X \subseteq E(\mathcal{M}')$ of size at most k such that $\mathsf{rank}(\mathcal{M}' - X) < \mathsf{rank}(\mathcal{M}') = \mathsf{rank}(\mathcal{M}) - h + 1$. Since $\mathsf{rank}(\mathcal{M}' - X) < \mathsf{rank}(\mathcal{M}) - h + 1$ if and only if $\mathsf{rank}(\mathcal{M}) - \mathsf{rank}(\mathcal{M} - X) \geq h$, $\mathcal{M}'$ has no k-fault-tolerant basis if and only if there is $X \subseteq E(\mathcal{M}')$ of size at most k such that $\mathsf{rank}(\mathcal{M}) - \mathsf{rank}(\mathcal{M} - X) \geq h$. This concludes the proof. ◀

We remark that since the hardness for Rank h-Reduction was proven by Fomin et al. [15] via a polynomial-time reduction from Clique, it is coNP-hard to decide whether a linear matroid has a k-fault-tolerant basis. We also note that the problem is in XP since we can decide in $n^{\mathcal{O}(k)}$ time whether an n-element matroid $\mathcal{M}$ has a k-fault-tolerant basis – simply check for each subset $X \subseteq E(\mathcal{M})$ of size k whether $\mathsf{rank}(\mathcal{M} - X) = \mathsf{rank}(\mathcal{M})$.

4 An FPT algorithm for the parameterization by rank and k

In this section, we prove Theorem 1. We construct a recursive branching algorithm that finds a set W of important elements of bounded size with the property that, if the input matroid $\mathcal{M}$ has a k-fault-tolerant basis, then there is a k-fault-tolerant basis $B \subseteq W$. Note that a k-fault-tolerant basis has minimum size by definition. Then, we select a k-fault-tolerant basis (if it exists) in W using brute force. The construction of W is inspired by Observation 7, which indicates that h-uniform sets are preferable in the construction of k-fault-tolerant bases. The following lemma is crucial for constructing W.

▶ **Lemma 8.** *Let $\mathcal{M}$ be a matroid of rank $r \geq 1$ and $k \geq 0$ be an integer. Let $X \subseteq E(\mathcal{M})$ be an h-uniform set of size at least $(h-1)[(k+1)r]^{r-1} + (k+1)r$ for some $1 \leq h \leq r$. Then, for any k-fault-tolerant basis B of $\mathcal{M}$, there is a k-fault-tolerant basis B' such that*

- **(i)** $B \setminus \mathsf{cl}(X) = B' \setminus \mathsf{cl}(X)$ *and*
- **(ii)** $B' \cap \mathsf{cl}(X) \subseteq X$.

Proof. Let B be a k-fault-tolerant basis of $\mathcal{M}$. If $B \subseteq \mathsf{cl}(X)$, then the claim is straightforward because in this case, $\mathsf{cl}(X) = E(\mathcal{M})$ and $r = h$. We can therefore select B' to be the set of $k+r$ arbitrary elements of X by Observation 7. We hence assume from now on that $B \setminus \mathsf{cl}(X) \neq \emptyset$.

Consider a k-fault-tolerant basis B' satisfying (i), that is, $B \setminus \mathsf{cl}(X) = B' \setminus \mathsf{cl}(X)$, such that the size of $B' \cap (\mathsf{cl}(X) \setminus X)$ is minimum. We claim that B' satisfies (ii), that is, $B' \cap \mathsf{cl}(X) \subseteq X$. Assume towards a contradiction that $B' \cap (\mathsf{cl}(X) \setminus X) \neq \emptyset$ and there is an element $x \in B' \cap (\mathsf{cl}(X) \setminus X)$. Let $Y = B' \setminus \{x\}$. Note that Y is not a k-fault-tolerant basis of $\mathcal{M}$. Hence, there is a set $F \subseteq Y$ of size at most k such that $\mathsf{rank}(Y \setminus F) < r$. Notice that $\mathsf{rank}(B' \setminus F) = r$ implies $\mathsf{rank}(Y \setminus F) = r - 1$. Denote by $\mathcal{F}$ the family of all sets $F \subseteq Y$ of size at most k such that $\mathsf{rank}(Y \setminus F) = r - 1$. For each $F \in \mathcal{F}$, let $Z_F \subseteq Y$ be an inclusion-maximal independent set in $Y \setminus F$ and let $\mathcal{Z} = \{Z_F \mid F \in \mathcal{F}\}$. Note that $\mathsf{cl}(Z_F) = \mathsf{cl}(Y \setminus F)$ and therefore $\mathsf{rank}(Z_F) = r - 1$ for each $F \in \mathcal{F}$. By Proposition 6, it holds that $|Y| \leq (k+1)r - 1$. Hence, $|\mathcal{Z}| \leq \binom{(k+1)r-1}{r-1} \leq [(k+1)r]^{r-1}$. We next show that $|\mathsf{cl}(Z_F) \cap X| \leq h - 1$ for each $F \in \mathcal{F}$.

To this end, assume towards a contradiction that $|\mathsf{cl}(Z_F) \cap X| \geq h$. Since X is h-uniform, we obtain that $X \subseteq \mathsf{cl}(\mathsf{cl}(Z_F) \cap X)$. Then $\mathsf{cl}(X) \subseteq \mathsf{cl}(\mathsf{cl}(Z_F) \cap X) \subseteq \mathsf{cl}(Z_F)$ and, in particular, $x \in \mathsf{cl}(Z_F) = \mathsf{cl}(Y \setminus F)$. However, in this case, $B' \setminus F = (Y \cup \{x\}) \setminus F \subseteq \mathsf{cl}(Y \setminus F)$. Since B' is a k-fault-tolerant basis, we have that $\mathsf{rank}(Y \setminus F) = \mathsf{rank}(B' \setminus F) = r$ contradicting $F \in \mathcal{F}$.

Recall that $|X| \geq (h-1)[(k+1)r]^{r-1} + (k+1)r$ and B' has at least one element outside X. Then, $|B'| \leq (k+1)r$ and therefore $|X \setminus B'| \geq (h-1)[(k+1)r]^{r-1} + 1$ by Proposition 6. Since $|\mathcal{Z}| < [(k+1)r]^{r-1}$ and, for each $F \in \mathcal{F}$, $|\mathsf{cl}(Z_F) \cap X| \leq h - 1$, a simple counting argument shows that there exists an element $y \in X \setminus B'$ such that $y \notin \mathsf{cl}(Z_F)$ for all $F \in \mathcal{F}$. Let $B^* = Y \cup \{y\} = (B' \setminus \{x\}) \cup \{y\}$. Note that by definition, (i) $B^* \setminus \mathsf{cl}(X) = B' \setminus \mathsf{cl}(X)$, (ii) $|B^* \cap (\mathsf{cl}(X) \setminus X)| < |B' \cap (\mathsf{cl}(X) \setminus X)|$, and (iii) $|B^*| = |B'|$. We next show that B^* is a k-fault-tolerant basis. Afterwards, we will show that this contradicts our choice of B'.

Towards the former, let $F \subseteq B^*$ be of size at most k. We prove that $\mathsf{rank}(B^* \setminus F) = r$. If $\mathsf{rank}(Y \setminus F) = r$, then $\mathsf{rank}(B^* \setminus F) \geq \mathsf{rank}(Y \setminus F) = r$. So we assume from now on that $\mathsf{rank}(Y \setminus F) < r$. Then, $\mathsf{rank}(Y \setminus F) = r - 1$ as shown above. Assume towards a contradiction that $y \in F$. Since $y \notin B'$, we have that $F' = (F \setminus \{y\}) \cup \{x\}$ is a subset of B' of size at most k. Since B' is a k-fault-tolerant basis, $\mathsf{rank}(B' \setminus F') = r$. However, $B' \setminus F' = Y \setminus F$ and $\mathsf{rank}(Y \setminus F) = r$, contradicting our assumption that $\mathsf{rank}(Y \setminus F) < r$. Hence, $y \notin F$ and therefore $F \subseteq Y$. Recall that $\mathsf{rank}(Y \setminus F) = r - 1$. This implies that $F \in \mathcal{F}$ and $Z_F \in \mathcal{Z}$. Since $y \notin \mathsf{cl}(Z_F)$ by the choice of y, $\mathsf{rank}(B^* \setminus F) = \mathsf{rank}((Y \setminus F) \cup \{y\}) > \mathsf{rank}(Y \setminus F) = r - 1$. Thus, $\mathsf{rank}(B^* \setminus F) = r$.

Since $\mathsf{rank}(B^* \setminus F) = r$ for every $F \subseteq B^*$ of size at most k and $|B^*| = |B'|$, we have that B^* is a k-fault-tolerant basis. However, we also obtain that (i) $B^* \setminus \mathsf{cl}(X) = B' \setminus \mathsf{cl}(X)$ and (ii) $|B^* \cap (\mathsf{cl}(X) \setminus X)| < |B' \cap (\mathsf{cl}(X) \setminus X)|$ contradicting the choice of B'. This shows that $B' \cap \mathsf{cl}(X) \subseteq X$ and completes the proof. ◀

Now we show how to construct a set W of important elements of bounded size.

▶ **Lemma 9.** *There is an algorithm that, given a matroid $\mathcal{M}$ of rank $r \geq 1$ with $\{e\} \in \mathcal{I}(\mathcal{M})$ for each $e \in E(\mathcal{M})$ and an integer $k \geq 0$, outputs a set $W \subseteq E(\mathcal{M})$ of at most $r^{r^2} \cdot [(k+1)r]^{r^3}$ elements in $(rk)^{\mathcal{O}(r^2)} \cdot n^{\mathcal{O}(1)}$ time such that $\mathcal{M}$ has a k-fault-tolerant basis if and only if $\mathcal{M}$ has a k-fault-tolerant basis $B \subseteq W$.*

Proof. Let $\mathcal{M}$ be a matroid of rank $r \geq 1$ such that $\{e\} \in \mathcal{I}(\mathcal{M})$ for any $e \in E(\mathcal{M})$ and let k be a non-negative integer. We construct a recursive branching algorithm $\textsc{Important}(\mathcal{M}, X)$ that takes as input a matroid $\mathcal{M}$ and a non-empty independent set $X \in \mathcal{I}(\mathcal{M})$, and outputs a set $Y \subseteq \mathsf{cl}(X)$ of size at most $|X|^{|X|^2} \cdot [(k+1)r]^{r|X|^2}$ with the property that for any k-fault-tolerant basis B of $\mathcal{M}$, there is a k-fault-tolerant basis B' such that

(i) $B \setminus \mathsf{cl}(X) = B' \setminus \mathsf{cl}(X)$ and
(ii) $B' \cap \mathsf{cl}(X) \subseteq Y$.

Let $h = |X|$. The base case is $h = 1$, where we do the following:

- Compute $\mathsf{cl}(X)$.
- If $|\mathsf{cl}(X)| < (k+1)r$, then set $Y := \mathsf{cl}(X)$ and output it.
- Otherwise, define Y to be the set of $(k+1)r$ arbitrary elements of $\mathsf{cl}(X)$ and output it.

For $h > 1$, we do the following:

- Compute $\mathsf{cl}(X)$.
- If $|\mathsf{cl}(X)| \leq (h-1)[(k+1)r]^{r-1} + (k+1)r$, then set $Y := \mathsf{cl}(X)$, output it, and stop.
- Find a h-uniform set $Z \subseteq \mathsf{cl}(X)$ that either has size $(h-1)[(k+1)r]^{r-1} + (k+1)r$ or is an inclusion-maximal h-uniform set of size at most $(h-1)[(k+1)r]^{r-1} + (k+1)r - 1$.
- If $|Z| = (h-1)[(k+1)r]^{r-1} + (k+1)r$, then set $Y := Z$ and output it.
- If Z is an inclusion-maximal h-uniform set of size at most $(h-1)[(k+1)r]^{r-1} + (k+1)r - 1$, then output $Y := \bigcup_{S \subseteq Z \text{ s.t. } |S|=h-1} Y_S$, where Y_S is the output of $\textsc{Important}(\mathcal{M}, S)$.

To compute an h-uniform set Z, we apply the following greedy procedure:

- Initially, set $Z := X$.
- While $Z \leq (h-1)[(k+1)r]^{r-1} + (k+1)r - 1$, do the following:
  - For every $x \in \mathsf{cl}(X) \setminus Z$, check whether $Z \cup \{x\}$ is h-uniform and set $Z := Z \cup \{x\}$ if this holds.
  - If $Z \cup \{x\}$ is not h-uniform for all $x \in \mathsf{cl}(X) \setminus Z$, then output Z and stop.
- If $Z = (h-1)[(k+1)r]^{r-1} + (k+1)r$, then output Z.

The crucial property of the algorithm is given in the following claim whose proof combines Lemma 8 and induction on h. Here and further, the proofs of the statements labeled ($\star$) are omited in this extended abstract and can be found in the full version of the paper [4].

▷ Claim 10 ($\star$). $\textsc{Important}(\mathcal{M}, X)$ outputs a set $Y \subseteq \mathsf{cl}(X)$ with the property that for any k-fault-tolerant basis B of $\mathcal{M}$, there is a k-fault-tolerant basis B' such that

(i) $B \setminus \mathsf{cl}(X) = B' \setminus \mathsf{cl}(X)$ and
(ii) $B' \cap \mathsf{cl}(X) \subseteq Y$.

It remains to analyze the size of Y and the running time. We next show an upper bound on the size of Y of $|Y| \leq h^{h^2} \cdot [(k+1)r]^{rh^2}$. We prove this via induction on h.

For $h = 1$, note that $|Y| \leq (k+1)r \leq h^{h^2} \cdot [(k+1)r]^{rh^2}$. For $h \geq 2$, consider any independent set $S \subseteq E(\mathcal{M})$ of size $h-1$. The algorithm outputs the set Y_S, which by the induction hypothesis, has size at most $N_{h-1} = (h-1)^{(h-1)^2}[(k+1)r]^{r(h-1)^2}$. By construction, we have

$$|Y| \leq \max\{(h-1)[(k+1)r]^{r-1} + (k+1)r, \binom{(h-1)[(k+1)r]^{r-1} + (k+1)r - 1}{h-1} N_{h-1}\}.$$

Observe first that

$$(h-1)[(k+1)r]^{r-1} + (k+1)r \leq h[(k+1)r]^r \leq h^{h^2} \cdot [(k+1)r]^{rh^2}. \tag{1}$$

For the second option, note that

$$\begin{aligned}
\binom{(h-1)[(k+1)r]^{r-1} + (k+1)r - 1}{h-1} N_{h-1} &\leq (h[(k+1)r]^r)^h \cdot N_{h-1} \\
&\leq (h[(k+1)r]^r)^h (h-1)^{(h-1)^2}[(k+1)r]^{r(h-1)^2} \\
&\leq h^h[(k+1)r]^{rh} \cdot h^{h^2-h}[(k+1)r]^{r(h^2-h)} \\
&\leq h^{h^2} \cdot [(k+1)r]^{rh^2}.
\end{aligned}$$

Combining this equation with Equation (1), we obtain the required upper bound for $|Y|$.

Finally, we evaluate the running time of Important and show that Important$(\mathcal{M}, X)$ runs in $(rk)^{\mathcal{O}(rh^2)} \cdot n^{\mathcal{O}(1)}$ time. To this end, notice that $\mathsf{cl}(X)$ can be constructed in polynomial time in the oracle model. To construct the h-uniform set Z of size $h(rk)^{\mathcal{O}(r)}$, we use the greedy procedure where for each $x \in \mathsf{cl}(X) \setminus Z$ for the considered Z, we go over all subsets S of Z of size $h-1$ and check whether $S \cup \{x\}$ is independent using the oracle. As $h \leq r$, the construction of Z can be done in $(kr)^{\mathcal{O}(rh)} \cdot n^{\mathcal{O}(1)}$ time. The algorithm makes $(rk)^{\mathcal{O}(rh)}$ recursive calls and the depth of the search tree is at most h. Summarizing, we obtain that the overall running time is $(rk)^{\mathcal{O}(rh^2)} \cdot n^{\mathcal{O}(1)}$.

To complete the proof and construct W, we call Important$(\mathcal{M}, X)$ for an arbitrary basis X of $\mathcal{M}$ and set $W = Y$ for the output set of the algorithm. Note that a basis can be found in polynomial time using the independence oracle. Then Claim 10 implies that if $\mathcal{M}$ has a k-fault-tolerant basis, then it also has one in W. Note that the other direction is trivial as $W \subseteq E(\mathcal{M})$. This concludes the proof. ◀

We are now ready to prove Theorem 1, which we restate here for convenience.

▶ **Theorem 1.** *Fault-Tolerant Basis can be solved in $(kr)^{\mathcal{O}(kr^4)} \cdot n^{\mathcal{O}(1)}$ time on n-element matroids of rank at most r given by independence oracles.*

Proof. Let $\mathcal{M}$ be a matroid of rank r and let $k \geq 0$ be an integer. Fault-Tolerant Basis is trivial for $r = 0$ and we can assume that $r \geq 1$. Notice that loops of $\mathcal{M}$, that is, elements e such that $\{e\} \notin \mathcal{I}(\mathcal{M})$ are irrelevant – a loop e is not included in any k-fault-tolerant basis and $e \in \mathsf{cl}(X)$ for any set $X \subseteq E(\mathcal{M})$. Hence, we can preprocess $\mathcal{M}$ and delete the loops. From now on, we assume that $\{e\} \in \mathcal{I}(\mathcal{M})$ for every $e \in E(\mathcal{M})$.

We apply the algorithm from Lemma 9, and in time $(rk)^{\mathcal{O}(r^3)} \cdot n^{\mathcal{O}(1)}$, find a set $W \subseteq E(\mathcal{M})$ of size at most $r^{r^2} \cdot [(k+1)r]^{r^3}$ such that whenever $\mathcal{M}$ has a k-fault-tolerant basis, $\mathcal{M}$ has a k-fault-tolerant basis $B \subseteq W$. We consider all candidate subsets B of W with $|B| \leq (k+1)r$ using the upper bound for the size of a k-fault-tolerant basis from Proposition 6. This

can be done in $(kr)^{\mathcal{O}(kr^4)}$ time. For each candidate set B of size at most $(k+1)r$, we verify whether B is a k-fault-tolerant basis as follows. In $\mathcal{O}([(k+1)r]^k)$ time, we check whether $\mathsf{rank}(B \setminus F) = r$ for every subset $F \subseteq B$ of size at most k. Among all candidate sets satisfying the above property, we select a set of minimum size which is a k-fault-tolerant basis of $\mathcal{M}$. The overall running time is $(kr)^{\mathcal{O}(kr^4)} \cdot n^{\mathcal{O}(1)}$. This concludes the proof. ◀

5 Partition matroids

In this section, we prove Proposition 5. The proof is based on the following structural lemma.

▶ **Lemma 11.** *Let $\mathcal{M}$ be a partition matroid with unit capacities for a partition $(P_1, \ldots, P_d)$ of the ground set. Let $X \subseteq E(\mathcal{M})$ and let $k \geq 0$ and $r \geq 1$ be integers. Then, the following properties hold.*

- **(i)** *If there is an integer $s \geq 1$ such that $|X| = s(r-1)+k+1$ and $|X \cap P_i| \leq s$ for every $i \in \{1, \ldots, d\}$, then for any set $F \subseteq X$ of size at most k, $\mathsf{rank}(X \setminus F) \geq r$.*
- **(ii)** *If X is an inclusion-minimal set such that for any set $F \subseteq X$ of size at most k, $\mathsf{rank}(X \setminus F) \geq r$, then there is an integer $s \geq 1$ such that $|X| = s(r-1)+k+1$ and $|X \cap P_i| \leq s$ for every $i \in \{1, \ldots, d\}$.*

Proof. To show (i), suppose that $|X| = s(r-1)+k+1$ and $|X \cap P_i| \leq s$ for each $i \in \{1, \ldots, d\}$ for some integer $s \geq 1$. Consider any subset $F \subseteq X$ of size at most k. Since $|X \cap P_i| \leq s$ for every $i \in \{1, \ldots, d\}$, it holds that $|(X \setminus F) \cap P_i| \leq s$ and hence $|(X \setminus F) \cap P_i| > 0$ for at least $\left\lceil \frac{|X \setminus F|}{s} \right\rceil$ sets P_i. Since $\mathcal{M}$ is a partition matroid with unit capacities, this is also a lower bound on $\mathsf{rank}(X \setminus F)$ and therefore

$$\mathsf{rank}(X \setminus F) \geq \left\lceil \frac{|X \setminus F|}{s} \right\rceil \geq \left\lceil \frac{s(r-1)+1}{s} \right\rceil \geq r.$$

This proves that $\mathsf{rank}(X) \geq r$ and $\mathsf{rank}(X \setminus F) \geq r$ for any set $F \subseteq X$ of size at most k.

To prove (ii), suppose that X is an inclusion-minimal set with $\mathsf{rank}(X) \geq r$ such that for any set $F \subseteq X$ of size at most k, $\mathsf{rank}(X \setminus F) \geq r$ holds. We assume without loss of generality that $|X \cap P_1| \geq \cdots \geq |X \cap P_d|$.

If $r = 1$, then we set $s = \max\{|X \cap P_i| \mid 1 \leq i \leq d\}$. Trivially, $|X| \geq k+1$ and $|X \cap P_i| \leq s$ for every $i \in \{1, \ldots, d\}$. We next show that $|X| = k+1$. Assume towards a contradiction that $|X| > k+1$. Any subset $X' \subseteq X$ of size $k+1$ satisfies $|X' \cap P_i| \leq s$ for every $i \in \{1, \ldots, d\}$. Hence, $\mathsf{rank}(X' \setminus F) > 1$ for any set F of size at most k by (i). This contradicts the the minimality of X and shows $|X| = k+1$.

If $r \geq 2$, then $|X \cap P_{r-1}| \geq 1$ and we set $s = |X \cap P_{r-1}|$. We will next show that $|X| = s(r-1)+k+1$ and $|X \cap P_i| \leq s$ for every $i \in \{1, \ldots, d\}$. Let $Y = \bigcup_{j=1}^{r-1}(X \cap P_j)$ and $Z = \bigcup_{j=r}^{d}(X \cap P_j)$. By definition of s, $|Y| \geq s(r-1)$.

If $|Z| \leq k$, then $\mathsf{rank}(X \setminus Z) = r-1 < r$, contradicting the choice of X. Thus, $|Z| \geq k+1$ and $|X| = |Y| + |Z| \geq s(r-1)+k+1$. There exist $Y' \subseteq Y$ such that $|Y' \cap P_j| = s$ for each $j \in \{1, \ldots, r-1\}$, and $Z' \subseteq Z$ such that $|Z'| = k+1$. Consider $X' = Y' \cup Z'$. By definition, $|X' \cap P_i| \leq s$ for every $i \in \{1, \ldots, d\}$. Then, by (i), we have that $\mathsf{rank}(X') \geq r$ and for any set $F \subseteq X'$ of size at most k, $\mathsf{rank}(X' \setminus F) \geq r$. By the minimality of X, we conclude that $X = X'$. This proves (ii). ◀

We are now ready to prove Proposition 5, which we restate here.

▶ **Proposition 5.** *Given an n-element partition matroid $\mathcal{M}$ with unit capacities together with a weight function $w\colon E(\mathcal{M}) \to \mathbb{Z}_{\geq 0}$ and integers $r, k \geq 0$, one can find in $\mathcal{O}(n^2)$ time either a set $X \subseteq E(\mathcal{M})$ of minimum weight such that, for any set $F \subseteq X$ of size at most k, it holds that $\mathsf{rank}(X \setminus F) \geq r$ or correctly decide that such a set does not exist.*

Proof. Consider a weighted partition matroid $\mathcal{M}$ with unit capacities and a weight function $w\colon E(\mathcal{M}) \to \mathbb{Z}_{\geq 0}$. Let $k \geq 0$ and $r \geq 0$ be integers. Let also $(P_1, \ldots, P_d)$ be the partition of $E(\mathcal{M})$ defining $\mathcal{M}$. If $r = 0$, then the claim of the theorem is trivial as $X = \emptyset$ is a solution. Thus, we can assume that $r \geq 1$.

For an integer $s \geq 1$, we say that s is *feasible* if there exists a set $X \subseteq E(\mathcal{M})$ with $|X| = s(r-1) + k + 1$ such that $|X \cap P_i| \leq s$ for every $i \in \{1, \ldots, d\}$. For each integer $1 \leq s \leq \max\{|P_i| \mid 1 \leq i \leq d\}$, we check whether s is feasible. This can be done by checking whether $\sum_{i=1}^{d} |\min\{|P_i|, s\}| \geq s(r-1) + k + 1$. If there is no feasible integer, then we conclude that there is no $X \subseteq E(\mathcal{M})$ with the property that $\mathsf{rank}(X \setminus F) \geq r$ for any subset $F \subseteq X$ of size at most k by Lemma 11. Otherwise, for every feasible s, we greedily choose a set X of minimum weight such that $|X| = s(r-1) + k + 1$ and $|X \cap P_i| \leq s$ for every $i \in \{1, \ldots, d\}$. Notice that the selection of such a set is equivalent to finding a minimum weight independent set of size $s(r-1) + k + 1$ in the partition matroid for $(P_1, \ldots, P_d)$ with the capacities $(c_1, \ldots, c_d)$ where $c_i = \min\{s, |P_i|\}$ for each $i \in \{1, \ldots, d\}$. By the well-known matroid properties [21], the greedy algorithm finds such a set X. By Lemma 11, we conclude that X is a minimum weight set such that for any set $F \subseteq X$ of size at most k, $\mathsf{rank}(X \setminus F) \geq r$. This concludes the description of our algorithm and its correctness proof.

To evaluate the running time, notice that the elements of $\mathcal{M}$ can be sorted by their weight in $\mathcal{O}(n \log n)$ time. Then we have at most n choices of s and for each s, the greedy algorithm works in $\mathcal{O}(n)$ time. Thus, the overall running time is in $\mathcal{O}(n^2)$, concluding the proof. ◀

As a corollary, we obtain the following for general matroids of rank at most two.

▶ **Corollary 12** ($\star$). *Weighted Fault-Tolerant Basis can be solved in $\mathcal{O}(n^2)$ on matroids of rank at most two given by an independence oracle.*

6 Complexity dichotomy for the parameterization by rank

In this section, we prove Theorem 4 which we restate here for convenience.

▶ **Theorem 4.** *For any integer $r \geq 3$, it is* NP*-hard to decide whether a linear matroid $\mathcal{M}$ of rank r over rationals given together with integers k and b has a k-fault-tolerant basis of size at most b. For $r \leq 2$, Weighted Fault-Tolerant Basis can be solved in $\mathcal{O}(n^2)$ on matroids given by an independence oracle.*

Proof. The claim for $r \leq 2$ is proven in Corollary 12. Thus, it remains to show the computational lower bound for $r \geq 3$. We show the claim for $r = 3$ and then explain how to extend it for any $r \geq 3$. We reduce from General Position Subset Selection. Recall that a set P of points on the Euclidean plane is said to be in general position if there are no three points on the same line on the plane. General Position Subset Selection is defined as follows. Given a set P of point on the plane and an integer $p \geq 1$, decide whether there is a subset $Q \subseteq P$ of at least p points in general position. The problem was shown to be NP-hard by Froese et al. [16] and the result holds for points with rational coordinates.

Let $(P = \left\{ \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \ldots, \begin{pmatrix} x_n \\ y_n \end{pmatrix} \right\}, p)$ be an instance of General Position Subset Selection. We assume without loss of generality that $p \geq 3$ as any set of at most two points is in general position. We set $k = p - 3$, construct the set $E = \left\{ \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}, \ldots, \begin{pmatrix} x_n \\ y_n \\ 1 \end{pmatrix} \right\}$ of vectors, and define $\mathcal{M}$ to be the linear matroid with the ground set E over rationals.

Note that $\mathsf{rank}(\mathcal{M}) \leq 3$ and that three distinct points $\begin{pmatrix} x_h \\ y_h \end{pmatrix}$, $\begin{pmatrix} x_i \\ y_i \end{pmatrix}$, and $\begin{pmatrix} x_j \\ y_j \end{pmatrix}$ of P are not on the same line if and only if $\mathsf{rank}\left(\left\{ \begin{pmatrix} x_h \\ y_h \\ 1 \end{pmatrix}, \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}, \begin{pmatrix} x_j \\ y_j \\ 1 \end{pmatrix} \right\}\right) = 3$. This implies that for any $I \subseteq \{1, \ldots, n\}$, the points of $\left\{ \begin{pmatrix} x_i \\ y_i \end{pmatrix} \mid i \in I \right\} \subseteq P$ are in general position if and only if $\left\{ \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \mid i \in I \right\} \subseteq E$ is a 3-uniform set for $\mathcal{M}$. Hence, P has a subset of at least p points in general position if and only if there is a 3-uniform subset of E of size at least p. By Proposition 6, a k-fault-tolerant basis of $\mathcal{M}$ has size at least $p = k + 3$. By Observation 7, $\mathcal{M}$ has a k-fault-tolerant basis of size p if and only if (P, p) is a yes-instance of General Position Subset Selection. This concludes the proof for $r = 3$.

To see the claim for $r > 3$, we perform the same reduction but in r dimensions. The constructed vectors have a 0 in all but the first three dimensions (where they have the entries as constructed above). For each $4 \leq i \leq r$, we then add $k + 1$ vectors with zero-entries everywhere except for dimension i, where the entry is 1. Note that any k-fault-tolerant basis of the linear matroid over the constructed vectors has to contain all $(r - 3)(k + 1)$ newly added vectors. Moreover, it has to contain a set P of vectors such that after removing any k of them, the first three dimensions have to be spanned by the remaining vectors. This is equivalent to the case $r = 3$ and concludes the proof. ◀

7 Conclusion

In this paper, we initiate the study of the parameterized complexity of Fault-Tolerant Basis. Our main result is that the problem is fixed-parameter tractable when parameterized by both k and rank r of the input matroid. This positive algorithmic result is complemented by computational lower bounds showing NP-hardness for constant values for any one of the two parameters alone. Our results lead to several open questions.

First, we do not know whether our FPT result from Theorem 1 could be extended for Weighted Fault-Tolerant Basis. Our approach based on the choice of h-uniform sets is tailored for the unweighted case, and one may need different techniques in the presence of weights.

Second, is it possible to extend our result to the non-uniform model introduced by Adjiashvili et al. [1]? Here, we assume that the set of elements of a matroid $\mathcal{M}$ is partitioned into two subsets S and V of *safe* and *vulnerable* elements, respectively. Then, the task is to either find a set $B \subseteq E(\mathcal{M})$ of minimum size such that for any sets $F \subseteq V \cap B$ of size at most k, $\mathsf{rank}(\mathcal{M} - F) = \mathsf{rank}(\mathcal{M})$ or correctly report that such a set does not exist. Fault-Tolerant Basis is the special case of this problem with $S = \emptyset$.

Third, can our results be extended to the model introduced by Adjiashvili et al. [2] where not arbitrary sets of k elements can fail but possible failure scenarios are part of the input?

Finally, observe that our computational lower bounds from Proposition 2, Observation 3, and Theorem 4 do not exclude efficient algorithms for Fault-Tolerant Basis on special classes of matroids. In particular, we proved in Proposition 5 that Weighted Fault-Tolerant Basis can be solved in polynomial time on truncations of partition matroids with unit capacities. While it is straightforward to see that Weighted Fault-Tolerant

Basis can be solved in polynomial time for partition matroids with arbitrary capacities,[1] it is not clear whether the problem can be solved in polynomial time on their truncations. The problem complexity for other fundamental classes of matroids, like transversal matroids and gammoids, is another interesting open question.

References

1 David Adjiashvili, Felix Hommelsheim, Moritz Mühlenthaler, and Oliver Schaudt. Fault-tolerant edge-disjoint *s-t* paths – beyond uniform faults. In *Proceedings of the 18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 5:1–5:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SWAT.2022.5.

2 David Adjiashvili, Sebastian Stiller, and Rico Zenklusen. Bulk-robust combinatorial optimization. *Mathematical Programming*, 149(1-2):361–390, 2015. doi:10.1007/S10107-014-0760-6.

3 Ishan Bansal, Joseph Cheriyan, Logan Grout, and Sharat Ibrahimpur. Improved approximation algorithms by generalizing the primal-dual method beyond uncrossable functions. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 15:1–15:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.15.

4 Matthias Bentert, Fedor V. Fomin, Petr A. Golovach, and Laure Morelle. Fault-tolerant matroid bases, 2025. doi:10.48550/arXiv.2506.22010.

5 Matthias Bentert, Jannik Schestag, and Frank Sommer. On the complexity of finding a sparse connected spanning subgraph in a non-uniform failure model. In *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 4:1–4:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.4.

6 Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53:464–501, 2011. doi:10.1137/080734510.

7 Sylvia C. Boyd, Joseph Cheriyan, Arash Haddadan, and Sharat Ibrahimpur. Approximation algorithms for flexible graph connectivity. In *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 9:1–9:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.FSTTCS.2021.9.

8 Parinya Chalermsook, Chien-Chung Huang, Danupon Nanongkai, Thatchaphol Saranurak, Pattara Sukprasert, and Sorrachai Yingchareonthawornchai. Approximating k-edge-connected spanning subgraphs via a near-linear time LP solver. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 37:1–37:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.37.

9 Chandra Chekuri and Rhea Jain. Approximation algorithms for network design in non-uniform fault models. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.36.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Alex G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin J. Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551, 2010. doi:10.1109/TIT.2010.2054295.

[1] Given a partition matroid defined by a partition $(P_1, \ldots, P_d)$ of the ground set and a d-tuple of capacities $(c_1, \ldots, c_d)$, to solve Weighted Fault-Tolerant Basis, we have to choose $c_i + k$ elements of minimum weight from each P_i.

12 Mitre Costa Dourado, Dirk Meierling, Lucia D Penso, Dieter Rautenbach, Fabio Protti, and Aline Ribeiro de Almeida. Robust recoverable perfect matchings. *Networks*, 66(3):210–213, 2015. doi:10.1002/NET.21624.

13 Michael Elad. *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. Springer, 2010. doi:10.1007/978-1-4419-7011-4.

14 Cristina G. Fernandes. A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem. *Journal of Algorithms*, 28(1):105–124, 1998. doi:10.1006/JAGM.1998.0931.

15 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Covering vectors by spaces: Regular matroids. *SIAM Journal on Discrete Mathematics*, 32(4):2512–2565, 2018. doi:10.1137/17M1151250.

16 Vincent Froese, Iyad A. Kanj, André Nichterlein, and Rolf Niedermeier. Finding points in general position. *International Journal of Computational Geometry and Applications*, 27(4):277–296, 2017. doi:10.1142/S021819591750008X.

17 Harold N. Gabow, Michel X. Goemans, Éva Tardos, and David P. Williamson. Approximating the smallest k-edge connected spanning subgraph by LP-rounding. *Networks*, 53(4):345–357, 2009. doi:10.1002/NET.20289.

18 Felix Hommelsheim, Moritz Mühlenthaler, and Oliver Schaudt. How to secure matchings against edge failures. *SIAM Journal on Discrete Mathematics*, 35(3):2265–2292, 2021. doi:10.1137/20M1336229.

19 Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 759–770. ACM, 1992. doi:10.1145/129712.129786.

20 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Transactions on Algorithms*, 14(2):14:1–14:20, 2018. doi:10.1145/3170444.

21 James G. Oxley. *Matroid theory*. Oxford University Press, 1992.

Property Testing of Curve Similarity

Peyman Afshani
Department of Computer Science, Aarhus University, Denmark

Maike Buchin
Faculty of Computer Science, Ruhr University Bochum, Germany

Anne Driemel
Institute for Computer Science, University of Bonn, Germany

Marena Richter
Institute for Computer Science, University of Bonn, Germany

Sampson Wong
Department of Computer Science, University of Copenhagen, Denmark

Abstract

We propose sublinear algorithms for probabilistic testing of the discrete and continuous Fréchet distance – a standard similarity measure for curves. We assume the algorithm is given access to the input curves via a query oracle: a query returns the set of vertices of the curve that lie within a radius δ of a specified vertex of the other curve. The goal is to use a small number of queries to determine with constant probability whether the two curves are similar (i.e., their discrete Fréchet distance is at most δ) or they are "ε-far" (for $0 < \varepsilon < 2$) from being similar, i.e., more than an ε-fraction of the two curves must be ignored for them to become similar.

We present two algorithms which are sublinear assuming that the curves are t-approximate shortest paths in the ambient metric space, for some $t \ll n$. The first algorithm uses $O(\frac{t}{\varepsilon} \log \frac{t}{\varepsilon})$ queries and is given the value of t in advance. The second algorithm does not have explicit knowledge of the value of t and therefore needs to gain implicit knowledge of the straightness of the input curves through its queries. We show that the discrete Fréchet distance can still be tested using roughly $O(\frac{t^3+t^2 \log n}{\varepsilon})$ queries ignoring logarithmic factors in t. Our algorithms work in a matrix representation of the input and may be of independent interest to matrix testing.

Our algorithms use a mild uniform sampling condition that constrains the edge lengths of the curves, similar to a polynomially bounded aspect ratio. Applied to testing the continuous Fréchet distance of t-straight curves, our algorithms can be used for $(1+\varepsilon')$-approximate testing using essentially the same bounds as stated above with an additional factor of $\text{poly}(\frac{1}{\varepsilon'})$.

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

Keywords and phrases Fréchet distance, Trajectory Analysis, Curve Similarity, Property Testing, Monotonicity Testing

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.84

Related Version *Full Version*: https://arxiv.org/abs/2502.17277

Funding *Anne Driemel*: Affiliated with Lamarr Institute for Machine Learning and Artificial Intelligence.

Acknowledgements Anne Driemel would like to thank Ronitt Rubinfeld and Ilan Newman for helpful discussions in the early stages of this research. The authors thank Sariel Har-Peled for feedback on an earlier version of this manuscript.

1 Introduction

We initiate the study of property testing for measures of curve similarity, motivated by the need for fast solutions for curve classification and clustering. Thus, our research lies at the intersection of property testing and computational geometry. While property testing is a very broad area, we believe this intersection has received far less attention than it deserves.


© Peyman Afshani, Maike Buchin, Anne Driemel, Marena Richter, and Sampson Wong;
licensed under Creative Commons License CC-BY 4.0



We also believe that property testing for well-studied measures such as the Fréchet distance is especially well-motivated due to its connections to other problems studied on the curves, such as clustering [3, 8], similarity search [1, 7, 14] and map reconstruction [4, 5].

Typically in property testing, we are given access to a (large) data set and the goal is to very quickly assess whether the data has a certain property. The classical notation of correctness is typically not suitable for property testing since these algorithms are intended to be randomized algorithms that access (very small) parts of data via queries. Instead, a property testing algorithm is considered correct if it can satisfy the following two conditions, with a probability close to 1 (e.g., at least $1-\gamma$ probability for some small value of γ): first, if the input has the desired property, the algorithm must return *accept* and second, if the input is "far" from having the property (under some suitable definition of "far"), the algorithm should *reject* the input. Our algorithms have *one-sided error*, meaning, they will not produce *false negatives* and thus rejection means that the input does not have the desired property.

For further reading on property testing, see [24, 2]. Here, we quickly review the main motivation for property testing in general and then we focus on computational geometry.

Property testing algorithms can be useful in many different scenarios. In particular, if the input is extremely large, it makes sense to obtain a quick approximate answer before deciding to run more expensive algorithms. This is especially useful if the testing algorithm has one-sided error; in that case, if the test returns *reject*, there is no need to do any extra work. For the Fréchet distance, there are applications [1, 7, 14] where negative filters are used to minimize expensive Fréchet distance computations; testing may be useful in such cases. Property testing is also useful if most inputs are expected to not have the desired property, or the input distribution is such that each input either has the property or it is "far" from having that property. In addition, the definition of property testing has a strong connection to topics in learning theory such as *probably approximately correct (PAC)* learners and this was in fact one of the important motivations behind its conception [23].

Another motivation for property testing is when small errors can be tolerated or objects that are close to having the desired property are acceptable outcomes; in this view, property testing is in spirit similar to approximation algorithms where an approximate answer is considered to be an acceptable output. In the context of similarity testing we can also compare it to a partial similarity measure. In fact, our chosen error model will be very close to a partial Fréchet distance [6]. For more details on motivations to study property testing see [17, 24]. See also the aforementioned references for more detailed results on property testing as surveying the known results on property testing is beyond the scope of this paper.

Computational geometry has a long tradition of using randomization and sampling to speed up algorithmic approaches [22, 19, 20, 18]. Property testing has received some attention within computational geometry, but is much less explored compared to other areas. There are fast and efficient testers for many basic geometric properties, such as convexity of a point set [13], disjointness of objects [13], the weight of Euclidean minimum spanning tree [10, 11, 12], clustering [21], the Voronoi property of a triangulation [13], ray shooting [9, 13], as well as LP-type problems [15].

Unlike in graph property testing, where it is assumed that we can sample a vertex uniformly at random, and query for its neighbors, it seems that, for geometric data, there is no commonly agreed upon query model. Different types of queries are used to access the data in the above cited works, such as range queries, cone nearest neighbor queries, queries for the bounding box of the data inside a query hyper-rectangle. In our paper, we use queries to the free space matrix of the two input curves, see Section 3 for the precise definition.

We mention that there is also work on matrix testing [16]. Compared to our setting, the input size is defined as n^2 – the number of entries needed to specify the matrix. In our case, we assume the input size to be n, the maximum of column and row dimension of the matrix.

2 Preliminaries

Let (M, d) be a metric space. We say a *(discrete) curve* P in (M, d) is an ordered point sequence $\langle p_1, \ldots, p_n \rangle$ with $p_i \in M$ for all $i = 1, \ldots, n$. We call the points of the curve *vertices*. We denote by $|P|$ the number of vertices in P and by $\ell(P)$ its *length*, which is defined as $\ell(P) = \sum_{i=1}^{n-1} d(p_i, p_{i+1})$. The *subcurve* of P between p_i and p_j is denoted by $P[i, j]$. A curve P is called *t-straight* if for any two vertices p_i and p_j in P, we have $\ell(P[i, j]) \leq t \cdot d(p_i, p_j)$. Denote by $[n]$ the set of integers from 1 to n and by $[n] \times [n] \subset \mathbb{N} \times \mathbb{N}$ the integer lattice of n times n integers. Given two curves $P = \langle p_1, \ldots, p_n \rangle$ and $Q = \langle q_1, \ldots, q_n \rangle$, we say that an ordered sequence $\mathcal{C}$ of elements in $[n] \times [n]$ is a *coupling* of P and Q, if it starts in $(1, 1)$, ends in (n, n) and for any consecutive tuples $(i, j), (i', j')$ in $\mathcal{C}$ it holds that $(i', j') \in \{(i + 1, j), (i, j + 1), (i + 1, j + 1)\}$. We define the *discrete Fréchet distance* between P and Q as follows

$$D_{\mathcal{F}}(P, Q) \coloneqq \min_{\text{coupling } \mathcal{C}} \max_{(i,j) \in \mathcal{C}} d(p_i, q_j).$$

For brevity, we simply call this the Fréchet distance between P and Q. The *free space matrix* of P and Q with distance value δ is an $n \times n$ matrix M_δ, where the i-th column corresponds to the vertex p_i of P and the j-th row corresponds to the vertex q_j of Q.

The entry $M_\delta[i, j]$ has the value 0 if $d(p_i, q_j) \leq \delta$ and 1 otherwise.[1] A coupling path $\mathcal{C}$ is the path through the $[n] \times [n]$ integer lattice defined by the coupling $\mathcal{C}$. We define the *cost* of such a path as $c(\mathcal{C}) = \sum_{(i,j) \in \mathcal{C}} M_\delta[i, j]$. Note that the Fréchet distance between P and Q is at most δ if and only if there exists a coupling path with cost 0 from $(1, 1)$ to (n, n).

Our analysis is based on a property of the free space matrix. We first define this property and then link the property to a certain class of well-behaved input curves.

▶ **Definition 1** (t-local). *Let M be a free space matrix of curves P and Q. We say that M is* t-local *if, for any tuples (i_1, j_1) and (i_2, j_2) with $M[i_1, j_1] = 0 = M[i_2, j_2]$, it holds that $|i_1 - i_2| \leq t \cdot (2 + |j_1 - j_2|)$ and $|j_1 - j_2| \leq t \cdot (2 + |i_1 - i_2|)$.*

See Figure 1 for a visualization. A consequence of this property is that zero-entries within one row or within one column cannot be too far away from each other.

▶ **Observation 2.** *Suppose M is t-local. If we have $M[i, j] = 0 = M[i, j']$, then we have $|j - j'| \leq 2t$. If we have $M[i, j] = 0 = M[i', j]$, we have $|i - i'| \leq 2t$.*

In the following, we argue why we think the t-locality assumption is reasonable. Note that t will not exceed n on any input. There are classes of well-behaved curves that satisfy the locality assumption for some $t \ll n$. Lemma 3 below shows that the free space matrix of approximate shortest paths is t-local. For simplicity, we impose the assumption that the lengths of the edges are bounded by a fixed multiple of δ, the query radius of the Fréchet distance. In the full version, we show that our approach can be adapted to work as well if the lengths of the edges are bounded by a constant multiple of any fixed value.

[1] Note we use 0 and 1 in switched roles compared to the conventions in the literature on the free space matrix. Our notation makes the cost-function of the path more intuitive.

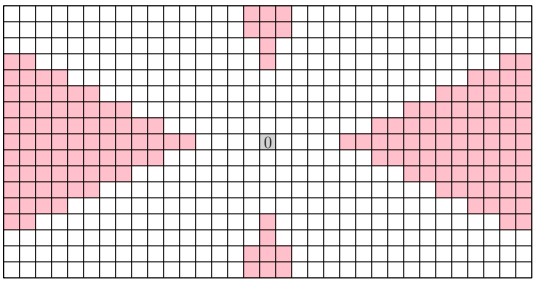


Figure 1 The red cells cannot be zero-entries if the gray cell is a zero-entry and the matrix is 2-local.

▶ **Lemma 3.** *Let P and Q be t-straight curves with edge lengths in $[\delta/\alpha, \alpha\delta]$ for some constant $\alpha \geq 1$. Let M be their free space matrix with distance value δ. Then, M is $\mathcal{O}(t)$-local.*

Proof. Let $(i_1, j_1), (i_2, j_2)$ be such that $M[i_1, j_1] = 0 = M[i_2, j_2]$ and $i_1 \leq i_2$. So we have $d(p_{i_1}, q_{j_1}) \leq \delta$ and $d(p_{i_2}, q_{j_2}) \leq \delta$. We also have $d(q_{j_1}, q_{j_2}) \leq \ell(Q[j_1, j_2]) \leq \alpha\delta\,|j_1 - j_2|$ because of the edge length constraint, where $Q[j_1, j_2]$ is replaced by $Q[j_2, j_1]$ if $j_2 < j_1$. Using the edge length constraint and t-straightness we derive $\frac{\delta}{\alpha}\,|i_2 - i_1| \leq \ell(P[i_1, i_2]) \leq t d(p_{i_1}, p_{i_2})$. These observations together with the triangle inequality yield

$$|i_2 - i_1| \leq \frac{\alpha t}{\delta}(d(p_{i_1}, q_{j_1}) + d(q_{j_1}, q_{j_2}) + d(q_{j_2}, p_{i_2})) \leq \alpha^2 t(2 + |j_2 - j_1|).$$

The other inequality is shown by reversing the roles of P and Q. ◀

3 Problem definition and results

The problem we study in this paper is the following. Assume we want to determine for two curves P and Q, each consisting of n vertices,[2] if their Fréchet distance is at most a given value of δ. We do not have direct access to the input curves. Instead, we have access to an oracle that returns the information in a given row or column of the δ-free space matrix in the form of a sorted list of indices of zero-entries. We call this a *query* and we want to determine $D_{\mathcal{F}}(P, Q) \leq \delta$ using as few (sublinear in n) queries as possible. Note that from the point of view of a data structure setting, our query corresponds to a classical ball range query with a vertex p of one curve and returns the list of vertex indices of the other curve that are contained in the ball of radius δ centered at p.

Our bounds on the number of queries will be probabilistic and will hold under a certain error model that allows for the coupling path to pass through a bounded number of one-entries. These are entries where the corresponding points are far from each other. In the full version, we discuss alternative error models and how they relate to our chosen error model.

▶ **Definition 4** ((ε, δ)-far)**.** *Given two curves P and Q consisting of n vertices each, we say that P and Q are (ε, δ)-far from each other if there exists no coupling path from $(1, 1)$ to (n, n) in the δ-free space matrix of cost εn or less.*

▶ **Definition 5** (Fréchet-tester)**.** *Assume we are given query-access to two curves P and Q, and we are given values $\delta > 0$ and $0 < \varepsilon < 2$. If the two curves have Fréchet distance at most δ, we must return "yes", and if they are (ε, δ)-far from each other w.r.t. the Fréchet distance, the algorithm must return "no", with probability at least $\frac{4}{5}$.*

[2] For ease of notation, our analysis assumes the input curves have the same number of vertices.

Results. We present two algorithms for testing whether a pair of input curves have discrete Fréchet distance at most a given real value δ. Assume that the algorithm has query access only to the δ-free space matrix of the input curves and that this matrix is t-local (see Definition 1). The first algorithm requires knowledge of t, and it uses $O(\frac{t}{\varepsilon}\log\frac{t}{\varepsilon})$ queries (Theorem 18). Note that this bound is intrinsically independent of n while t can be at most n, by the definition of locality. The second algorithm does not require knowledge of t in advance (and thus it can be applied to any matrix) and requires $\mathcal{O}((t^3+t^2\log n)\frac{\log\log t}{\varepsilon})$ many queries (Theorem 25). This algorithm gains its knowledge of t with a variant of monotonicity testing (Theorem 23). Using Lemma 3 these results directly imply the same bounds for the discrete Fréchet distance of t-straight curves with bounded edge lengths. In Section 6 we show how to apply our algorithms to testing the continuous Fréchet distance. As a result, we obtain essentially the same bounds with respect to t-straight curves while removing the assumption on the edge lengths entirely (Theorem 29).

Techniques. Our algorithms use the concept of permeability, which tests whether a specified subcurve has small partial Fréchet distance to any subcurve of the other curve. Concretely, the algorithm samples a block of consecutive columns (or rows) in the δ-free space matrix and checks if there is a coupling path of cost 0 passing somewhere through this block (see Algorithm 1). If the algorithm happens to find a block that does not satisfy this condition, then we know that the global Fréchet distance is larger than the distance parameter δ and the algorithm can safely return 'no'. The difficulty lies in proving that after a certain number of random queries, the algorithm can return 'yes' with sufficient certainty. To this end, we show that if the two curves are (ε,δ)-far then the algorithm, which tests subcurves of varying sizes up to roughly $O(\frac{t}{\varepsilon})$, is likely to sample a permeability query that would return 'no'.

In the full version, we discuss a simple tester for the Hausdorff distance and a simple t-approximate Fréchet tester. This algorithm merely tests for columns and rows that contain one-entries only and are therefore somewhat blocked for coupling paths in the free space matrix. The example in Figure 2 shows that such simple queries are not sufficient to test the Fréchet distance exactly.

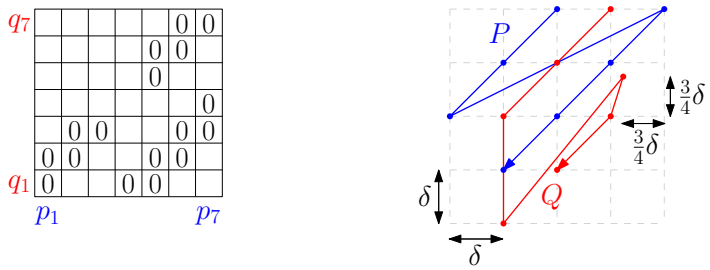


Figure 2 On the left we have M_δ for the curves on the right. All zero-entries are written.

4 Testing the discrete Fréchet distance

In this section, we describe and analyze a Fréchet tester under the assumption that the free space matrix is t-local for a given value of t.

4.1 The algorithm

The idea of Algorithm 1 is to sample a number of columns and rows and check whether there is locally a coupling path of cost zero possible. The following definition classifies when such a (local) path exists.

▶ **Definition 6** (Permeability). *We say a block $[i, i']$ of consecutive columns (resp., rows) from index i to index i' is* permeable *if there exists a coupling path of cost zero that starts in column (resp., row) i and ends in column (resp., row) i'.*

If a column (resp. row) is individually not permeable, i.e. it contains only one-entries, we call it a *barrier-column* (resp. *barrier-row*). Note that any non-permeable block of rows or columns is a witness to the fact that no global coupling path of cost 0 exists and that the algorithm can answer "no".

The algorithm first tests if the first or last entry (i.e. $M[1,1]$ or $M[n,n]$) is a one-entry. If not, it queries $\mathcal{O}\left(\frac{t}{\varepsilon}\right)$ randomly sampled columns and rows and checks if any of them is a barrier-column or barrier-row. In Lines 6-8 we sample $\mathcal{O}(\frac{t}{2^{i+1}})$ random intervals of length 2^{i+2} for each i from 0 to $\mathcal{O}\left(\log \frac{t}{\varepsilon}\right)$. We then check all corresponding blocks of columns and rows for permeability and if all blocks are permeable, we return "yes".

Algorithm 1 Fréchet-Tester1$(M, t, \varepsilon, k = \frac{24t}{\varepsilon}, c = 4)$.

1. **If** $M[1,1] = 1$ or $M[n,n] = 1$ **then return** "no".
2. **repeat** $\lceil k \rceil$ **times**:
3. $\quad j \leftarrow$ sample an index uniformly at random from $[n]$.
4. $\quad$ **if** row j or column j of M is a barrier-column or barrier-row **then return** "no".
5. $K \leftarrow \lceil \frac{\varepsilon n}{32t} \rceil - 1$, $\ell \leftarrow \lceil \frac{128t}{\varepsilon} \rceil$, let $\mathcal{J}$ be a set of intervals and set $\mathcal{J} \leftarrow \emptyset$.
6. **for** $i = 0, \ldots, \lfloor \log_2 \ell \rfloor$ **do**:
7. $\quad I \leftarrow$ sample $\lceil \frac{4cn}{2^{i+1}K} \rceil$ different indices uniformly at random from $\{0, 1, \ldots, \frac{n}{2^{i+1}} - 2\}$.
8. $\quad$ **for each** $j \in I$ **do**: add $[j2^{i+1}, (j+2)2^{i+1}]$ to $\mathcal{J}$.
9. **foreach** $[i, j] \in \mathcal{J}$ **do**
10. $\quad$ **if** block $[i, j]$ of consecutive columns is not permeable **then return** "no".
11. $\quad$ **if** block $[i, j]$ of consecutive rows is not permeable **then return** "no".
12. **return** "yes".

To check if a block of k columns (or rows) is permeable, the algorithm first performs k queries to obtain the positions of zero-entries in these columns (or rows), then we build the induced subgraph of the grid for these zero-entries, connect all neighboring zero-entries according to the possible steps of a coupling and then connect all zero-entries of the last column to a sink and all zero-entries of the first column from a source. It remains to check if there is a path from source to sink, which can be done in linear time since the graph is acyclic. The running time is linear in the total number of zero-entries queried. We call this a *permeability query*.

4.2 Basic properties of permeability

For two curves P and Q with n vertices each, and a t-local free space matrix, Algorithm 1 returns "no" only if it has determined that no coupling path of cost zero from $(1,1)$ to (n,n) can exist (by finding blocks of rows or columns that are not permeable). We need to show that it returns "yes" correctly with sufficiently high probability.

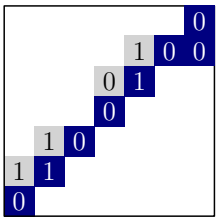


Figure 3 The blue path is a coupling path of cost 2. By adding the gray entries, we get a diagonal-restricted path of cost 5.

For technical reasons, our analysis uses a specific kind of coupling paths that are more restricted than the ones given in Section 2. We introduce them here and argue that it suffices to consider these coupling paths only.

▶ **Definition 7** (diagonal-restricted path). *A* diagonal-restricted path $\mathcal{C}$ *through the free space matrix* M *is a path corresponding to a coupling* $\mathcal{C}$ *in which for any consecutive tuples of the form* $(i,j),(i+1,j+1) \in \mathcal{C}$ *it holds that* $M[i,j] = 0 = M[i+1,j+1]$.

In other words, a diagonal-restricted path $\mathcal{C}$, is allowed to take diagonal steps in the free space matrix only if both entries are zero. Hence, any one-entry visited by $\mathcal{C}$ must be reached by either a vertical or horizontal step in M and it must be left by a vertical or horizontal step in M. This definition, which might seem overly technical at first sight, will simplify a lot of our arguments. The following observations show that it suffices to consider diagonal-restricted paths instead of all coupling paths in the analysis of a Fréchet tester.

▶ **Observation 8.** *For two curves* P *and* Q *with* n *vertices each, the following holds:*

- **i)** *There exists a diagonal-restricted path of cost zero from* $M_\delta[1,1]$ *to* $M_\delta[n,n]$ *if and only if there exists a coupling path of cost zero from* $M_\delta[1,1]$ *to* $M_\delta[n,n]$.
- **ii)** *If there is no diagonal-restricted path of cost at most* εn *through* M_δ *from* $M_\delta[1,1]$ *to* $M_\delta[n,n]$, *the curves are* $(\frac{\varepsilon}{3},\delta)$*-far. See Figure 3.*
- **iii)** *If* P *and* Q *are* (ε,δ)*-far, there is no diagonal-restricted path of cost at most* εn *through* M_δ *from* $M_\delta[1,1]$ *to* $M_\delta[n,n]$.

Below, we present three structural lemmas that help with the analysis of Algorithm 1. For the proofs of these lemmas, we refer to the full version. The first lemma considers a subpath of a diagonal-restricted path $\mathcal{C}$ that starts and ends in a zero-entry and visits only one-entries in between. We show that the bounding box of the two zero-entries must otherwise be filled with one-entries. Note that this lemma does not hold for general coupling paths of Section 2. Using it will simplify our analysis and is the main motivation for introducing Definition 7.

▶ **Lemma 9** (Box of ones). *Let* $\mathcal{C}$ *be a diagonal-restricted path of minimum cost. Let* $(i,j),(i',j') \in \mathcal{C}$ *be zero-entries such that* $\mathcal{C}$ *visits only one-entries between them. Then, for all tuples* $(k,l) \in \{(k,l) \mid i \le k \le i', j \le l \le j'\} \setminus \{(i,j),(i',j')\}$ *we have that* $M[k,l] = 1$.

We give a definition describing when two columns or rows satisfy the constraints for t-locality.

▶ **Definition 10.** *The columns* i_1 *and* i_2 pass t-locality *if any two zero-entries* (i_1,j_1) *and* (i_2,j_2) *in columns* i_1 *and* i_2 *satisfy* $|j_1 - j_2| \le t \cdot (2 + |i_1 - i_2|)$. *Similarly the rows* j_1 *and* j_2 pass t-locality *if any two zero-entries* (i_1,j_1) *and* (i_2,j_2) *in rows* j_1 *and* j_2 *satisfy* $|i_1 - i_2| \le t \cdot (2 + |j_1 - j_2|)$.

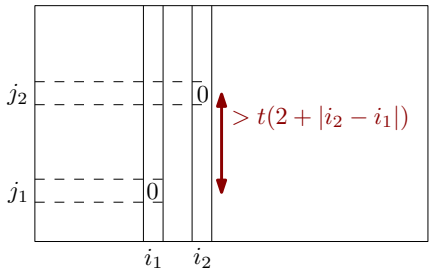


Figure 4 Two columns that do not pass t-locality.

A column (resp. row) can also pass t-locality with itself if $i_1 = i_2$ (resp. $j_1 = j_2$). Intuitively, two columns pass t-locality, when all their zero-entries are not too far away vertically. Note that all columns (resp. rows) pass t-locality with each other if M is t-local.

The second lemma shows that if an optimal diagonal-restricted path contains a long sequence of one-entries, then there must be many barrier-columns and barrier-rows.

▶ **Lemma 11** (Barrier Lemma). *Let $\mathcal{C}$ be a diagonal-restricted path of minimum cost. Suppose that there are two zero-entries $(i,j),(i',j') \in \mathcal{C}$ such that $\mathcal{C}$ visits no zero-entry and at least $4t$ one-entries in between them and suppose that all columns in $[i,i']$ and all rows in $[j,j']$ pass t-locality with each other. Then there is a total of at least $\lceil \frac{i'-i+j'-j}{2t} \rceil$ barrier-rows between j and j' and barrier-columns between i and i'.*

The third lemma shows that if an optimal diagonal-restricted path has a long stretch with relatively many one-entries on the path, there cannot be any long coupling path of cost zero in the same sequence of rows or columns of the matrix M, implying impermeability.

▶ **Lemma 12** (Impermeability Lemma). *Let $\mathcal{C}$ be a diagonal-restricted path through M of minimum cost. Suppose $(i,j),(i',j') \in \mathcal{C}$ with $j' - j > 2t$ (resp. $i' - i > 2t$) correspond to zero-entries in M and the subpath of $\mathcal{C}$ from (i,j) to (i',j') visits at least $4t-1$ one-entries and suppose that any column in $[i,i']$ and any row in $[j,j']$ passes t-locality with itself. Then, the block of columns $[i,i']$ (resp. rows $[j,j']$) is not permeable.*

4.3 Analysis of the algorithm

Algorithm 1 samples a range of intervals in Lines 6-8 which are then tested with a permeability query. For this subroutine we can prove the following lemma. For a proof see the full version.

▶ **Lemma 13.** *Let $\mathcal{I} = \{[i,j] \mid 1 \leq i < j \leq n\}$ be a set of all non-empty intervals from $[n]$ and let $X \subset \mathcal{I}$ be a set of K unknown intervals such that the length of each interval is at most ℓ and each element of $[n]$ is contained in at most c of these intervals for some constant $c \geq 1$. Lines 6-8 of Algorithm 1 select a subset $\mathcal{J} \subset \mathcal{I}$ of intervals with $\sum_{J \in \mathcal{J}} |J| = O(\frac{n \log \ell}{K})$ such that with probability at least $9/10$, at least one of the intervals in X is contained in at least one interval in $\mathcal{J}$.*

The next lemma shows that Algorithm 1 is very likely to return "no" in Line 4 if there are many barrier-columns and barrier-rows. The proof can be found in the full version.

▶ **Lemma 14.** *If the total number of barrier-columns and barrier-rows is more than $\frac{3n}{k}$, then Algorithm 1 returns "no" in Line 4 with probability at least $\frac{9}{10}$.*

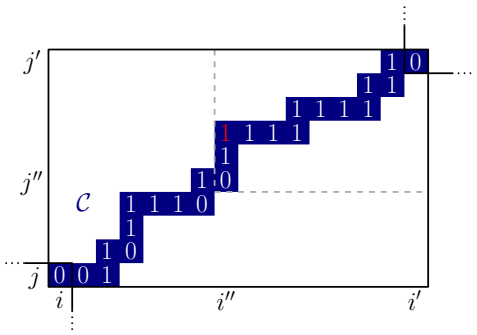


Figure 5 Example for grouping in the proof of Lemma 15 with $t = 2$. The group starts in (i, j). The red one-entry is the $(4t + 1)$-th one-entry in the group. So we stop the group at the next zero-entry (i', j'). Since it contains more than $8t$ one-entries, this is a hollow group. The entries $(i'', j''), (i', j')$ satisfy the conditions in the Barrier (Lemma 11).

▶ **Lemma 15** (Main Lemma). *Let M be t-local and $M[1, 1] = 0 = M[n, n]$. Suppose the total number of barrier-columns and barrier-rows is at most $\frac{\varepsilon n}{8t}$. Let $\mathcal{C}$ be a diagonal-restricted path with lowest cost through M and suppose that $c(\mathcal{C}) > \varepsilon n$. Then, with probability at least $\frac{9}{10}$ at least one sampled interval in the set $\mathcal{J}$ during the execution of Algorithm 1 corresponds to a non-permeable block of columns or rows.*

Proof. We first divide the path $\mathcal{C}$ into a minimum number of groups that each contain more than $4t$ one-entries and start and end with a zero-entry. A visualization of this can be seen in Figure 5. Here, the start of one group is the same entry as the end of the prior group. We do this greedily by following $\mathcal{C}$ until we have visited $4t + 1$ one-entries. Then, we end the group at the next zero-entry after that and repeat the process. Note that the last group may contain fewer than $4t + 1$ one-entries.

We will first show that there are not too many groups that contain more than $8t$ one-entries. Then, we will show that within the remaining groups, there are sufficiently many groups that are not too big. With these groups we then form the set X for the interval sampling in Lemma 13 (i.e., each interval in X will induce a permeability query that fails).

A group that contains more than $8t$ one-entries is called *hollow*, otherwise, it is *dense*.

▷ Claim 16. At most $\frac{\varepsilon n}{2}$ of the one-entries of $\mathcal{C}$ are in hollow groups.

For a proof of the claim, we refer to the full version. So we know that at least $\frac{\varepsilon n}{2}$ of the one-entries of $\mathcal{C}$ are in dense groups. Since a dense group contains at most $8t$ one-entries, we have at least $\lceil \frac{\varepsilon n}{16t} \rceil$ dense groups. Let G be a dense group that starts in (i, j) and ends in (i', j'). We say that $i' - i + j' - j + 2$ is the *size* of G. This corresponds to the sum of the number of columns and rows that it visits. In total, every row and column is counted at most twice by the definition of the groups. So the sum of all group sizes is at most $4n$. We observe that at least half of the groups (which are at least $\lceil \frac{\varepsilon n}{32t} \rceil$ many) have size at most $2 \cdot 4n / \lceil \frac{\varepsilon n}{16t} \rceil \leq \frac{128t}{\varepsilon}$. We call these groups *small*. At last, we create a set X of intervals so that we can apply Lemma 13 to it. Each of the intervals $[i, i']$ in X will have the property that either the block $[i, i']$ of consecutive columns or the block $[i, i']$ of consecutive rows is not permeable.

Let G be a small dense group and suppose it starts in (i, j) and ends in (i', j'). We observe that unless G is the very last group, it visits at least $4t + 1$ one-entries and therefore either visits more than $2t$ rows or more than $2t$ columns. In the first case, we can apply the

Impermeability Lemma to see that the block $[i, i']$ of columns is not permeable. So we add $[i, i']$ to X. In the second case, we can apply the Impermeability Lemma to see that the block $[j, j']$ of rows is not permeable. So we add $[j, j']$ to X. In the end the set X has at least $\lceil \frac{\varepsilon n}{32t} \rceil - 1$ elements and each of them has length at most $\frac{128t}{\varepsilon}$. If we take a closer look at the definitions of the groups, we see that all columns are only contained in one group except for the first and last column of each group. So we can say that every column is contained in at most two groups, i.e. at most two intervals of X. The same holds for rows. So, we have that each element of $[n]$ is contained in at most four of the intervals in X. So we can apply Lemma 13 to X with $\ell = \lceil \frac{128t}{\varepsilon} \rceil$, $K = \lceil \frac{\varepsilon n}{32t} \rceil - 1$, $c = 4$ to see that the algorithm samples one of the intervals from X with probability at least $\frac{9}{10}$. So the permeability query fails for the columns or for the rows with the respective interval, which yields the desired statement. ◀

▶ **Lemma 17.** *Algorithm 1 performs $\mathcal{O}(\frac{t}{\varepsilon} \log \frac{t}{\varepsilon})$ queries.*

Proof. Line 4 performs $\mathcal{O}(\frac{t}{\varepsilon})$ queries. By Lemma 13, Lines 6-8 sample a set $\mathcal{J}$ of intervals that contain a total of $\mathcal{O}(\frac{n \log \ell}{K})$ rows and columns. Each of these columns and rows cause a query in a permeability query. As $\ell = \lceil \frac{128t}{\varepsilon} \rceil$ and $K = \lceil \frac{\varepsilon n}{32t} \rceil - 1$, the claim follows. ◀

▶ **Theorem 18.** *Let $\delta > 0$ and $0 < \varepsilon < 2$ be given. Let P and Q be curves with n vertices such that their free space matrix with value δ is t-local and t is known. Then, Algorithm 1 is a Fréchet-tester that uses $\mathcal{O}(\frac{t}{\varepsilon} \log \frac{t}{\varepsilon})$ queries.*

Proof. If the Fréchet distance of P and Q is at most δ, there is a coupling path of cost 0 through M and Algorithm 1 always returns "yes". So we only need to determine the probability of a false positive. Suppose P and Q are (ε, δ)-far and therefore, a minimum cost diagonal-restricted path visits more than εn one-entries. If there are at least $\frac{\varepsilon n}{8t}$ barrier-columns and barrier-rows, we return "no" with probability at least $\frac{9}{10}$ in Line 4 by Lemma 14. If there are fewer than $\frac{\varepsilon n}{8t}$ barrier-columns and barrier-rows and we get past Line 4, we return "no" with probability at least $\frac{9}{10}$ in Lines 10 and 11 by the Lemma 15. So in any case, the probability for a false positive is at most $\frac{1}{10}$. The number of queries follows from Lemma 17. ◀

5 Locality-oblivious testing of the discrete Fréchet distance

Our main goal in this section is to obtain a Fréchet-tester that does not require an additional input (the value of t). Our approach for this is to first run a "tester" to find an estimate for t and then run Algorithm 1 with that parameter. Unfortunately, this combination is more complicated than just running two testers, for a number of different reasons: First, Algorithm 1 assumes that the matrix is t-local whereas the best we can do is to show that with some probability close to 1, it is "close" to being t-local. This change breaks the proof of correctness of Algorithm 1. The second complication is that we cannot plug in *any* tester for locality, we need a specific one for our analysis that also guarantees some additional properties. Guaranteeing these properties requires more queries than just testing for locality.

5.1 Testing the locality of the Free Space Matrix

To this end, we present a specific error model that is suitable for our purposes. In the following we define how t-locality of a matrix can be violated. First, it could be that two columns or rows do not pass t-locality according to Definition 10. In this case, we say that the respective columns or rows *fail t-locality*. We introduce a second definition that considers a single column and looks for violations of t-locality within that column and within the rows that have a zero-entry in this column.

▶ **Definition 19.** *A single column i (resp. row j)* fails second order t-locality *if either it contains two zero-entries $(i,j),(i,j')$ (resp. $(i,j),(i',j)$) with $|j'-j|>2t$ (resp. $|i'-i|>2t$) or if there exists a zero-entry (i,j) and two zero-entries (i_1,j) and (i_2,j) (resp. (i,j_1) and (i,j_2)) such that $|i_1-i_2|>2t$ (resp. $|j_1-j_2|>2t$). Otherwise, it* passes second order t-locality.

Intuitively, a single column fails second order t-locality if it has a zero-entry in a row that also has two zero-entries more than $2t$ distance apart (similar intuition holds for the rows). See Figure 6. Observe that if a matrix is not t-local then there either exists a pair of columns or rows that fail t-locality or a column or row that fails second order t-locality.

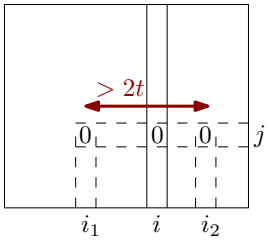


Figure 6 A column i fails second order t-locality.

We now want to define what it means for a matrix to be ζ-close to t-local. This definition is specifically designed to fit the Fréchet-tester in Section 5. Other, maybe more natural definitions of (t,ζ)-close may need different property-testers. However, testing t-locality is not our main focus in this paper.

▶ **Definition 20** (strongly (t,ζ)-local)**.** *We say that a free space matrix M is* strongly (t,ζ)-local *for some $t\geq 1$ and $\zeta\in[0,1]$ if we can partition the set of all columns and rows into two disjoint sets W and I with $|I|=\zeta n$ such that the following properties hold: i) columns $1,n$ and rows $1,n$ are in W, ii) any two columns and any two rows in W pass t-locality and iii) any row or column in W passes second order t-locality. We call the set W the* witness set *and the columns and rows in I* ignored.

If there are $(1-\zeta)n$ rows and $(1-\zeta)n$ columns such that no two of these rows or columns fail t-locality and none of these columns or rows fail second order t-locality, then they serve as a witness set to see that the matrix is $(t,2\zeta)$-local. Because of this, we will check for failures of t-locality and second order t-locality among the columns and rows.

▶ **Observation 21.** *When we query two columns i and i', we can check if they fail t-locality by comparing the lowest zero-entry in column i with the highest zero-entry in column i' and vice versa. Also, after querying a column i, testing for second order t-locality can be done with at most $2t+1$ additional row queries at all zero-entries of the column i.*

Our testing algorithm is quite simple and, in fact, it is an extension of the classic "monotonicity" testing in an array [2]. The algorithm is given in Algorithm 2.

▶ **Lemma 22.** *Given $\sigma>0$ and $t\geq 1$, Algorithm 2 returns "yes" if M is t-local; if M is not strongly $(2t,\sigma)$-local, it returns "no" with probability at least $\frac{9}{10}$. It uses $\mathcal{O}(\frac{t+\log n}{\sigma})$ queries.*

For a detailed proof of Lemma 22 we refer to the full version. The proof relies on standard techniques for monotonicity testing [2]. The Locality-tester is given a value t as an argument. Next, we show that it is also possible to obtain a good estimate of the locality, as well. The

Algorithm 2 Locality-tester(M,σ,t).

1. $T \leftarrow$ binary search tree of height $\lceil \log_2 n \rceil$ on $[n]$ with root $r = \lceil \frac{n}{2} \rceil$.
2. **repeat** $\lceil \frac{3}{\sigma} \rceil + 2$ times:
3. First two iterations: $i \leftarrow 1$ and $i \leftarrow n$. Later: Choose $i \in [2, n-1]$ unif. at random.
4. **If** column i or row i fails second order t-locality **then** return "no".
5. **foreach** j on the r-i-path in T **do**:
6. **If** columns i and j fail t-locality **then** return "no".
7. **If** rows i and j fail t-locality **then** return "no".
8. return "yes".

straightforward approach is to start with a small estimate for t, e.g., $t = 1$, use Algorithm 2 and every time it fails, we can double the t and keep repeating. Our algorithm is very similar to this approach. The difference is merely that we use different ζ values throughout the iterations and we call the locality tester multiple times for the same t.

▶ **Theorem 23.** *Suppose M is t^*-local and we are given $\zeta > 0$. We can design an algorithm that returns a value $t \leq 2t^*$ such that the matrix is strongly $(t, \zeta/t^2)$-local with probability at least $\frac{8}{9}$. The number of queries used is $\mathcal{O}(\frac{\log\log t}{\zeta}(t^3 + t^2 \log n))$.*

Proof. The algorithm operates in rounds. In the i-th round (starting from $i = 1$) we run Locality-tester(M, $\frac{\zeta}{2^{2(i+1)}}, 2^i$) for $2(1 + \log i)$ times. If the locality-tester returns "no" in any of these runs, we go to round $i + 1$, otherwise, we return $t = 2^{i+1}$.

First, we examine the number of queries. Let $i^* = \lceil \log t^* \rceil$. Then, the algorithm terminates before or in round i^* since no two columns or rows fail 2^i-locality for any $2^i \geq t^*$ and all rows and columns pass second order t^*-locality. So we have at most i^* rounds. In round i, we perform $\mathcal{O}((2 + 2\log i) \cdot \frac{2^{2(i+1)}}{\zeta} \cdot (2^i + \log n))$ queries by Lemma 22. The total number of all queries from round 1 to the last round (which is at most i^*) is $\mathcal{O}(\frac{\log\log t}{\zeta}(t^3 + t^2 \log n))$.

By Lemma 22, "no" answers are always correct. However, with probability at most $\frac{1}{10}$, the algorithm can return "yes" even though M is not $(2 \cdot 2^i, \frac{\zeta}{2^{2(i+1)}})$-local. As all the executions of the algorithm are independent, the probability that we get "yes" in all $2(1 + \log i)$ calls of the locality-tester in round i is at most $10^{-(2(1+\log i))} < \frac{1}{100i^2}$. Using a union bound, the probability that we return a value t while M is not $(t, \frac{\zeta}{t^2})$-local is at most $\sum_{i=1}^{\infty} \frac{1}{100i^2} < \frac{1}{9}$. ◀

5.2 The combined algorithm

We now present our algorithm for testing the Fréchet distance without knowing the locality parameter t in advance. Algorithm 3, uses the results from Section 5.1 to first estimate the locality of the matrix M and then uses the Fréchet-tester from Section 4 to decide if the Fréchet distance of the two curves is smaller or equal to δ.

Algorithm 3 Fréchet-tester2(M, ε).

1. $t \leftarrow$ Output of the Algorithm in Theorem 23 for M and $\zeta = \frac{\varepsilon}{1600}$.
2. **return** Fréchet-tester1($M, t, \varepsilon/3, \frac{4800t^2}{\varepsilon}, 2$).

The following lemma is analogous to the main lemma (Lemma 15) from Section 4. For a proof of this lemma we refer to the full version.

▶ **Lemma 24.** *Let $\varepsilon > 0$ and let M be strongly $(t, \frac{\varepsilon}{1600t^2})$-local and $M[1,1] = 0 = M[n,n]$. Suppose the total number of barrier-columns and barrier-rows is at most $\frac{\varepsilon n}{1600t^2}$. Let $\mathcal{C}$ be a diagonal-restricted path with lowest cost through M and suppose that $c(\mathcal{C}) > \varepsilon n$. Then, with probability at least $\frac{9}{10}$ at least one sampled interval in the set $\mathcal{J}$ in the call of Algorithm 1 during Algorithm 3 corresponds to a non-permeable block of columns or rows with probability at least $\frac{9}{10}$.*

▶ **Theorem 25.** *Let $\delta > 0$ and $0 < \varepsilon < 2$ be given. Let P and Q be curves with n vertices such that their free space matrix is t^*-local. Then, Algorithm 3 is a Fréchet-tester that uses $\mathcal{O}(((t^*)^3 + (t^*)^2 \log n)\frac{\log\log t^*}{\varepsilon})$ queries.*

Proof. The Locality-tester uses $\mathcal{O}(\frac{\log\log t^*}{\varepsilon}((t^*)^3 + (t^*)^2 \log n))$ queries. Since $t \leq 2t^*$, we need $\frac{(t^*)^2}{\varepsilon}$ queries to test for barrier-columns and barrier-rows and $\mathcal{O}\left(\frac{t^*}{\varepsilon}\log\left(\frac{t^*}{\varepsilon}\right)\right)$ queries for the permeability queries. The total number of queries follows since $\frac{t^*}{\varepsilon} \in \mathcal{O}(n^2)$.

If the Fréchet distance of P and Q is at most δ, Algorithm 3 will always return "yes". So we have to bound the probability of false positives. Assume that P and Q are (ε, δ)-far and therefore, a minimum cost diagonal-restricted path $\mathcal{C}$ through M has cost higher than εn. Suppose that M is strongly $(t, \frac{\varepsilon}{1600t^2})$-local for the output t of the algorithm from Theorem 23. By the same theorem, this happens with probability at least $\frac{8}{9}$. If the matrix contains more than $\frac{\varepsilon n}{1600t^2}$ barrier-columns and barrier-rows, the subroutine that calls Algorithm 1 returns "no" with probability at least $\frac{9}{10}$ by Lemma 14. If the matrix contains at most $\frac{\varepsilon n}{1600t^2}$ barrier-columns and barrier-rows and the algorithm proceeds until the interval sampling, we are in the setting of Lemma 24 and we return "no" with probability at least $\frac{9}{10}$. Therefore, the probability that the algorithm returns "no" is at least $\frac{8}{9} \cdot \frac{9}{10} = \frac{4}{5}$. So the probability of a false positive is at most $\frac{1}{5}$. ◀

6 Testing the continuous Fréchet distance

This section describes how the Fréchet-testers for the discrete Fréchet distance can be used to serve as $(1+\varepsilon')$-approximate Fréchet-testers for the continuous Fréchet distance. First, we define (ε, δ)-far for the continuous Fréchet-distance. Then, we explain how to use our Fréchet-testers on instances of the continuous Fréchet distance with this error model.

We first define the continuous Fréchet distance. We set F_n to be the set of all continuous non-decreasing functions $f\colon [0,1] \to [1,n]$ with $f(0) = 1$ and $f(1) = n$. Let P and Q be polygonal curves with n and m vertices. The *continuous Fréchet distance* is defined to be

$$\delta_{\mathcal{F}}(P,Q) = \min_{f\in F_n, g\in F_m} \max_{\alpha\in[0,1]} d(P(f(\alpha)), Q(g(\alpha))).$$

Now we want to introduce a possible error model for the continuous Fréchet distance. Given a value $\delta > 0$ and $f \in F_n$, $g \in F_m$, we define

$$\mathcal{P}_{f,g}(P,Q) \coloneqq \int_{d(P(f(t)),Q(g(t)))>\delta} (||(P\circ f)'(t)|| + ||(Q\circ g)'(t)||)dt$$

as the partial difference, namely the total length of the portions of the two curves P and Q that are not matched with distance at most δ by f and g. Building on this, we define the *partial Fréchet difference* to be

$$\mathcal{P}_{\delta}(P,Q) \coloneqq \min_{f\in F_n, g\in F_m} \mathcal{P}_{f,g}(P,Q).$$

This is essentially the same as the partial Fréchet distance as defined in [6].[3]

▶ **Definition 26** ((ε,δ)-far)**.** *Given two polygonal curves P and Q, we say that P and Q are (ε,δ)-far from one another w.r.t. the continuous Fréchet distance if $\mathcal{P}_\delta(P,Q) \geq \varepsilon(\ell(P)+\ell(Q))$.*

▶ **Definition 27** (μ-approximate Fréchet-tester)**.** *Assume we are given query-access to two curves P and Q, and we are given values $\delta > 0$ and $0 < \varepsilon < 1$. If the curves have continuous Fréchet distance at most δ, we must return "yes", if they are $(\varepsilon,\mu\delta)$-far from each other w.r.t. the continuous Fréchet distance, the algorithm must return "no", with probability at least $\frac{4}{5}$.*

In order to use the first Fréchet-tester introduced in this paper, we transform the polygonal curves into discrete curves by subsampling vertices along the edges of the curves. Denote by P_a the discrete curve that arises from P if we subsample vertices along the edges of P such that P visits a curve segment of length a in between any two vertices for some $a \in (0,\ell(P)]$. The last edge might be longer than a but shorter than $2a$. Hence, P_a has $\lfloor \frac{\ell(P)}{a} \rfloor$ edges. This yields that $D_{\mathcal{F}}(P_a,Q_a) - 2a < \delta_{\mathcal{F}}(P,Q) \leq D_{\mathcal{F}}(P_a,Q_a) + 2a$. If P is t-straight, so is P_a for any $a \in (0,\ell(P)]$. The next lemma shows that the notions of (ε,δ)-far are also related.

▶ **Lemma 28.** *Let $0 < a \leq \min\{\ell(P),\ell(Q)\}$. If P and Q are $(12\varepsilon,\delta+2a)$-far from each other, then P_a and Q_a are (ε,δ)-far from each other.*

For a proof see the full version. We adjust our query oracle such that it can access the free space matrix of P_a and Q_a. So we can apply the Fréchet-testers to P_a and Q_a for an adequate choice of a. We show that for given $\varepsilon' > 0$ and $0 < \varepsilon < 2$, we can choose a such that this yields a $(1+\varepsilon')$-approximate Fréchet-tester for the continuous Fréchet distance of P and Q. Let $\varepsilon'' > 0$. For $a = \varepsilon''\delta$, we test $P_{\varepsilon''\delta}$ and $Q_{\varepsilon''\delta}$ for the distance $\delta' := (1+2\varepsilon'')\delta$. We observe that $M_{\delta'}$ is $\mathcal{O}(\frac{t}{\varepsilon''})$-local if P and Q are t-straight. If $\delta_{\mathcal{F}}(P,Q) \leq \delta$, we have $D_{\mathcal{F}}(P_{\varepsilon''\delta},Q_{\varepsilon''\delta}) \leq \delta'$ and if P and Q are $(\varepsilon,\mu\delta)$-far, we have that $P_{\varepsilon''\delta}$ and $Q_{\varepsilon''\delta}$ are $(\frac{\varepsilon}{12},\delta')$-far for $\mu = 1+4\varepsilon'' = 1+\varepsilon'$ for $\varepsilon'' = \varepsilon'/4$. So, if we know t, we can apply the first Fréchet-tester for $\frac{\varepsilon}{12}$ and δ'. If t is unknown, we apply the second Fréchet-tester for the same values.

▶ **Theorem 29.** *Let $\delta,\varepsilon,\varepsilon' > 0$ be given and assume $\varepsilon < 1$. Let P and Q be t-straight curves with $\ell(P) = \ell(Q)$. If t is known, there exists a $(1+\varepsilon')$-approximate Fréchet-tester w.r.t. the continuous Fréchet distance that performs at most $\mathcal{O}\left(\frac{t}{\varepsilon\cdot\varepsilon'}\log\frac{t}{\varepsilon\cdot\varepsilon'}\right)$ queries. If t is unknown, there exists a $(1+\varepsilon')$-approximate Fréchet-tester w.r.t. the continuous Fréchet distance that performs at most $\mathcal{O}\left(\left((\frac{t}{\varepsilon'})^3 + (\frac{t}{\varepsilon'})^2\log\left\lceil\frac{\ell(P)}{\varepsilon'\delta}\right\rceil\right)\frac{\log\log\frac{t}{\varepsilon'}}{\varepsilon}\right)$ queries.*

A natural consequence of the above theorem is that for t-straight curves, the number of queries is bounded in terms of n and the aspect ratio of the input.

References

1 Julian Baldus and Karl Bringmann. A fast implementation of near neighbors queries for Fréchet distance (GIS cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 99:1–99:4. ACM, 2017. doi:10.1145/3139958.3140062.

2 Arnab Bhattacharyya and Yuichi Yoshida. *Property Testing: Problems and Techniques.* Springer Nature, 2022.

[3] Note that, in line with their definition, we assume the reparameterizations to be (piecewise) differentiable.

3 Milutin Brankovic, Kevin Buchin, Koen Klaren, André Nusser, Aleksandr Popov, and Sampson Wong. (k, ℓ)-medians clustering of trajectories using continuous dynamic time warping. In *SIGSPATIAL '20: 28th International Conference on Advances in Geographic Information Systems*, pages 99–110. ACM, 2020. doi:10.1145/3397536.3422245.

4 Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristán, Rodrigo I. Silveira, Frank Staals, and Carola Wenk. Clustering trajectories for map construction. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 14:1–14:10. ACM, 2017. doi:10.1145/3139958.3139964.

5 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Jorren Hendriks, Erfan Hosseini Sereshgi, Vera Sacristán, Rodrigo I. Silveira, Jorrick Sleijster, Frank Staals, and Carola Wenk. Improved map construction using subtrajectory clustering. In *LocalRec'20: Proceedings of the 4th ACM SIGSPATIAL Workshop on Location-Based Recommendations, Geosocial Networks, and Geoadvertising*, pages 5:1–5:4. ACM, 2020. doi:10.1145/3423334.3431451.

6 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 645–654. SIAM, 2009. doi:10.1137/1.9781611973068.71.

7 Kevin Buchin, Yago Diez, Tom van Diggelen, and Wouter Meulemans. Efficient trajectory queries under the Fréchet distance (GIS cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 101:1–101:4. ACM, 2017. doi:10.1145/3139958.3140064.

8 Kevin Buchin, Anne Driemel, Natasja van de L'Isle, and André Nusser. klcluster: Center-based clustering of trajectories. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 496–499. ACM, 2019. doi:10.1145/3347146.3359111.

9 Bernard Chazelle, Ding Liu, and Avner Magen. Sublinear geometric algorithms. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 531–540, 2003. doi:10.1145/780542.780620.

10 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6):1370–1379, 2005. doi:10.1137/S0097539702403244.

11 Artur Czumaj, Funda Ergün, Lance Fortnow, Avner Magen, Ilan Newman, Ronitt Rubinfeld, and Christian Sohler. Approximating the weight of the Euclidean minimum spanning tree in sublinear time. *SIAM Journal on Computing*, 35(1):91–109, 2005. doi:10.1137/S0097539703435297.

12 Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(3):904–922, 2009. doi:10.1137/060672121.

13 Artur Czumaj, Christian Sohler, and Martin Ziegler. Property testing in computational geometry. In *European Symposium on Algorithms*, pages 155–166. Springer, 2000. doi:10.1007/3-540-45253-2_15.

14 Fabian Dütsch and Jan Vahrenhold. A filter-and-refinement-algorithm for range queries based on the Fréchet distance (GIS cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 100:1–100:4. ACM, 2017. doi:10.1145/3139958.3140063.

15 Rogers Epstein and Sandeep Silwal. Property testing of lp-type problems. In *47th International Colloquium on Automata, Languages, and Programming*, volume 168 of *LIPIcs*, pages 98:1–98:18, 2020. doi:10.4230/LIPICS.ICALP.2020.98.

16 Eldar Fischer and Ilan Newman. Testing of matrix properties. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 286–295, 2001. doi:10.1145/380752.380812.

17 Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.

18 J.E. Goodman, J. O'Rourke, and C.D. Tóth. *Handbook of discrete and computational geometry, third edition*. CRC Press LLC, January 2017. doi:10.1201/9781315119601.

19 Sariel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, USA, 2011.

20 Jirí Matousek. *Lectures on discrete geometry*, volume 212 of *Graduate texts in mathematics*. Springer, 2002.

21 Morteza Monemizadeh. Facility Location in the Sublinear Geometric Model. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX-/RANDOM 2023)*, volume 275 of *LIPIcs*, pages 6:1–6:24, 2023. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.6.

22 Ketan. Mulmuley. *Computational geometry : an introduction through randomized algorithms*. Prentice-Hall, Englewood Cliffs, N.J, 1994.

23 Dana Ron. Property testing. *Combinatorial Optimization – Dordrecht*, 9(2):597–643, 2001.

24 Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends® in Theoretical Computer Science*, 5(2):73–205, 2010. doi:10.1561/0400000029.

A Simple yet Exact Analysis of the MultiQueue

Stefan Walzer
Karlsruhe Institute of Technology, Germany

Marvin Williams
Karlsruhe Institute of Technology, Germany

Abstract

The MultiQueue is a relaxed concurrent priority queue consisting of n internal priority queues, where an insertion uses a random queue and a deletion considers two random queues and deletes the minimum from the one with the smaller minimum. The *rank error* of the deletion is the number of smaller elements in the MultiQueue.

Alistarh et al. [2] have demonstrated in a sophisticated potential argument that the expected rank error remains bounded by $\mathcal{O}(n)$ over long sequences of deletions.

In this paper we present a simpler analysis by identifying the stable distribution of an underlying Markov chain and with it the long-term distribution of the rank error exactly. Simple calculations then reveal the expected long-term rank error to be $\frac{5}{6}n - 1 + \frac{1}{6n}$. Our arguments generalize to deletion schemes where the probability to delete from a given queue depends only on the rank of the queue. Specifically, this includes deleting from the best of c randomly selected queues for any $c > 1$.

2012 ACM Subject Classification Mathematics of computing → Stochastic processes; Theory of computation → Data structures design and analysis

Keywords and phrases MultiQueue, concurrent data structure, stochastic process, Markov chain

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.85

Related Version *Full Version*: `https://arxiv.org/abs/2410.08714` [20]

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882 500).



1 Introduction

Priority queues maintain a set of elements from a totally ordered domain, with an *insert* operation adding an element and a *deleteMin* operation extracting the smallest element. They are a fundamental building block for a wide range of applications such as task scheduling, graph algorithms, and discrete event simulation. The parallel nature of modern computing hardware motivates the design of concurrent priority queues that allow multiple processing elements to insert and delete elements concurrently. *Strict*[1] concurrent priority queues (e.g., [19, 11, 7, 17]) suffer from poor scalability due to inherent contention on the smallest element [4, 8]. To alleviate this contention, in *relaxed* priority queues deletions should still preferentially extract small elements but need not always extract the minimum. In other words, the correctness requirement is turned into a quality measure: We speak of a *rank error* of $r - 1$ if the extracted element has rank r among all elements currently in the priority queue. In many scenarios, relaxed priority queues outperform strict priority queues, as the higher scalability outweighs the additional work caused by the relaxation. They are an active field of research and a vast range of designs has been proposed [10, 16, 22, 3, 18, 21, 15, 23].

[1] in the sense of *linearizability*



© Stefan Walzer and Marvin Williams;
licensed under Creative Commons License CC-BY 4.0



The MultiQueue. The MultiQueue, initially proposed by Rihani et al. [16] and improved upon by Williams et al. [21], emerged as the state-of-the art relaxed priority queue. Due to its high scalability and robust quality, the MultiQueue inspired a number of follow-up works [15, 23]. Its design uses the *power-of-two-choices* paradigm and is delightfully simple: We use n (sequential) priority queues for some fixed $n \in \mathbb{N}$. Each insertion adds its element to a queue chosen uniformly at random and each deletion picks *two* queues uniformly at random and deletes from the one with the smaller minimum. Figure 1 illustrates the MultiQueue

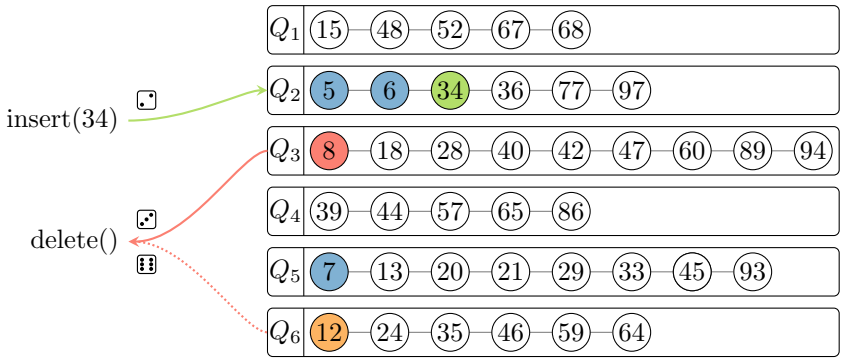


Figure 1 The MultiQueue with $n = 6$ queues and some elements already inserted. The shown insertion picks queue 2 to insert the element 34 (green). The shown deletion picks queues 3 and 6 with minima 8 (red) and 12 (orange), and deletes the 8 since it is smaller. The deletion exhibits a rank error of 3 due to the 3 smaller elements highlighted in blue.

with $n = 6$ queues. We can generalize this design to pick any number $c > 1$ of queues for deletions where even non-integer choices of c make sense: We would then pick $\lfloor c \rfloor + 1$ queues with probability $c - \lfloor c \rfloor$ and $\lfloor c \rfloor$ queues otherwise. In practice, the individual queues are protected by mutual exclusion locks, and n is proportional to the number of processing elements to find unlocked queues in (expected) constant time.

Existing theory on the MultiQueue. Our theoretical understanding of the MultiQueue is still incomplete. One obstacle is that the order of operations matters: Intuitively, deletions can cause the distribution of elements to drift apart and increase the expected rank error, while insertions of small elements can mask accumulated differences. This suggests that the worst-case setting is when following the insertion of sufficiently many elements, only deletions occur. Like Alistarh et al. [2], we exclusively consider this setting. At first glance, this process seems closely related to the classical balls-into-bins process, where balls are placed one after the other into the least loaded of two randomly chosen bins. Famously, the difference between the highest load and the average load for n bins is in $\mathcal{O}(\log \log n)$ with high probability for any number of balls. Numerous variants of the balls-into-bins process have been proposed and studied [5, 12, 6, 14]. However, reducing the process to a balls-into-bins process imposes multiple difficulties. The state of the MultiQueue is not fully described by the number of elements in each queue but also involves information about the ranks of the elements. Moreover, deleting an element from a queue can affect the ranks of elements in other queues. Despite these challenges, Alistarh et al. [2] managed to transfer the potential argument from the balls-into-bins analysis by Peres et al. [14] to a MultiQueue analysis via an intermediate "exponential process" that avoids correlations between the elements in the queues. They prove that the expected rank error is in $\mathcal{O}(n)$ and the expected worst-case rank error is in $\mathcal{O}(n \log n)$ for any number of deletions and any $c \in (1, 2]$, while the rank errors diverge

with the number of deletions for $c = 1$. In follow-up work, this technique is generalized to other relaxed concurrent data structures [1] and a process where each processing element has its own priority queue and "steals" elements from other processing elements with some probability [15].

Contribution. In this paper, we present an analysis of the MultiQueue that, compared to the potential argument by Alistarh et al. [2], is simultaneously simpler and more precise. We characterise the exact long-term distribution of the rank error for any $c > 1$ and any $n \in \mathbb{N}$. From this distribution we can derive, for instance, that the expected rank error for $c = 2$ is $\frac{5}{6}n - 1 + \frac{1}{6n}$. In addition to covering any $c > 1$, our analysis generalizes to all deletion schemes where the probability for a queue to be selected solely depends on the rank of the queue (when ordering the queues according to their smallest element).

We achieve this by modelling the deletion process as a Markov chain and observing that its stationary distribution can be described using a sequence of $n - 1$ independent geometrically distributed random variables. In the full version of this paper [20] we also apply our techniques to the elegant exponential process by Alistarh et al. [2], which we believe to be of independent interest.

Outline. First, in Section 2, we introduce the formal model of a generalized MultiQueue and the setting in which we analyze it. We then state our main results, including Theorem 1 that characterizes the long-term distribution of the ranks of the top-elements in the MultiQueue. In Section 3, we present our analysis of the generalized MultiQueue, leading to the proof of Theorem 1. We apply Theorem 1 to compute the expected rank error for any $c > 1$ and any $n \in \mathbb{N}$ in Section 4. Finally, in Section 5, we discuss implications and limitations of our results and outline possible future work.

2 Formal Model and Results

Analogous to Alistarh et al. [2], we analyze the MultiQueue in a simplified setting where only deletions occur.

The σ-MultiQueue. A σ-MultiQueue consists of n priority queues and a choice distribution σ on $[n]$ that captures a generalised deletion strategy (as explained below). The n queues are initially populated by randomly partitioning an infinite set $\{x_1 < x_2 < x_3 < \ldots\}$ of elements. [2] The queues, identified with the sets of elements they contain, are denoted by $Q_1, \ldots, Q_n$, indexed such that $\min Q_1 < \min Q_2 < \ldots < \min Q_n$. The minima are also called *top-elements*. We then perform a sequence of deletions. Each time, we select an index $i \in [n]$ according to σ and delete the top-element of Q_i. [3] Then, we relabel the queues such that their top-elements appear in ascending order again. Let $(Q_1^{(s)}, \ldots, Q_n^{(s)})$ be the sequence of queues after s deletions and $r_i^{(s)}$ the rank of the top-element $\min Q_i^{(s)}$ among $Q_1^{(s)} \cup \ldots \cup Q_n^{(s)}$, for any $s \in \mathbb{N}$ and $i \in [n]$.

Intuitively, σ should be biased towards smaller values of i, i.e., towards selecting queues with smaller top-elements, to ensure that the rank error does not diverge over time. Using the notation $\sigma_i := \Pr_{i^* \sim \sigma}[i^* = i]$ and $\sigma_{\leq i} := \Pr_{i^* \sim \sigma}[i^* \leq i]$, the formal requirement for σ turns out as:

[2] Note that every queue receives an infinite number of elements with probability 1.
[3] We write $[n]$ as a shorthand for $\{1, \ldots, n\}$.

$$\forall i \in [n-1] : \sigma_{\leq i} > \tfrac{i}{n} \qquad (\star)$$

If σ does not satisfy $(\star)$, the MultiQueue deletes too frequently from "bad" queues (i.e., with large top-elements) and the gap between "good" and "bad" queues increases over time. We prove a corresponding formal claim in the full version of this paper [20].

Surprisingly independent random variables. One might think that the ranks $r_1^{(s)} < \ldots < r_n^{(s)}$ are correlated in complex ways. While they *are* correlated, they effectively arise as the prefix sum of $n-1$ *independent* random variables. More precisely, our main theorem draws attention to the differences between the ranks of consecutive top-elements.

▶ **Theorem 1** (Proof in Section 3). *Let $(r_1^{(s)}, \ldots, r_n^{(s)})$ denote the ranks of the top-elements of a σ-MultiQueue with σ satisfying $(\star)$ after s deletions. Then $(r_1^{(s)}, \ldots, r_n^{(s)})$ converges in distribution to a sequence $(r_1, \ldots, r_n)$ of random variables where*

$$r_i = 1 + \sum_{j=1}^{i-1} \delta_j \text{ for } i \in [n] \text{ with } \delta_i \sim \mathrm{Geom}_1\Big(1 - \frac{i}{n\sigma_{\leq i}}\Big) \text{ for } i \in [n-1],$$

where $\mathrm{Geom}_1(p)$ denotes the geometric distribution of the number of Bernoulli trials with success probability p until (and including) the first success.

From the distribution of the ranks of the top-elements given by Theorem 1 it is straightforward to derive the rank error distribution.

▶ **Corollary 2** (Proof in Section 3). *Let $R^{(s)}$ denote the rank error exhibited by the deletion in step s in the σ-MultiQueue with σ satisfying $(\star)$. Then $R^{(s)}$ converges in distribution to the random variable R where*

$$R = r_{i^*} - 1 \text{ with } i^* \sim \sigma \text{ and } r_i \text{ as in Theorem 1.}$$

The expected rank error is (in the long run)

$$\mathbb{E}[R] = \sum_{i=1}^{n-1} \frac{\sigma_{\leq i}(1-\sigma_{\leq i})}{\sigma_{\leq i} - \frac{i}{n}}.$$

Application to the c-MultiQueue. For $c > 1$ we define the c-MultiQueue to be the σ-MultiQueue where σ is the distribution that corresponds to picking the best out of c randomly selected queues as described in Section 1. For instance, the probability to select one of the first i queues with $c = 2$ is $\sigma_{\leq i} = 1 - (1 - \frac{i}{n})^2$. Note that $(\star)$ is satisfied (see Lemma 8). We can then derive several useful quantities from Theorem 1, including the expected rank error.

▶ **Theorem 3.** *Consider the expectation $\mathbb{E}[R]$ of the rank error $R = r_{i^*} - 1$ of the c-MultiQueue in the long run (i.e., after convergence).*

(i) *For $c = 2$ we have $\mathbb{E}[R] = \frac{5}{6}n - 1 + \frac{1}{6n} = \frac{5}{6}n - \Theta(1)$.*

(ii) *For $c = 1 + \varepsilon$ with $\varepsilon \in (0,1)$, we have $\mathbb{E}[R] = (\frac{1}{\varepsilon} - \frac{\varepsilon}{6})n - \frac{1}{\varepsilon} + \frac{\varepsilon}{6n} = (\frac{1}{\varepsilon} - \frac{\varepsilon}{6})n - \mathcal{O}(1/\varepsilon)$.*

(iii) *For any $c > 1$ we have $n \cdot \int_0^1 f_c(x)\,dx - \frac{c}{c-1} \leq \mathbb{E}[R] \leq n \cdot \int_0^1 f_c(x)\,dx$, where $f_c : (0,1) \to \mathbb{R}$ is defined as follows using $\varepsilon := c - \lfloor c \rfloor \in [0,1)$*

$$f_c(x) = x^{\lfloor c \rfloor - 1}(1 - \varepsilon + \varepsilon x) \cdot \frac{1 - x^{\lfloor c \rfloor}(1 - \varepsilon + \varepsilon x)}{1 - x^{\lfloor c \rfloor - 1}(1 - \varepsilon + \varepsilon x)}.$$

(iv) *Cruder but simpler bounds for any $c \geq 2$ are: $\frac{n}{\lceil c \rceil} - \frac{c}{c-1} \leq \mathbb{E}[R] \leq \frac{n}{\lfloor c \rfloor - 1}$.*

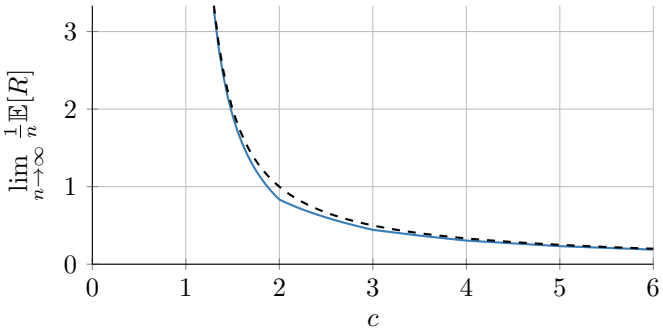


Figure 2 For large n, the c-MultiQueue has an expected rank error of $\mathbb{E}[R] = n \cdot \int_0^1 f_c(x)\,dx + o(n)$ where $f_c(x)$ is defined in Theorem 3. The solid line shows $c \mapsto \int_0^1 f_c(x)\,dx$ and hence the constant in front of the leading term. The dashed line $c \mapsto \frac{1}{c-1}$ is an asymptote for both $c \to 1$ and $c \to \infty$.

Figure 2 plots the expected asymptotic rank error per queue depending on c, and an approximation $\frac{1}{c-1}$. We also give concentration bounds for the rank error in Theorem 9.

The Exponential-Jump Process. As an intermediate step in their analysis, Alistarh et al. [2] introduce the "exponential process", where new top-elements are not given by the current state but generated by adding an exponential random variable to the current top-element. We reformulate this process as the equivalent *exponential-jump process* (EJP) as follows. The EJP involves n tokens on the real number line and a distribution σ on $[n]$. In every step, we sample $i^* \sim \sigma$ and $X \sim \mathrm{Exp}(1)$. We then identify the i^*th token from the left and move it a distance of X to the right. More formally, the state of the process is given by the sequence $t_1 < \ldots < t_n$ of positions of the tokens and the state transition can be described as

$$(t_1, \ldots, t_n) \rightsquigarrow \mathrm{sort}(t_1, \ldots, t_{i^*} + X, \ldots, t_n) \text{ where } i^* \sim \sigma \text{ and } X \sim \mathrm{Exp}(1). \tag{EJP}$$

We provide an exact analysis, which we believe to be of independent interest, and explain the connection between the EJP and the MultiQueue in the full version of this paper [20]. With the same methods as before, we analyse the differences $(d_1, \ldots, d_{n-1})$ with $d_i = t_{i+1} - t_i$.

▶ **Theorem 4.** *Let $n \in \mathbb{N}$ and let σ be a distribution on $[n]$ that satisfies $(\star)$. The EJP admits a stationary distribution π for the differences $(d_1, \ldots, d_{n-1})$ with*

$$\pi = \bigtimes_{i=1}^{n-1} \mathrm{Exp}(n \cdot \sigma_{\le i} - i).$$

In other words, the distances between neighbouring tokens are, in the long-run, mutually independent and exponentially distributed with parameters as given.

3 A Direct Analysis of the σ-MultiQueue

When analysing random processes, it is often a good idea to reveal information only when needed, keeping the rest hidden behind a veil of probability. In our case, the idea is to conceal the queue an element is in until the element becomes a top-element.

We discuss this idea using the example in Figure 3 where $n = 4$ queues are initially populated with the set $\mathbb{N}$.

In (a) we see an explicit representation of a possible state, where queues are labeled in increasing order of their top-elements. We can tell, for instance, that when removing the top-element 7 of queue 2 then the new top-element would be 9. In (b) we keep track of the

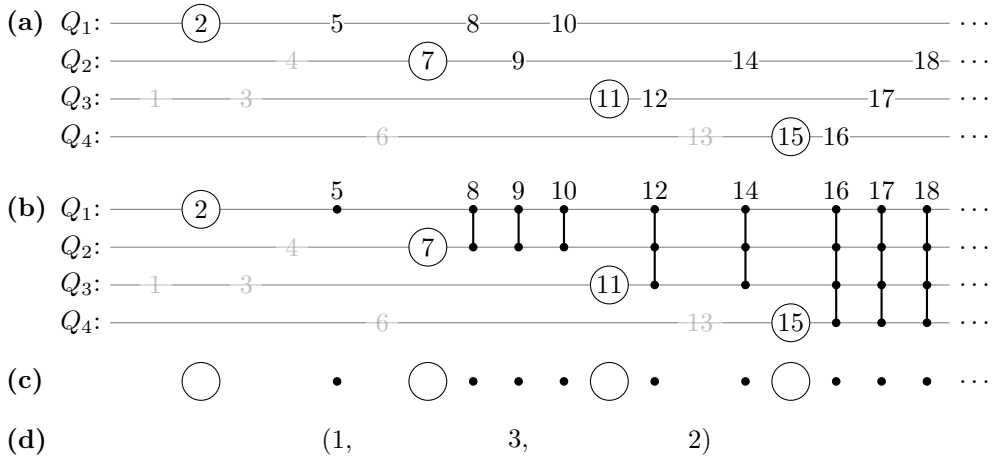


Figure 3 Increasingly abstract ways for modeling the state of a MultiQueue system with 4 queues. **(a)** contains all information. **(b)** assumes that the queues in which the elements reside has only been partially revealed. **(c)** abstracts away from concrete elements. **(d)** represents the information in (c) using numbers.

current and past top-elements of all queues but do not reveal ahead of time which queue each element of $\mathbb{N}$ is assigned to. As far as we know, the elements $16, 17, 18, \ldots$ are assigned to each of the four queues with equal probability. It is unavoidable that we obtain partial information, however: If an element is smaller than the top-element of some queue, it cannot possibly be contained in that queue. The elements 8, 9 and 10 are in queue 1 and 2 with probability 1/2 each, and the elements 12 and 14 are in queues 1, 2 and 3 with probability 1/3 each. Element 5 is surely contained in queue 1, but we can treat this as a degenerate probability distribution rather than as a special case. Note what happens when element 7 is deleted: First, 8 has a chance of 1/2 of being the new top-element of queue 2. If it turns out that 8 is not the new top-element, then 9 gets the same chance, then 10. If all three elements are rejected, then element 12 is considered, getting a chance of 1/3 (because it could still be in three queues) and so on.

Since we are only interested in the ranks of top-elements over time, we can forget the removed elements and the concrete element values and arrive at representation (c), showing n *balls* "◯" representing top-elements and *dots* "•" representing other elements. Equivalently, in (d) we list the sequence of dot-counts in between the balls, omitting the infinite number of dots to the right of the last ball.

We now represent the σ-MultiQueue as a Markov chain with states in $\mathbb{N}_0^{n-1}$ as in (d), borrowing language from (b) and (c) when useful. Since the state space is countably infinite, the role of the transition matrix is filled by an infinite family P of transition probabilities where $P(\vec{d}_{\text{start}}, \vec{d}_{\text{end}})$ denotes the probability to transition from state $\vec{d}_{\text{start}} \in \mathbb{N}_0^{n-1}$ to state $\vec{d}_{\text{end}} \in \mathbb{N}_0^{n-1}$. These probabilities are implicitly described below. We write $\vec{d}$ for a state $(d_1, \ldots, d_{n-1}) \in \mathbb{N}_0^{n-1}$ and $\vec{e}_i = 0^{i-1}10^{n-i-1} \in \{0,1\}^{n-1}$ denotes the ith unit vector for $i \in [n-1]$. We avoid special cases related to the last ball by defining $d_n = \infty$ and $\vec{e}_n = \vec{0}$.

A state $\vec{d}_{\text{start}}$ transitions to another state $\vec{d}_{\text{end}}$ via a sequence of *transitional states* $(\vec{d}, i) \in \mathbb{N}_0^{n-1} \times [n]$ in which one ball $i \in [n]$ (numbered from left to right) is marked as the *active ball*.

1. Given state $\vec{d}_{\text{start}}$, we sample $i^* \sim \sigma$ and obtain the transitional state $(\vec{d}_{\text{start}}, i^*)$.
   Interpretation: In terms of (c) we activate ball i^* and in terms of (b) we delete the top-element from queue Q_{i^*} and look for a new one.

2. As long as we are in a transitional state $(\vec{d}, i)$, there are two cases:

2.1 If $d_i > 0$, then with probability $(i-1)/i$ we continue with transitional state $(\vec{d} - \vec{e}_i + \vec{e}_{i-1}, i)$ and with probability $1/i$ the transition ends with state $\vec{d}_{\text{end}} = \vec{d} - \vec{e}_i$.
Interpretation: In terms of (c) the active ball decides to skip past the dot to the right of it, or consumes the dot and stops. In terms of (b), we reveal whether the next top-element candidate for Q_i is contained in Q_i; if it is, the candidate becomes the new top-element and we stop, otherwise we continue the search.

2.2 If $d_i = 0$, then we continue with transitional state $(\vec{d}, i+1)$.
Interpretation: In terms of (c) the active ball overtakes another ball, thereby becoming ball $i+1$. In terms of (b), we update the ordering of the queues since the new top-element of Q_i is now known to be larger than the top-element of queue Q_{i+1}.

For clarity, we illustrate the possible transitions for a concrete state $\vec{d}_{\text{start}} \in \mathbb{N}_0^{n-1}$ and resulting transition probabilities in Figure 4. We remark that the following analysis does not refer to the transition probabilities directly, but uses their implicit characterisation.

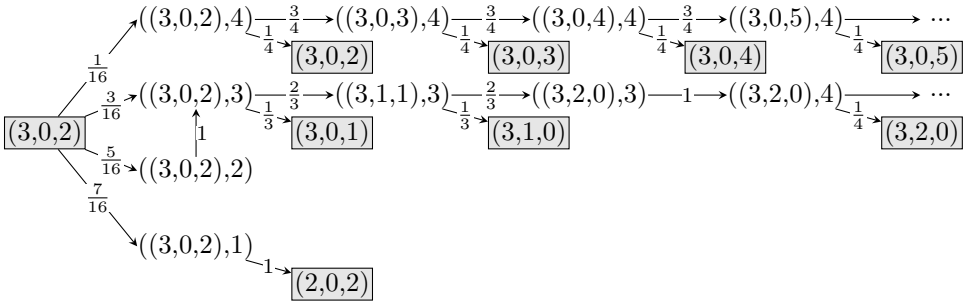


Figure 4 Possible states and transitional states reachable from $\vec{d}_{\text{start}} = (3,0,2)$ in a single state transition. The probabilities $\frac{1}{16}, \frac{3}{16}, \frac{5}{16}, \frac{7}{16}$ on the outgoing edges of $(3,0,2)$ assume the special case of the 2-MultiQueue. Taking the example $\vec{d}_{\text{end}} = (3,1,0)$, the probability to transition from $\vec{d}_{\text{start}}$ to $\vec{d}_{\text{end}}$ arises from the two corresponding paths as $P(\vec{d}_{\text{start}}, \vec{d}_{\text{end}}) = \frac{3}{16} \cdot \frac{2}{3} \cdot \frac{1}{3} + \frac{5}{16} \cdot 1 \cdot \frac{2}{3} \cdot \frac{1}{3}$.

We now state the main result of this section, which characterizes a stationary distribution of P. With this lemma, we can finally prove Theorem 1 and Corollary 2.

▶ **Theorem 5.** *Let $n \in \mathbb{N}$ and let σ be a distribution on $[n]$ that satisfies $(\star)$. The transition probabilities P admits the stationary distribution π given by*

$$\pi = \bigtimes_{i=1}^{n-1} \text{Geom}\Big(1 - \frac{i}{n \cdot \sigma_{\le i}}\Big),$$

where $\text{Geom}(p)$ denotes the geometric distribution of the number of failed Bernoulli trials with success probability p before the first success, and $\bigtimes$ denotes the direct product of distributions.

In particular, the $n-1$ components of $\vec{d} \sim \pi$ are *independent* random variables. We further make the following useful observation.

▶ **Observation 6.** *For any $\vec{d} \in \mathbb{N}_0^{n-1}$ and any $i \in [n]$ we have $\pi(\vec{d} + \vec{e}_i) = \pi(\vec{d}) \cdot \frac{i}{n \cdot \sigma_{\le i}}$.*

Proof of Observation 6. If $i = n$ then $\vec{e}_i = \vec{0}$ and $\sigma_{\le i} = 1$ so the claim is trivial. Now assume $i < n$. For any $\vec{d} \in \mathbb{N}_0^{n-1}$ we have

$$\pi(\vec{d}) = \prod_{i=1}^{n-1} (1-p_i)^{d_i} p_i \text{ where } p_i = 1 - \frac{i}{n \cdot \sigma_{\le i}}.$$

Hence, $\pi(\vec{d}+\vec{e}_i) = \pi(\vec{d}) \cdot (1-p_i)$ and the claim follows. ◀

The following auxiliary lemma captures the main insight required in the proof of Theorem 5.

▶ **Lemma 7.** *Let $\nu(\vec{d}, i) \in \mathbb{R}_{\geq 0}$ be the probability that a transitional state $(\vec{d}, i) \in \mathbb{N}_0^{n-1} \times [n]$ occurs when transitioning from state $\vec{d}_{\text{start}} \sim \pi$ according to P with σ satisfying $(\star)$. Then,*

$$\nu(\vec{d}, i) = \pi(\vec{d}) \cdot \sigma_{\leq i}.$$

Proof of Lemma 7. We prove $\nu(\vec{d}, i) = \pi(\vec{d}) \cdot \sigma_{\leq i}$ by induction. In the induction step we may assume that $\nu(\vec{d}', i') = \pi(\vec{d}') \cdot \sigma_{\leq i'}$ holds whenever $i' < i$ or when $i = i'$ and $d'_1 + \ldots + d'_{i-1} < d_1 + \ldots + d_{i-1}$. In general, there are three ways in which a transitional state $(\vec{d}, i)$ might be reached:

- **(i)** The transition started with $\vec{d} = \vec{d}_{\text{start}}$ and ball i was activated.
- **(ii)** Ball i has skipped a dot and we thus reached $(\vec{d}, i)$ from $(\vec{d} - \vec{e}_{i-1} + \vec{e}_i, i)$.
- **(iii)** Ball i has just overtaken another ball and we thus reached $(\vec{d}, i)$ from $(\vec{d}, i-1)$.

For $i = 1$, consider a transitional state $(\vec{d}, 1)$ where the first ball is active. Here, only (i) is possible since the first ball never skips a dot (transition ends with probability 1 in rule 2.1) and there is no ball to its left. The probability for $(\vec{d}, 1)$ to occur is thus

$$\nu(\vec{d}, 1) = \pi(\vec{d}) \cdot \sigma_1 = \pi(\vec{d}) \cdot \sigma_{\leq 1} \qquad \text{(recall that } \sigma_i := \Pr_{i^* \sim \sigma}[i^* = i] \text{ and } \sigma_{\leq i} := \Pr_{i^* \sim \sigma}[i^* \leq i]\text{)}.$$

For $i > 1$, there are two cases depending on whether $d_{i-1} > 0$, i.e., whether there is a dot in between ball $i-1$ and ball i. If $d_{i-1} > 0$, then only (i) and (ii) are possible, so

$$\begin{aligned}
\nu(\vec{d}, i) &= \pi(\vec{d}) \cdot \sigma_i + \nu(\vec{d} - \vec{e}_{i-1} + \vec{e}_i, i) \cdot (1 - \tfrac{1}{i}) \\
&= \pi(\vec{d}) \cdot \sigma_i + \pi(\vec{d} - \vec{e}_{i-1} + \vec{e}_i) \cdot \sigma_{\leq i} \cdot (1 - \tfrac{1}{i}) && \text{(Induction)} \\
&= \pi(\vec{d}) \cdot \sigma_i + \pi(\vec{d}) \cdot \frac{n \cdot \sigma_{\leq i-1}}{i-1} \cdot \frac{i}{n \cdot \sigma_{\leq i}} \cdot \sigma_{\leq i} \cdot (1 - \tfrac{1}{i}) && \text{(Observation 6)} \\
&= \pi(\vec{d})(\sigma_i + \sigma_{\leq i-1}) = \pi(\vec{d}) \cdot \sigma_{\leq i}.
\end{aligned}$$

If $d_{i-1} = 0$, then only (i) and (iii) are possible, so

$$\nu(\vec{d}, i) = \pi(\vec{d}) \cdot \sigma_i + \nu(\vec{d}, i-1) \overset{\text{ind.}}{=} \pi(\vec{d}) \cdot \sigma_i + \pi(\vec{d}) \cdot \sigma_{\leq i-1} = \pi(\vec{d}) \cdot \sigma_{\leq i}.$$ ◀

With Lemma 7 in place, we can now prove Theorem 5.

Proof of Theorem 5. Given a state $\vec{d}_{\text{start}} \sim \pi$, we transition to a new state $\vec{d}_{\text{end}}$ according to the transition probabilities P. To end in $\vec{d}_{\text{end}}$, we first need to reach the transitional state $(\vec{d}_{\text{end}} + \vec{e}_i, i)$ for some $i \in [n]$ and then decide to end the transition there. Note that we need $(\vec{d}_{\text{end}} + \vec{e}_i, i)$ rather than $(\vec{d}_{\text{end}}, i)$, since ending the transition (rule 2.1) reduces d_i by one. The probability to end in $\vec{d}_{\text{end}}$ is therefore

$$\begin{aligned}
\Pr[\vec{d}_{\text{end}} = \vec{d}] &= \sum_{i=1}^{n} \nu(\vec{d} + \vec{e}_i, i) \cdot \tfrac{1}{i} \overset{\text{Lem. 7}}{=} \sum_{i=1}^{n} \pi(\vec{d} + \vec{e}_i) \cdot \sigma_{\leq i} \cdot \tfrac{1}{i} \\
&\overset{\text{Obs. 6}}{=} \sum_{i=1}^{n} \pi(\vec{d}) \cdot \frac{i}{n \cdot \sigma_{\leq i}} \cdot \sigma_{\leq i} \cdot \tfrac{1}{i} = \pi(\vec{d}) \sum_{i=1}^{n} \tfrac{1}{n} = \pi(\vec{d}).
\end{aligned}$$

It follows that $\vec{d}_{\text{end}}$ is again distributed according to π and π is a stationary distribution. ◀

Finally, we prove Theorem 1 and Corollary 2.

Proof of Theorem 1. Let $\vec{d}^{(s)} = (d_1^{(s)}, \dots, d_{n-1}^{(s)})$ with $d_i^{(s)} = r_{i+1}^{(s)} - r_i^{(s)} - 1$. Then, $(\vec{d}^{(s)})_{s\in\mathbb{N}}$ is a Markov chain with transition probabilities P. Since we can reach $(0, \dots, 0)$ from any state and vice versa, the Markov chain is irreducible. The Markov chain is aperiodic since $(0, \dots, 0)$ can transition into itself (if ball n is activated and immediately stops). This implies that the stationary distribution π that we found is unique and that $\vec{d}^{(s)}$ converges in distribution to $d \sim \pi$ (see [13, Theorem 1.8.3]). Let $\vec{\delta}^{(s)} = (\delta_1^{(s)}, \dots, \delta_{n-1}^{(s)})$ with $\delta_i^{(s)} = d_i^{(s)} + 1$. Clearly, $(\vec{\delta}^{(s)})_{s\in\mathbb{N}}$ converges in the same way, except that the geometric random variables are shifted. By definition, we have $r_i^{(s)} = 1 + \sum_{j=1}^{i-1} \delta_j^{(s)}$ so the claimed distributional limit $(r_1, \dots, r_n)$ of $(r_1^{(s)}, \dots, r_n^{(s)})$ follows. ◀

Proof of Corollary 2. Theorem 1 states the ranks of the top-elements converge in distribution to $(r_1, \dots, r_n)$. The distribution for R follows from the fact that we select the queue to delete from according to σ and deleting an element with rank r yields a rank error of $r - 1$. Using the fact that $\mathbb{E}_{X\sim\mathrm{Geom}_1(p)}[X] = 1/p$, we have for the expected rank error

$$\begin{aligned}\mathbb{E}[R] = \mathbb{E}\left[r_{i^*} - 1\right] &= \sum_{i=1}^{n} \sigma_i \sum_{j=1}^{i-1} \mathbb{E}[\delta_j] = \sum_{j=1}^{n-1} \mathbb{E}[\delta_j] \sum_{i=j+1}^{n} \sigma_i = \sum_{j=1}^{n-1} \mathbb{E}[\delta_j] \cdot (1 - \sigma_{\le j}) \\ &= \sum_{j=1}^{n-1} \frac{n\sigma_{\le j}}{n\sigma_{\le j} - j} \cdot (1 - \sigma_{\le j}) = \sum_{j=1}^{n-1} \frac{\sigma_{\le j}(1 - \sigma_{\le j})}{\sigma_{\le j} - \frac{j}{n}}.\end{aligned}$$ ◀

4 Application to the c-MultiQueue

In this section we apply our results on the general σ-MultiQueue to the c-MultiQueue with $c > 1$. After checking in Lemma 8 that the corresponding σ satisfies $(\star)$, we proceed to compute expected rank errors (Theorem 3) and derive a concentration bound (Theorem 9). This involves straightforward (though mildly tedious) calculations.

▶ **Lemma 8.** *In the c-MultiQueue with $c = \lfloor c \rfloor + \varepsilon > 1$ we have*

$$\sigma_{\le i} = 1 - (1 - \tfrac{i}{n})^{\lfloor c \rfloor}(1 - \varepsilon\tfrac{i}{n}) \text{ for all } i \in [n], \text{ and } \sigma \text{ satisfies } (\star).$$

Proof. Recall that we sample $\lfloor c \rfloor$ queues with probability $1 - \varepsilon$ and $\lfloor c \rfloor + 1$ queues otherwise. We fail to select one of the first i queues only if none of them were sampled. Hence for $i < n$:

$$\begin{aligned}\sigma_{\le i} &= 1 - (1 - \varepsilon)(1 - \tfrac{i}{n})^{\lfloor c \rfloor} - \varepsilon(1 - \tfrac{i}{n})^{\lfloor c \rfloor + 1} = 1 - (1 - \tfrac{i}{n})^{\lfloor c \rfloor}(1 - \varepsilon + \varepsilon(1 - \tfrac{i}{n})) \\ &= 1 - (1 - \tfrac{i}{n})^{\lfloor c \rfloor}(1 - \varepsilon\tfrac{i}{n}) > 1 - (1 - \tfrac{i}{n}) = \tfrac{i}{n}.\end{aligned}$$

The inequality uses that we have $\lfloor c \rfloor \ge 2$, or $\varepsilon > 0$, or both. ◀

We now prove Theorem 3, restated here for easier reference.

▶ **Theorem 3.** *Consider the expectation $\mathbb{E}[R]$ of the rank error $R = r_{i^*} - 1$ of the c-MultiQueue in the long run (i.e., after convergence).*

(i) *For $c = 2$ we have $\mathbb{E}[R] = \frac{5}{6}n - 1 + \frac{1}{6n} = \frac{5}{6}n - \Theta(1)$.*

(ii) *For $c = 1 + \varepsilon$ with $\varepsilon \in (0, 1)$, we have $\mathbb{E}[R] = (\frac{1}{\varepsilon} - \frac{\varepsilon}{6})n - \frac{1}{\varepsilon} + \frac{\varepsilon}{6n} = (\frac{1}{\varepsilon} - \frac{\varepsilon}{6})n - \mathcal{O}(1/\varepsilon)$.*

(iii) *For any $c > 1$ we have* $n \cdot \int_0^1 f_c(x)\,dx - \frac{c}{c-1} \le \mathbb{E}[R] \le n \cdot \int_0^1 f_c(x)\,dx$, *where* $f_c : (0,1) \to \mathbb{R}$ *is defined as follows using* $\varepsilon := c - \lfloor c \rfloor \in [0,1)$

$$f_c(x) = x^{\lfloor c \rfloor - 1}(1 - \varepsilon + \varepsilon x) \cdot \frac{1 - x^{\lfloor c \rfloor}(1 - \varepsilon + \varepsilon x)}{1 - x^{\lfloor c \rfloor - 1}(1 - \varepsilon + \varepsilon x)}.$$

(iv) *Cruder but simpler bounds for any* $c \ge 2$ *are:* $\frac{n}{\lceil c \rceil} - \frac{c}{c-1} \le \mathbb{E}[R] \le \frac{n}{\lfloor c \rfloor - 1}$.

Proof of Theorem 3. We have just checked in Lemma 8 that σ satisfies $(\star)$ and computed $\sigma_{\le i} = 1 - (1 - \frac{i}{n})^{\lfloor c \rfloor}(1 - \varepsilon\frac{i}{n})$. We can therefore specialise the formula for the expected rank error from Corollary 2 by plugging in σ and simplifying, which yields

$$\begin{aligned}
\mathbb{E}[R] &= \sum_{i=1}^{n-1}(1 - \sigma_{\le i})\frac{\sigma_{\le i}}{\sigma_{\le i} - \frac{i}{n}} = \sum_{i=1}^{n-1}(1 - \tfrac{i}{n})^{\lfloor c \rfloor}(1 - \varepsilon\tfrac{i}{n})\frac{1 - (1 - \frac{i}{n})^{\lfloor c \rfloor}(1 - \varepsilon\frac{i}{n})}{1 - (1 - \frac{i}{n})^{\lfloor c \rfloor}(1 - \varepsilon\frac{i}{n}) - \frac{i}{n}} \\
&= \sum_{i=1}^{n-1}(1 - \tfrac{i}{n})^{\lfloor c \rfloor - 1}(1 - \varepsilon\tfrac{i}{n})\frac{1 - (1 - \frac{i}{n})^{\lfloor c \rfloor}(1 - \varepsilon\frac{i}{n})}{1 - (1 - \frac{i}{n})^{\lfloor c \rfloor - 1}(1 - \varepsilon\frac{i}{n})} && \text{(cancel } 1 - \tfrac{i}{n}\text{)} \\
&= \sum_{i=1}^{n-1}(\tfrac{i}{n})^{\lfloor c \rfloor - 1}(1 - \varepsilon + \varepsilon\tfrac{i}{n})\frac{1 - (\frac{i}{n})^{\lfloor c \rfloor}(1 - \varepsilon + \varepsilon\frac{i}{n})}{1 - (\frac{i}{n})^{\lfloor c \rfloor - 1}(1 - \varepsilon + \varepsilon\frac{i}{n})} && \text{(substitute } i \to n - i.\text{)} \\
&= \sum_{i=1}^{n-1} f_c(\tfrac{i}{n}). && \text{(using the definition of } f_c(x) \text{ given above.)}
\end{aligned}$$

We are now ready to prove claim (i). Using that $f_2(x) = x \cdot \frac{1-x^2}{1-x} = x \cdot (1 + x) = x + x^2$, we have

$$\begin{aligned}
\mathbb{E}[R] &= \sum_{i=1}^{n-1} f_2(\tfrac{i}{n}) = \sum_{i=1}^{n-1}\left(\tfrac{i}{n} + (\tfrac{i}{n})^2\right) = \frac{1}{n}\sum_{i=1}^{n-1} i + \frac{1}{n^2}\sum_{i=1}^{n-1} i^2 \\
&= \frac{n-1}{2} + \frac{(n-1)(2n-1)}{6n} = \tfrac{5}{6}n - 1 + \tfrac{1}{6n}.
\end{aligned}$$

Similarly, we can prove (ii). First, we simplify $f_c(x)$ for $c = 1 + \varepsilon$ with $\varepsilon \in (0,1)$:

$$\begin{aligned}
f_{1+\varepsilon}(x) &= (1 - \varepsilon + \varepsilon x) \cdot \frac{1 - x(1 - \varepsilon + \varepsilon x)}{\varepsilon - \varepsilon x} = (1 - \varepsilon + \varepsilon x) \cdot \frac{(1-x)(1 + \varepsilon x)}{\varepsilon(1-x)} \\
&= (1 - \varepsilon + \varepsilon x) \cdot \frac{1 + \varepsilon x}{\varepsilon} = \tfrac{1-\varepsilon}{\varepsilon} + (2 - \varepsilon)x + \varepsilon x^2.
\end{aligned}$$

We then get (omitting a simple calculation):

$$\begin{aligned}
\mathbb{E}[R] &= \sum_{i=1}^{n-1} f_{1+\varepsilon}(\tfrac{i}{n}) = \sum_{i=1}^{n-1}(\tfrac{1-\varepsilon}{\varepsilon} + (2 - \varepsilon)\tfrac{i}{n} + \varepsilon(\tfrac{i}{n})^2) \\
&= \tfrac{1-\varepsilon}{\varepsilon}(n-1) + (2 - \varepsilon)\tfrac{n(n-1)}{2n} + \varepsilon\tfrac{(n-1)n(2n-1)}{6n^2} = \ldots = (\tfrac{1}{\varepsilon} - \tfrac{\varepsilon}{6})n - \tfrac{1}{\varepsilon} + \tfrac{\varepsilon}{6n}.
\end{aligned}$$

We now turn our attention to general $c > 1$ again. Our goal is to approximate the sum by an integral. To bound the approximation error effectively, we will first show that f_c is monotonic. For this let us examine the fractional term $g_c(x)$ occurring in $f_c(x)$:

$$g_c(x) := \frac{1-x^{\lfloor c\rfloor}(1-\varepsilon+\varepsilon x)}{1-x^{\lfloor c\rfloor-1}(1-\varepsilon+\varepsilon x)} = 1+\frac{(x^{\lfloor c\rfloor-1}-x^{\lfloor c\rfloor})(1-\varepsilon+\varepsilon x)}{1-x^{\lfloor c\rfloor-1}(1-\varepsilon+\varepsilon x)}$$
$$= 1+\frac{x^{\lfloor c\rfloor-1}(1-x)(1-\varepsilon+\varepsilon x)}{1-x^{\lfloor c\rfloor-1}+\varepsilon x^{\lfloor c\rfloor-1}(1-x)} = 1+\frac{x^{\lfloor c\rfloor-1}(1-x)(1-\varepsilon+\varepsilon x)}{(1-x)\sum_{i=0}^{\lfloor c\rfloor-2}x^i+\varepsilon x^{\lfloor c\rfloor-1}(1-x)}$$
$$= 1+\frac{1-\varepsilon+\varepsilon x}{\sum_{i=1}^{\lfloor c\rfloor-1}x^{-i}+\varepsilon}. \tag{1}$$

From (1) we can see that $g_c(x)$ does not have a singularity at $x=1$ and by setting $g_c(1)=f_c(1)=1+\frac{1}{c-1}=\frac{c}{c-1}$ both functions are continuous on $[0,1]$. We also see from (1) that $g_c(x)$ is increasing in x and decreasing in c (recall that c determines $\varepsilon=c-\lfloor c\rfloor$). Because the other factor $x^{\lfloor c\rfloor-1}(1-\varepsilon+\varepsilon x)$ of $f_c(x)$ is also increasing in x and decreasing in c, we conclude that $f_c(x)$ as a whole is increasing in x and decreasing in c. This makes the upper and lower sums bounding the integral of f_c particularly simple:

$$\sum_{i=1}^{n-1}f_c(\tfrac{i}{n})=\sum_{i=0}^{n-1}f_c(\tfrac{i}{n})\le n\int_0^1 f_c(x)\,dx\le\sum_{i=1}^{n}f_c(\tfrac{i}{n})=f_c(1)+\sum_{i=1}^{n-1}f_c(\tfrac{i}{n})=\tfrac{c}{c-1}+\sum_{i=1}^{n-1}f_c(\tfrac{i}{n}).$$

By rearranging we obtain (iii). We now turn to the cruder bound (iv). We begin with $c\in\mathbb{N}\setminus\{1\}$. In this case we have $f_c(x)=x^{c-1}\cdot g_c(x)$ where $1\le g_c(x)\le\frac{c}{c-1}$ for all $x\in[0,1]$. This gives:

$$\tfrac{1}{c}=\int_0^1 x^{c-1}\,dx\le\int_0^1 f_c(x)\,dx\le\int_0^1 x^{c-1}\tfrac{c}{c-1}\,dx=\tfrac{c}{c-1}\int_0^1 x^{c-1}\,dx=\tfrac{1}{c-1}.$$

Combining this with (iii) gives $\frac{n}{c}-\frac{c}{c-1}\le\mathbb{E}[R]\le\frac{n}{c-1}$ when $c\in\mathbb{N}\setminus\{1\}$. When $c\notin\mathbb{N}$ we can use that $f_c(x)$ is decreasing in c and round conservatively to obtain the claimed result. ◀

▶ **Theorem 9.** *In the 2-MultiQueue[4], the highest rank error observed over a polynomial number of deletions is in $\mathcal{O}(n\log n)$ with high probability.*

Proof. We will use a tail bound by Janson [9, Theorem 2.1] on the sum of geometrically distributed random variables from. It reads

$$\Pr[X\ge k\cdot\mu]\le e^{-p_*\mu(k-1-\log k)}\text{ for any }k\ge 1,$$

where X is a sum of geometrically distributed random variables (with possibly differing success probabilities), p_* is the smallest success probability and $\mu=\mathbb{E}[X]$.

We apply this bound to $X=r_n-1$, which has the required form by Theorem 1. It is easy to check that $p_*=\delta_{n-1}=\frac{1}{n+1}=\Theta(\frac{1}{n})$ and

$$\mu=\mathbb{E}[X]=\sum_{i=1}^{n-1}\frac{1-(1-\frac{i}{n})^2}{1-(1-\frac{i}{n})^2-\frac{i}{n}}=\sum_{i=1}^{n-1}\frac{2-\frac{i}{n}}{1-\frac{i}{n}}=n-1+n\sum_{i=1}^{n-1}\tfrac{1}{i}=\Theta(n\log n).$$

Since the rank error R clearly satisfies $R\le r_n-1=X$ we have for k large enough

$$\Pr[R>k\cdot n\log n]\le\Pr[X>k\cdot n\log n]=e^{-\Theta(\frac{1}{n}n\log(n)k)}=n^{-\Theta(k)}.$$

By a union bound, the probability that we observe a rank error exceeding $k\cdot n\log n$ at some point during n^c deletions is at most $n^c\cdot n^{-\Theta(k)}$. For large enough k, this probability is still small. ◀

[4] A similar analysis for $c=1+\varepsilon$ is also possible.

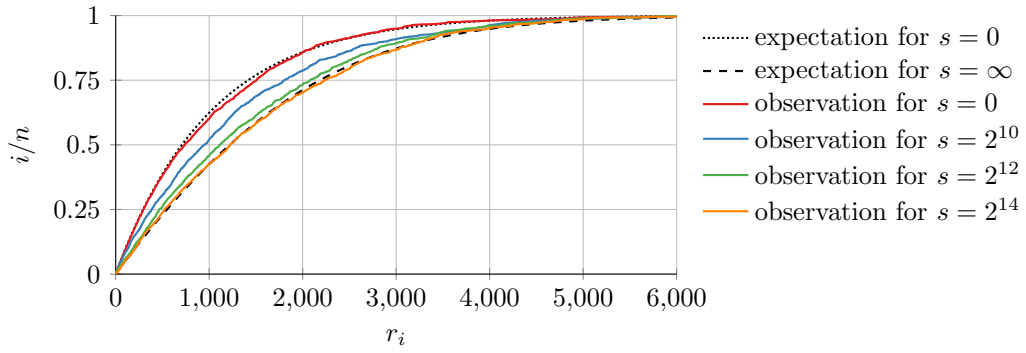


Figure 5 The convergence of the 2-MultiQueue with $n = 2^{10}$ to its stable state. After s deletions, let $r_i^{(s)}$ be the rank of queue i. The plot shows the observed ranks $i \mapsto r_i^{(s)}$ for some values of s, as well as the expected ranks $i \mapsto \mathbb{E}[r_i^{(0)}]$ and $i \mapsto \mathbb{E}[r_i^{(\infty)}]$ in the initial state and the converged state.

To give some intuition for how quickly the 2-MultiQueue converges to its stable state and for how close the observed ranks of top-elements are to their expectation we provide experimental data in Figure 5.

5 Future Work

While we have fully analyzed the long-term behavior of the MultiQueue in the deletion-only setting, there are still open questions in the general setting. Specifically, our methods are not directly applicable when insertions of arbitrary elements can happen after deletions. We conjecture that the deletion-only setting represents the worst-case in the sense that the expected rank error cannot be made worse even by adversarial insertions. On the flip side, we conjecture that the expected rank error is never better than before the first deletion. Our reasoning is as follows: Inserting sufficiently large elements does not affect deletions and is equivalent to inserting before deleting. When inserting many small elements, the state drifts towards the state where no deletions happened. In summary, we expect the state of the MultiQueue always to be "between" the insertion-only and the deletion-only setting, and that our analysis yields accurate predictions after sufficiently many deletions regardless of previous operations.

Our analysis applies directly to applications that do insert new elements after the first deletion, such as heap sort and simple batch job scheduling. A much broader range of applications, including Dijkstra's algorithm and discrete-event simulations, process elements in *monotonous* fashion, meaning that the elements deleted from the priority queue are monotonically increasing. When using a MultiQueue, newly inserted elements are unlikely to become top-elements, and we expect the state of the MultiQueue to stay "close" to the deletion-only setting.

Williams et al. [21] proposed the *delay* as an additional quality metric and *stickiness* as a way to increase throughput. The delay of an element e measures how many elements worse than e are deleted while e resides in the queue. Stickiness lets threads reuse the same queue for multiple consecutive operations. We believe that the delay can be analyzed directly with our approach and that the Markov chain can be adapted to handle stickiness as well.

In practice, it is relevant how fast the system stabilizes and converges to the postulated distributions of ranks or what rank errors are to be expected until then. Thus, analyzing the convergence speed is a natural next step.

Alistarh et al. [1] analyze the MultiQueue in concurrent settings where comparisons can become *stale*, meaning that after deciding which queue to delete from but before actually deleting from it, its top-element might change. We find it interesting whether our analysis can be adapted to this scenario as well.

References

1 Dan Alistarh, Trevor Brown, Justin Kopinsky, Jerry Z. Li, and Giorgi Nadiradze. Distributionally Linearizable Data Structures. In *SPAA*, 2018. doi:10.1145/3210377.3210411.

2 Dan Alistarh, Justin Kopinsky, Jerry Li, and Giorgi Nadiradze. The Power of Choice in Priority Scheduling. In *PODC*, 2017. doi:10.1145/3087801.3087810.

3 Dan Alistarh, Justin Kopinsky, Jerry Li, and Nir Shavit. The SprayList: A scalable relaxed priority queue. In *PPoPP*, 2015. doi:10.1145/2688500.2688523.

4 Hagit Attiya, Rachid Guerraoui, Danny Hendler, Petr Kuznetsov, Maged M. Michael, and Martin Vechev. Laws of order: Expensive synchronization in concurrent algorithms cannot be eliminated. In *POPL*, 2011. doi:10.1145/1926385.1926442.

5 Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced Allocations. *SICOMP*, 29(1), 1999. doi:10.1137/S0097539795288490.

6 Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced Allocations: The Heavily Loaded Case. *SICOMP*, 35(6), 2006. doi:10.1137/S009753970444435X.

7 Irina Calciu, Hammurabi Mendes, and Maurice Herlihy. The Adaptive Priority Queue with Elimination and Combining. In *DC*, 2014. doi:10.1007/978-3-662-45174-8_28.

8 Faith Ellen, Danny Hendler, and Nir Shavit. On the Inherent Sequentiality of Concurrent Objects. *SICOMP*, 41(3), 2012. doi:10.1137/08072646X.

9 Svante Janson. Tail bounds for sums of geometric and exponential variables. *S&P L*, 135, 2018. doi:10.1016/j.spl.2017.11.017.

10 Richard M. Karp and Yanjun Zhang. Randomized parallel algorithms for backtrack search and branch-and-bound computation. *JACM*, 40(3), 1993. doi:10.1145/174130.174145.

11 Jonatan Lindén and Bengt Jonsson. A Skiplist-Based Concurrent Priority Queue with Minimal Memory Contention. In *OPODIS*, 2013. doi:10.1007/978-3-319-03850-6_15.

12 Michael Mitzenmacher, Andréa W. Richa, and Ramesh Sitaraman. The Power of Two Random Choices: A Survey of Techniques and Results, 2001. URL: http://www.eecs.harvard.edu/~michaelm/postscripts/handbook2001.pdf.

13 J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge university press, 1997. doi:10.1017/CBO9780511810633.

14 Yuval Peres, Kunal Talwar, and Udi Wieder. Graphical balanced allocations and the $(1+\beta)$-choice process. *RS & A*, 47(4), 2015. doi:10.1002/rsa.20558.

15 Anastasiia Postnikova, Nikita Koval, Giorgi Nadiradze, and Dan Alistarh. Multi-queues can be state-of-the-art priority schedulers. In *PPoPP*, 2022. doi:10.1145/3503221.3508432.

16 Hamza Rihani, Peter Sanders, and Roman Dementiev. MultiQueues: Simple Relaxed Concurrent Priority Queues. In *SPAA*, 2015. doi:10.1145/2755573.2755616.

17 Adones Rukundo and Philippas Tsigas. TSLQueue: An Efficient Lock-Free Design for Priority Queues. In *Euro-Par*, 2021. doi:10.1007/978-3-030-85665-6_24.

18 Konstantinos Sagonas and Kjell Winblad. The Contention Avoiding Concurrent Priority Queue. In *LCPC*, 2017. doi:10.1007/978-3-319-52709-3_23.

19 N. Shavit and I. Lotan. Skiplist-based concurrent priority queues. In *IPDPS*, 2000. doi:10.1109/IPDPS.2000.845994.

20 Stefan Walzer and Marvin Williams. A Simple yet Exact Analysis of the MultiQueue, 2025. doi:10.48550/arXiv.2410.08714.

21 Marvin Williams, Peter Sanders, and Roman Dementiev. Engineering MultiQueues: Fast Relaxed Concurrent Priority Queues. In *ESA*, 2021. doi:10.4230/LIPIcs.ESA.2021.81.

22 Martin Wimmer, Jakob Gruber, Jesper Larsson Träff, and Philippas Tsigas. The lock-free k-LSM relaxed priority queue. In *PPoPP*, 2015. doi:10.1145/2688500.2688547.

23 Guozheng Zhang, Gilead Posluns, and Mark C. Jeffrey. Multi Bucket Queues: Efficient Concurrent Priority Scheduling. In *SPAA*, 2024. doi:10.1145/3626183.3659962.

New Algorithms for Pigeonhole Equal Subset Sum

Ce Jin
MIT, Cambridge, MA, USA

Ryan Williams
MIT, Cambridge, MA, USA

Stan Zhang
MIT, Cambridge, MA, USA

Abstract

We study the *Pigeonhole Equal Subset Sum* problem, which is a total-search variant of the Subset Sum problem introduced by Papadimitriou (1994): we are given a set of n positive integers $\{w_1, \ldots, w_n\}$ with the additional restriction that $\sum_{i=1}^{n} w_i < 2^n - 1$, and want to find two different subsets $A, B \subseteq [n]$ such that $\sum_{i \in A} w_i = \sum_{i \in B} w_i$.

Very recently, Jin and Wu (ICALP 2024) gave a randomized algorithm solving Pigeonhole Equal Subset Sum in $O^*(2^{0.4n})$ time, beating the classical meet-in-the-middle algorithm with $O^*(2^{n/2})$ runtime. In this paper, we refine Jin and Wu's techniques to improve the runtime even further to $O^*(2^{n/3})$.

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

Keywords and phrases pigeonhole principle, subset sums

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.86

Related Version This paper is based on Stan Zhang's Master's thesis.
Previous Version: `https://hdl.handle.net/1721.1/156830` [25]

Funding *Ce Jin*: Supported by the Jane Street Graduate Research Fellowship, NSF grant CCF-2330048, and a Simons Investigator Award.
Ryan Williams: Work supported in part by NSF CCF-2420092.
Stan Zhang: Self Funded

1 Introduction

Subset Sum is a fundamental NP-hard problem: given positive integers $w_1, w_2, \ldots, w_n$ and a target integer t, we want to find a subset $A \subseteq [n]$ such that $\sum_{i \in A} w_i = t$.[1] Subset Sum has a simple meet-in-the-middle algorithm in about $O(2^{n/2})$ time, given by Horowitz and Sahni in 1974 [13]. There has been a long line of research attempting to improve that meet-in-the-middle running time for Subset Sum [4, 5, 22, 3, 14, 8, 9, 12], but it remains open whether there exists an algorithm running in $O(2^{(1/2-\varepsilon)n})$ time for any constant $\varepsilon > 0$. The fastest known algorithm for Subset Sum only has a poly(n)-factor improvement over $O(2^{n/2})$ [12].

Despite the lack of progress on the time complexity of Subset Sum, there has been significant progress for solving *variants* of the problem. Examples include Equal Subset Sum [16], 2-Subset Sum and Shifted Sums [2], more general subset balancing problems [11], algorithms solving either Equal Subset Sum or Subset Sum [20], Merlin–Arthur protocols for Subset Sum [17, 1], Subset Sum in low space [7, 18], and quantum algorithms for Subset Sum [2]. There is also a large body of works on pseudo-polynomial-time algorithms and approximation schemes for Subset Sum and related problems (see e.g., [10] and the references therein). In this work, we focus on exact exponential-time algorithms.

[1] Throughout this paper, we write $[n] = \{1, \ldots, n\}$.

© Ce Jin, Ryan Williams, and Stan Zhang;
licensed under Creative Commons License CC-BY 4.0

An important variant of Subset Sum is the Equal Subset Sum problem:

EQUAL SUBSET SUM [24]
Input: integers $w_1, w_2, \ldots, w_n$ where $|w_i| \leq 2^{\text{poly}(n)}$
Output: two different subsets $A, B \subseteq [n]$ such that $\sum_{i \in A} w_i = \sum_{i \in B} w_i$ (if exist).

Woeginger and Yu [24] first studied the Equal Subset Sum problem and showed that it is NP-Complete, similarly to Subset Sum. Since we can assume the solution satisfies $A \cap B = \emptyset$ (otherwise, we can return $(A \setminus B, B \setminus A)$ instead), a standard meet-in-the-middle approach can solve Equal Subset Sum in $O^*(3^{n/2}) \leq O^*(1.7321^n)$ time. Mucha, Nederlof, Pawlewicz, and Węgrzycki [16] gave a faster $O^*(1.7088^n)$-time algorithm in 2019, answering an open question of [23].[2]

A special case of the Equal Subset Sum problem is the *Pigeonhole Equal Subset Sum* problem (also sometimes called Pigeonhole Equal Sums). This problem is the focus of our paper.

PIGEONHOLE EQUAL SUBSET SUM [19]
Input: positive integers $w_1, w_2, \ldots, w_n$, with promise $\sum_{i=1}^{n} w_i < 2^n - 1$.
Output: two different subsets $A, B \subseteq [n]$ such that $\sum_{i \in A} w_i = \sum_{i \in B} w_i$.

Pigeonhole Equal Subset Sum is a *total search problem*, in that there always exists a solution; our goal is merely to find one. To see why this is true, note that there are 2^n subsets $S \subseteq [n]$, which can only attain $2^n - 1$ possible subset sums $\sum_{i \in S} w_i \in \{0, 1, \ldots, 2^n - 2\}$ due to the promise $\sum_{i=1}^{n} w_i < 2^n - 1$, so by the pigeonhole principle there must exist a pair of subsets with the same subset sum.

The Pigeonhole Equal Subset Sum problem was introduced by Papadimitriou in 1994 [19], and has been studied in the TFNP literature [6, 21]. It belongs to the total search complexity class PPP, and is conjectured to be PPP-complete [19].

Pigeonhole Equal Subset Sum can be solved faster than the best known Equal Subset Sum algorithm. Until very recently, the fastest known algorithm for Pigeonhole Equal Subset Sum had time complexity $O^*(2^{n/2})$, either by a simple binary search combined with the classical meet-in-the-middle approach by Horowitz and Sahni [13] (see a short description in [15, Section 2]), or by a mod-p dynamic programming algorithm by Allcock, Hamoudi, Joux, Klingelhöfer, and Santha [2, Theorem 6.2]. Recently, a major advance by Jin and Wu [15] gave a randomized algorithm for Pigeonhole Equal Subset Sum running in $O^*(2^{0.4n})$ time. Roughly speaking, the key idea of Jin and Wu was to show that a Pigeonhole Equal Subset Sum instance with very few solutions must satisfy a certain structural property: namely, the input numbers should resemble the geometric progression $1, 2, 4, \ldots, 2^{n-1}$ (see the formal statement in Lemma 4).

1.1 Our contribution

In this paper, we improve the time complexity of Jin and Wu [15]'s Pigeonhole Equal Subset Sum algorithm from $O^*(2^{0.4n})$ to the clean bound $O^*(2^{n/3})$:

▶ **Theorem 1** (Main). *Pigeonhole Equal Subset Sum can be solved by a randomized algorithm in $O^*(2^{n/3})$ time.*

[2] Throughout this paper, we use $O^*(\cdot), \Omega^*(\cdot)$ to hide $\text{poly}(n)$ factors, where n is the number of input integers.

Jin and Wu used similar techniques to give a *polynomial-space* algorithm for Pigeonhole Equal Subset Sum in $O^*(2^{0.75n})$ time, improving the straightforward $O^*(2^n)$-time polynomial-space algorithm based on binary search and meet-in-the-middle. We show that their polynomial-space algorithm can also be further improved:

▶ **Theorem 2.** *Pigeonhole Equal Subset Sum can be solved by a randomized algorithm in* $O^*(2^{2n/3})$ *time and* $\mathrm{poly}(n)$ *space.*

Our improved algorithms are based on the same structural property (see the formal statement in Lemma 4) proved by Jin and Wu [15], but we exploit it in a more efficient way. Jin and Wu [15] only exploited the property in the case when there are few solutions; if there are many solutions, they simply switched to a subsampling approach to try to hit a solution randomly, without using the structural property. In contrast, we make use of the structural property even in the case where there are many solutions, thereby improving the overall time complexity.

The rest of the paper is organized as follows: In Section 2, we give an overview of Jin and Wu's algorithm, describe their aforementioned structural property precisely, and extract a few lemmas from their work which are useful to us. Then in Section 3, we present our improved algorithm for Pigeonhole Equal Subset Sum, proving Theorem 1. In Section 4 we prove Theorem 2. Finally, we discuss some future research directions in Section 5.

2 A Summary of Jin and Wu's Algorithm

Throughout this paper, we use $w_1, \ldots, w_n$ to denote the input integers, and we use the shorthand $w(A) = \sum_{i \in A} w_i$ for the subset sum attained by the subset of indices $A \subseteq [n]$.

For an Equal Subset Sum instance $(w_1, \ldots, w_n)$ (not necessarily satisfying the pigeonhole promise), Jin and Wu [15] considered the parameter $F(w_1, \ldots, w_n)$ defined as follows:

▶ **Definition 3** ($f_t(\cdot)$ and $F(\cdot)$)**.** *For* $w_1, \ldots, w_n \in \mathbb{Z}$ *and* $t \in \mathbb{Z}$*, let*

$$f_t(w_1, \ldots, w_n) \coloneqq |\{S \subseteq [n] : w(S) = t\}|,$$

be the number of ways to achieve subset sum t*. When* $w_1, \ldots, w_n$ *are clear from the context, we abbreviate* $f_t = f_t(w_1, \ldots, w_n)$*.*

Then, define

$$F(w_1, \ldots, w_n) \coloneqq \sum_{t \in \mathbb{Z}} \max\{0, f_t - 1\}.$$

Observe that $F(w_1, \ldots, w_n) \in \{0, 1, \ldots, 2^n - 1\}$*.*

Note that an Equal Subset Sum instance $(w_1, \ldots, w_n)$ has a solution $A, B \subseteq [n], A \neq B, w(A) = w(B)$ if and only if $F(w_1, \ldots, w_n) \geq 1$.

Suppose the Pigeonhole Equal Subset Sum input instance consists of positive integers $w_1 < w_2 < \cdots < w_n$ (assuming no trivial solution $w_i = w_j, i \neq j$ exists) where $w([n]) < 2^n - 1$ (recall that this upper bound on the sum of all weights is the promise given by Pigeonhole Equal Subset Sum). Jin and Wu's key structural property relies on the following extra condition [15, Equation (1)]:

$$w([i]) \geq 2^i - 1, \text{ for all } i \in [n-1]. \tag{1}$$

That is, Equation (1) says that the total weight of the set $\{1, \ldots, i\}$ is at least $2^i - 1$. We may assume without loss of generality that Equation (1) holds. To see why, note that if $w([i]) < 2^i - 1$ for some $i \leq n-1$, then $\{w_1, \ldots, w_i\}$ is also a Pigeonhole Equal Subset Sum instance. We can solve this strictly smaller instance instead, and obtain a solution $A, B \subseteq [i] \subsetneq [n], A \neq B, w(A) = w(B)$. Hence, we may assume such i does not exist, so Equation (1) holds.

Using Equation (1), Jin and Wu [15] proved a key structural lemma for Pigeonhole Equal Subset Sum instances $(w_1, \ldots, w_n)$. Their lemma says that if $F(w_1, \ldots, w_n)$ is small, then the sequence $w_1, \ldots, w_n$ is close to the geometric progression $1, 2, 4, \ldots, 2^{n-1}$ with small additive error.

▶ **Lemma 4** ([15, Equation (8)]). *Suppose positive integers $w_1 < w_2 < \cdots < w_n$ satisfy $w([n]) < 2^n - 1$ and Equation (1). Then, for all $i \in [n]$,*

$$-i \cdot F(w_1, \ldots, w_n) \leq w_i - 2^{i-1} \leq F(w_1, \ldots, w_n). \tag{2}$$

Lemma 4 will be used in our improved algorithm as well. For completeness, in Section A we include a short proof of Lemma 4 given by Jin and Wu [15]. Jin and Wu gave two different algorithms, which we summarize below in Lemma 5 and Lemma 6. The first algorithm is suitable when the Pigeonhole Equal Subset Sum instance $(w_1, \ldots, w_n)$ has small $F(w_1, \ldots, w_n)$. It was obtained by using Lemma 4 to speed up the standard binary search and meet-in-the-middle approach:

▶ **Lemma 5** ([15, Lemma 4]). *Given parameter $1 \leq \Delta \leq 2^n/(3n^2)$, a Pigeonhole Equal Subset Sum instance $(w_1, \ldots, w_n)$ (where $w_i < w_{i+1}$) satisfying $F(w_1, \ldots, w_n) \leq \Delta$ and Equation (1) can be solved deterministically in $O^*(\sqrt{\Delta})$ time.*

The second algorithm of Jin and Wu works well when $F(w_1, \ldots, w_n)$ is large. The intuition is that, in this case there are many solutions, so one can perform subsampling and still expect at least one solution to survive. Jin and Wu used this subsampling idea to speed up the mod-p dynamic programming algorithm (which had previously been used for Subset Sum and related problems, e.g., [2, 4]):

▶ **Lemma 6** ([15, Lemma 5]). *Given parameter $2^{n/2} \leq \Delta < 2^n$, an Equal Subset Sum instance $(w_1, \ldots, w_n)$ with $F(w_1, \ldots, w_n) \geq \Delta$ can be solved in $O^*((2^{2n}/\Delta)^{1/3})$ time by a randomized algorithm.*

We remark that originally [15, Lemma 5] was stated for Pigeonhole Equal Subset Sum only. However, upon inspecting their proof, one can easily verify that the pigeonhole promise was never used, so that the algorithm actually works for Equal Subset Sum as well. This point turns out to be useful for our improvement later.

Jin and Wu's final algorithm for Pigeonhole Equal Subset Sum combines Lemma 5 and Lemma 6, and the overall runtime is balanced at the bottleneck case $\Delta \approx 2^{0.8n}$, leading to an overall worst case runtime of $O^*(2^{0.4n})$.

In Jin and Wu's algorithm, Lemma 4 was used in developing the algorithm of Lemma 5 (the algorithm for the small $F(w_1, \ldots, w_n)$ case), but was ignored in the case where $F(w_1, \ldots, w_n)$ is large. In our improved algorithm, we show how the structural property of Lemma 4 can be exploited even when $F(w_1, \ldots, w_n)$ is large. This will allow us to reduce the original Pigeonhole Equal Subset Sum instance to a small number of Equal Subset Sum instances, which have sizes much smaller than n (and still have large F-values).

3 Our Improvement for Pigeonhole Equal Subset Sum

In this section we prove our main Theorem 1. Let $w_1 < w_2 < \cdots < w_n$ be the input Pigeonhole Equal Subset Sum instance. Throughout, let $k \in \{0, 1, \ldots, n-1\}$ be such that

$$2^k \leq F(w_1, \ldots, w_n) < 2^{k+1}. \tag{3}$$

Our algorithm does not know the value of k in advance; instead, it tries all n possible values of k in parallel, and terminates as soon as any one of them succeeds. This only increases the runtime by an n factor. Hence, in the following we assume that the value of k is known.

We will separately consider the input items in $[n] \setminus [k] = \{k+1, \ldots, n\}$ and in $[k]$. At a high level, our algorithm performs the following two steps:

1. Guess a suitable pair of subsets $X, Y \subseteq [n] \setminus [k]$.
2. Find subsets $A_0, B_0 \subseteq [k]$ such that $w(A_0) - w(B_0) = w(Y) - w(X)$, i.e., $(A_0 \sqcup X, B_0 \sqcup Y)$ is a solution for Pigeonhole Equal Subset Sum.

For the first step, we will use Lemma 4 and Equation (3) to show that there are only $\mathrm{poly}(n)$ many pairs of $X, Y \subseteq [n] \setminus [k]$ that we need to consider. For the second step, we will reduce the task of finding A_0, B_0 to a smaller Equal Subset Sum instance with $(k+1)$ input integers and large F-value, which can be solved by Lemma 6. (In comparison, Jin and Wu's original algorithm applied Lemma 6 to solve Equal Subset Sum on n integers.)

Now we define the *close pairs*, which are the pairs of subsets (X, Y) that we need to consider for the first step:

▶ **Definition 7** (Close pairs). *Define the set of* close pairs *as*

$$\mathcal{C} := \{(X, Y) : X, Y \subseteq [n] \setminus [k] \text{ and } |w(X) - w(Y)| \leq (k+1)2^{k+1}\}.$$

Define the set of disjoint close pairs *as*

$$\mathcal{D} := \{(X, Y) \in \mathcal{C} : X \cap Y = \emptyset\}.$$

By the identity $w(X) - w(Y) = w(X \setminus Y) - w(Y \setminus X)$, we have

$$\mathcal{D} = \{(X \setminus Y, Y \setminus X) : (X, Y) \in \mathcal{C}\}. \tag{4}$$

The following simple lemma explains why it is sufficient to only consider close pairs:

▶ **Lemma 8.** *If $A, B \subseteq [n]$ and $w(A) = w(B)$, then $(A \setminus [k], B \setminus [k]) \in \mathcal{C}$.*

Proof. Since $w(A \setminus [k]) + w(A \cap [k]) = w(A) = w(B) = w(B \setminus [k]) + w(B \cap [k])$, we have

$$|w(A \setminus [k]) - w(B \setminus [k])| = |w(B \cap [k]) - w(A \cap [k])| \leq w([k]).$$

By the upper bounds in Lemma 4 and Equation (3), we have

$$w([k]) \leq \sum_{i=1}^{k} (2^{i-1} + F(w_1, \ldots, w_n)) < (k+1)2^{k+1}.$$

The claim then follows from the definition of $\mathcal{C}$. ◀

Now we bound the number of close pairs and disjoint close pairs:

▶ **Lemma 9.** *We have $|\mathcal{D}| \leq 200n^5$ and $|\mathcal{C}| \leq 200n^5 2^{n-k}$. Moreover, $\mathcal{D}$ can be computed in $\mathrm{poly}(n)$ time.*

Proof. First, we prove the claimed upper bound on $|\mathcal{D}|$. For $X \subseteq [n] \setminus [k]$, denote $\widetilde{w}(X) \coloneqq \sum_{i \in X} 2^{i-1}$. By Lemma 4 and Equation (3), for all $i \in [n]$,

$$|w_i - 2^{i-1}| \leq i \cdot F(w_1, \ldots, w_n) \leq n \cdot 2^{k+1}.$$

Hence, $|\widetilde{w}(X) - w(X)| \leq n^2 \cdot 2^{k+1}$ for all $X \subseteq [n] \setminus [k]$. Therefore, for all $(X, Y) \in \mathcal{D}$, we have $|\widetilde{w}(X) - \widetilde{w}(Y)| \leq |w(X) - w(Y)| + 2n^2 \cdot 2^{k+1} \leq (k+1)2^{k+1} + 2n^2 \cdot 2^{k+1} \leq 6n^2 2^k$. Now it remains to characterize all members of $\mathcal{D}'$ defined as

$$\mathcal{D}' \coloneqq \{(X, Y) \in 2^{[n]\setminus[k]} \times 2^{[n]\setminus[k]} : X \cap Y = \emptyset \text{ and } |\widetilde{w}(X) - \widetilde{w}(Y)| \leq 6n^2 2^k\} \supseteq \mathcal{D}.$$

Note that $(\emptyset, \emptyset) \in \mathcal{D}'$. Now consider any $(X, Y) \in \mathcal{D}'$ not equal to $(\emptyset, \emptyset)$, and let i be the maximum element in $X \cup Y$. By symmetry, we assume $i \in X$ without loss of generality, and thus $Y \subseteq [i-1] \setminus [k]$. Then,

$$\begin{aligned}\widetilde{w}(X) - \widetilde{w}(Y) &= \Big(2^{i-1} + \sum_{j \in X \setminus \{i\}} 2^{j-1}\Big) - \Big(\sum_{j=k+1}^{i-1} 2^{j-1} - \sum_{j \in \{k+1,\ldots,i-1\} \setminus Y} 2^{j-1}\Big) \\ &= 2^k + \sum_{j \in X \setminus \{i\}} 2^{j-1} + \sum_{j \in \{k+1,\ldots,i-1\} \setminus Y} 2^{j-1} \geq 0.\end{aligned}$$

On the other hand, $\widetilde{w}(X) - \widetilde{w}(Y) \leq 6n^2 2^k$ by definition of $\mathcal{D}'$. Hence, for every $j \in [i-1] \setminus [k]$ with $2^{j-1} \geq 6n^2 2^k$, we must have $j \in Y$ and $j \notin X$ (otherwise, 2^{j-1} would appear as a summand above, violating $\widetilde{w}(X) - \widetilde{w}(Y) \leq 6n^2 2^k$). Then, to fully determine (X, Y), it only remains to decide for each integer $k+1 \leq j < \log_2(6n^2 2^k) + 1$ whether j belongs to X, Y, or neither. There are at most $3^{\log_2(6n^2 2^k)+1-k} = 3^{\log_2(6n^2)+1} < 60n^4$ possible choices in total. Since there are up to n choices of i and two choices for whether i belongs to X or Y, we conclude that $|\mathcal{D}'| \leq 1 + 2 \cdot n \cdot 60n^4 < 200n^5$.

The above argument can be easily converted into an algorithm that computes the set $\mathcal{D}'$ in $\mathrm{poly}(n)$ time. Since $\mathcal{D}' \supseteq \mathcal{D}$, we have $|\mathcal{D}| \leq 200n^5$ as claimed, and we can also compute $\mathcal{D}$ in $\mathrm{poly}(n)$ time by checking every pair in $\mathcal{D}'$ for containment in $\mathcal{C}$.

Now we bound $|\mathcal{C}|$. Recall from Equation (4) that every $(X, Y) \in \mathcal{C} \subseteq 2^{[n]\setminus[k]} \times 2^{[n]\setminus[k]}$ can be mapped to $(X \setminus Y, Y \setminus X) \in \mathcal{D}$, and note that every $(X', Y') \in \mathcal{D}$ can only have at most 2^{n-k} pre-images in $\mathcal{C}$ under this mapping. Hence, $|\mathcal{C}| \leq 2^{n-k}|\mathcal{D}| \leq 200n^5 2^{n-k}$. ◀

The following lemma allows us to reduce the original problem to Equal Subset Sum instances on k or $k+1$ integers with large F-value:

▶ **Lemma 10.** *For sets $X, Y \subseteq [n] \setminus [k]$, define the k- or $(k+1)$-tuple $W_{X,Y}$ as*

$$W_{X,Y} \coloneqq \begin{cases} (w_1, \ldots, w_k) & \text{if } X = Y, \\ (w_1, \ldots, w_k, w(X) - w(Y)) & \text{if } X \neq Y. \end{cases} \tag{5}$$

Then, there must exist $(X, Y) \in \mathcal{D}$, such that $F(W_{X,Y}) \geq F(w_1, \ldots, w_n)/|\mathcal{C}|$.[3]

Before proving Lemma 10, we first use it to finish the proof of our main theorem:

[3] Note that in our definition of $W_{X,Y}$ for $(X, Y) \in \mathcal{D}$, the first case $X = Y$ happens only if $X = Y = \emptyset$. However we later will consider $W_{X,Y}$ where (X, Y) may not belong to $\mathcal{D}$.

Proof of Theorem 1 (assuming Lemma 10). As mentioned in the beginning of this section, we can assume the integer k defined by Equation (3) is known. We assume k satisfies

$$2^k \geq 200n^5 \cdot 2^{2n/3}, \tag{6}$$

since otherwise we can already use Lemma 5 to solve the Pigeonhole Equal Subset Sum instance $(w_1, \ldots, w_n)$ in $O^*(\sqrt{2^{k+1}}) = O^*(2^{n/3})$ time as required.

Compute $\mathcal{D}$ with $|\mathcal{D}| \leq \text{poly}(n)$ by Lemma 9 in $\text{poly}(n)$ time. Enumerate each $(X, Y) \in \mathcal{D}$, and then solve the Equal Subset Sum instance $W_{X,Y}$ (defined in Equation (5)) using Lemma 6. If the algorithm in Lemma 6 successfully finds a solution $(U, V), U \neq V$ in the instance $W_{X,Y}$ (where we can assume $U \cap V = \emptyset$ without loss of generality), then we distinguish between two cases:

- **Case where $U, V \subseteq [k]$:**
  Then, since $w(U) = w(V)$ and $U \neq V$, we can return (U, V) as a solution for the original Pigeonhole Equal Subset Sum instance $w_1, \ldots, w_n$.
- **Case where $k + 1 \in U \cup V$:**
  This can only happen in the $X \neq Y$ case, where $W_{X,Y}$ contains the extra $(k+1)$-th element $w(X) - w(Y)$. Without loss of generality assume $k + 1 \in U, k + 1 \notin V$. Then, define $A = (U \setminus \{k+1\}) \sqcup X$ and $B = V \sqcup Y$, both of which are subsets of $[n]$. Note that $A \neq B$ due to $X \neq Y$. Then we have $w(A) = w(U \setminus \{k+1\}) + w(X) = \big(w(V) - (w(X) - w(Y))\big) + w(X) = w(V) + w(Y) = w(B)$. So we can return (A, B) as a solution for the original Pigeonhole Equal Subset Sum instance $w_1, \ldots, w_n$.

Now we analyze the runtime. By Lemma 10, for at least one $(X, Y) \in \mathcal{D}$, $F(W_{X,Y}) \geq F(w_1, \ldots, w_n)/|\mathcal{C}| \geq (200n^5)^{-1} \cdot 2^{2k-n} =: \Delta$ (where we used Lemma 9 and Equation (3)). Then, applying Lemma 6 to solve $W_{X,Y}$ takes time $O^*((2^{2(k+1)}/\Delta)^{1/3}) = O^*\big(2^{n/3}\big)$. Note that the precondition $\Delta \geq 2^{(k+1)/2}$ required by Lemma 6 is satisfied due to Equation (6). We can try all $|\mathcal{D}| \leq \text{poly}(n)$ possibilities for the pair (X, Y) in parallel and run the algorithm of Lemma 6 until the earliest of them succeeds. The total time complexity is still $O^*(2^{n/3})$. ◀

It remains to prove Lemma 10.

Proof of Lemma 10. We will show that there exists a pair $(X, Y) \in \mathcal{C}$ satisfying the desired bound $F(W_{X,Y}) \geq F(w_1, \ldots, w_n)/|\mathcal{C}|$. Once this is shown, by $w(X) - w(Y) = w(X \setminus Y) - w(Y \setminus X)$ we have $W_{X \setminus Y, Y \setminus X} = W_{X,Y}$, so the pair $(X \setminus Y, Y \setminus X) \in \mathcal{D}$ also satisfies the desired bound, which would finish the proof.

We start by rewriting $F(W_{X,Y})$ as follows:

▷ Claim 11. For sets $X, Y \subseteq [n] \setminus [k]$ and integer t, define

$$h(X, t) \coloneqq \big|\big\{S \subseteq [n] : S \setminus [k] = X \text{ and } w(S) = t\big\}\big|,$$

which is the number of ways to extend X into $[n]$ to achieve a sum of t, and define

$$g(X, Y, t) \coloneqq \begin{cases} \max\{h(X, t) + h(Y, t) - 1, 0\} & \text{if } X \neq Y, \\ \max\{h(X, t) - 1, 0\} & \text{if } X = Y. \end{cases}$$

Then, $F(W_{X,Y}) = \sum_{t \in \mathbb{Z}} g(X, Y, t)$.

Proof of Claim 11. By definition of $h(X,t)$, observe that $h(X,t) = f_{t-w(X)}(w_1,\dots,w_k)$ (where $f_t(\cdot)$ was defined in Definition 3). Hence,

$$\begin{aligned} F(W_{X,X}) &= \sum_{t'\in\mathbb{Z}} \max\{f_{t'}(w_1,\dots,w_k)-1,0\} \\ &= \sum_{t\in\mathbb{Z}} \max\{f_{t-w(X)}(w_1,\dots,w_k)-1,0\} \\ &= \sum_{t\in\mathbb{Z}} \max\{h(X,t)-1,0\}. \end{aligned}$$

This finishes the proof for the $X=Y$ case. Now observe that

$$\begin{aligned} h(X,t)+h(Y,t) &= f_{t-w(X)}(w_1,\dots,w_k) + f_{t-w(Y)}(w_1,\dots,w_k) \\ &= f_{t-w(Y)}(w_1,\dots,w_k,w(X)-w(Y)), \end{aligned}$$

where the last equality can be seen from separately considering subsets of $\{w_1,\dots,w_k,w(X)-w(Y)\}$ which include $w(X)-w(Y)$ and those which do not. Therefore, if $X\neq Y$, then we similarly have

$$\begin{aligned} F(W_{X,Y}) &= \sum_{t'\in\mathbb{Z}} \max\{f_{t'}(w_1,\dots,w_k,w(X)-w(Y))-1,0\} \\ &= \sum_{t\in\mathbb{Z}} \max\{h(X,t)+h(Y,t)-1,0\} \end{aligned}$$

as desired. ◁

Now we relate $F(w_1,\dots,w_n)=\sum_{t\in\mathbb{Z}}\max\{f_t(w_1,\dots,w_n)-1,0\}$ to the expressions from Claim 11. For any t with $f_t(w_1,\dots,w_n)\geq 1$, fix an arbitrary $A_t\subseteq[n]$ with $w(A_t)=t$, and let $X_t \coloneqq A_t\setminus[k]$. Note that $h(X_t,t)\geq 1$. Then, by Lemma 8, every $Y\subseteq[n]\setminus[k]$ with $h(Y,t)\geq 1$ must satisfy $(X_t,Y)\in\mathcal{C}$. Hence,

$$\begin{aligned} f_t(w_1,\dots,w_n)-1 &= -1 + \sum_{Y\subseteq[n]\setminus[k]:h(Y,t)\geq 1} h(Y,t) && \text{(by definition of } f_t(\cdot) \text{ and } h(\cdot)) \\ &\leq -1 + \sum_{Y:(X_t,Y)\in\mathcal{C}} h(Y,t) \\ &= (h(X_t,t)-1) + \sum_{Y:(X_t,Y)\in\mathcal{C},Y\neq X_t} h(Y,t) \\ &\leq (h(X_t,t)-1) + \sum_{Y:(X_t,Y)\in\mathcal{C},Y\neq X_t} (h(X_t,t)+h(Y,t)-1) \\ &\leq \sum_{Y:(X_t,Y)\in\mathcal{C}} g(X_t,Y,t) \\ &\leq \sum_{(X,Y)\in\mathcal{C}} g(X,Y,t). \end{aligned}$$

This means that for all $t\in\mathbb{Z}$,

$$\max\{f_t(w_1,\dots,w_n)-1,0\} \leq \sum_{(X,Y)\in\mathcal{C}} g(X,Y,t).$$

By summing over all $t\in\mathbb{Z}$ and substituting using Claim 11, we get

$$F(w_1,\dots,w_n) \leq \sum_{(X,Y)\in\mathcal{C}} F(W_{X,Y}).$$

Then, by averaging, there exists $(X,Y)\in\mathcal{C}$ such that $F(W_{X,Y})\geq F(w_1,\dots,w_n)/|\mathcal{C}|$ as claimed. ◀

4 A Polynomial-Space Algorithm

In this section we prove Theorem 2. We use the following two lemmas from Jin and Wu [15], which can be viewed as polynomial-space versions of Lemma 5 and Lemma 6 respectively:

▶ **Lemma 12** ([15, Lemma 11]). *Given parameter $1 \leq \Delta \leq 2^n/(3n^2)$, a Pigeonhole Equal Subset Sum instance $(w_1, \ldots, w_n)$ (where $w_i < w_{i+1}$) satisfying $F(w_1, \ldots, w_n) \leq \Delta$ and Equation (1) can be solved deterministically in $\mathrm{poly}(n)$ space and $O^*(\Delta)$ time.*

▶ **Lemma 13** ([15, Lemma 13]). *Given parameter $1 \leq \Delta \leq 2^n$, an Equal Subset Sum instance $(w_1, \ldots, w_n)$ with $F(w_1, \ldots, w_n) \geq \Delta$ can be solved in $O^*(2^{1.5n}/\Delta)$ time and $\mathrm{poly}(n)$ space by a randomized algorithm.*

Again, although [15, Lemma 13] was originally stated for the Pigeonhole Equal Subset Sum problem, it actually works for Equal Subset Sum as well because the proof does not use the pigeonhole promise.

Proof sketch of Theorem 2. We follow the same proof outline of Theorem 1, except that we replace the exponential-space subroutines, Lemma 5 and Lemma 6, by their polynomial-space counterparts, Lemma 12 and Lemma 13, respectively. The resulting algorithm is correct and runs in polynomial space. It only remains to re-analyze the time complexity.

Recall that we can assume the integer $k \in \{0, 1, \ldots, n-1\}$ satisfying $2^k \leq F(w_1, \ldots, w_n) < 2^{k+1}$ (Equation (3)) is known. Let $M \in [1, 2^n/(3n^2)]$ be determined later. If $2^k \leq M$, we choose to use Lemma 12 and solve the given Pigeonhole Equal Subset Sum instance in $O^*(2^{k+1}) = O^*(M)$ time. If $2^k > M$, we follow the main proof of Theorem 1 to reduce the original problem to $\mathrm{poly}(n)$ many Equal Subset Sum instances of size $k+1$ (or k) in which at least one instance has F-value $\Omega^*(2^{2k-n})$, so the total time complexity of applying Lemma 13 becomes $O^*(2^{1.5k}/2^{2k-n}) = O^*(2^{n-0.5k}) \leq O^*(2^n/\sqrt{M})$. By choosing $M = 2^{2n/3}$, the overall time complexity becomes $O^*(M + 2^n/\sqrt{M}) = O^*(2^{2n/3})$. ◀

5 Open Questions

We list a few open questions related to this work.

- Can we improve the $O^*(2^{n/3})$ time complexity for Pigeonhole Equal Subset Sum? Can we improve the $O^*(1.7088^n)$ time complexity for Equal Subset Sum [16]?
- Can we obtain fast *deterministic* algorithms for Pigeonhole Equal Subset Sum? Our algorithm (and Jin and Wu's algorithm [15]) uses Lemma 6 which is based on random subsampling. To the best of our knowledge, the $O^*(2^{n/2})$-time algorithms via meet-in-the-middle ([2] or [15, Section 2]) remain the fastest known deterministic algorithms.
- The authors of [2] proposed a more difficult *Modular version* of Pigeonhole Equal Subset Sum problem, and obtained a $O^*(2^{n/2})$-time algorithm. In this problem, we are given integers $w_1, \ldots, w_n$ and a modulus $m \leq 2^n - 1$, and need to find two distinct subsets $A, B \subseteq [n]$ such that $\sum_{i \in A} w_i \equiv \sum_{i \in B} w_i \pmod{m}$. Can we improve the $O^*(2^{n/2})$ time complexity for Modular Pigeonhole Equal Subset Sum?
- The k-SUM problem can be viewed as the fine-grained or parameterized version of the Subset Sum problem. Analogously, we propose the following Pigeonhole Equal k-SUM problem (for fixed integers $k \geq 2$):

> Pigeonhole Equal k-SUM [19]
> **Input:** k length-n arrays $A_1, \ldots, A_k$ of integers from the range $[0, \frac{n^k-1}{k})$.
> **Output:** two different k-tuples $(i_1, \ldots, i_k), (j_1, \ldots, j_k) \in [n]^k$ such that $A_1[i_1] + \cdots + A_k[i_k] = A_1[j_1] + \cdots + A_k[j_k]$.

This is a total search problem, and the brute force algorithm runs in $O(n^k)$ time. It is easy to improve it to $\widetilde{O}(n^{\lceil k/2 \rceil})$ time, using the standard binary search with meet-in-the-middle (following the same idea as the warm-up Pigeonhole Equal Subset Sum algorithm described in [15, Section 2], except that we replace the $O^*(2^{n/2})$-time subroutine for counting Subset Sum solutions by the analogous $\widetilde{O}(n^{\lceil k/2 \rceil})$-time algorithm for k-SUM). Can we improve the runtime to faster than $\widetilde{O}(n^{\lceil k/2 \rceil})$, say for the $k = 3$ case?

References

1 Shyan Akmal, Lijie Chen, Ce Jin, Malvika Raj, and R. Ryan Williams. Improved merlin-arthur protocols for central problems in fine-grained complexity. *Algorithmica*, 85(8):2395–2426, 2023. doi:10.1007/S00453-023-01102-6.

2 Jonathan Allcock, Yassine Hamoudi, Antoine Joux, Felix Klingelhöfer, and Miklos Santha. Classical and quantum algorithms for variants of subset-sum via dynamic programming. In *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.6.

3 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Subset sum in the absence of concentration. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 48–61. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.48.

4 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Dense subset sum may be the hardest. In *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.13.

5 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Sharper upper bounds for unbalanced uniquely decodable code pairs. *IEEE Trans. Inf. Theory*, 64(2):1368–1373, 2018. doi:10.1109/TIT.2017.2688378.

6 Frank Ban, Kamal Jain, Christos H. Papadimitriou, Christos-Alexandros Psomas, and Aviad Rubinstein. Reductions in PPP. *Inf. Process. Lett.*, 145:48–52, 2019. doi:10.1016/j.ipl.2018.12.009.

7 Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas. Faster space-efficient algorithms for subset sum, k-sum, and related problems. *SIAM J. Comput.*, 47(5):1755–1777, 2018. doi:10.1137/17M1158203.

8 Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In *Advances in Cryptology – EUROCRYPT 2011 – 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011. doi:10.1007/978-3-642-20465-4_21.

9 Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In *Advances in Cryptology – ASIACRYPT 2020 – 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 633–666. Springer, 2020. doi:10.1007/978-3-030-64834-3_22.

10 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. An improved pseudopolynomial time algorithm for subset sum. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2202–2216. IEEE, 2024. doi:10.1109/FOCS61266.2024.00129.

11 Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio. Average-case subset balancing problems. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 – 12, 2022*, pages 743–778. SIAM, 2022. doi:10.1137/1.9781611977073.33.

12 Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio. Subset sum in time $2^{n/2}$ / poly(n). In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPIcs*, pages 39:1–39:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.39.

13 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823.

14 Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *Advances in Cryptology – EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 – June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010. doi:10.1007/978-3-642-13190-5_12.

15 Ce Jin and Hongxun Wu. A faster algorithm for pigeonhole equal sums. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 94:1–94:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.94.

16 Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Węgrzycki. Equal-subset-sum faster than the meet-in-the-middle. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 73:1–73:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.73.

17 Jesper Nederlof. A short note on merlin-arthur protocols for subset sum. *Inf. Process. Lett.*, 118:15–16, 2017. doi:10.1016/j.ipl.2016.09.002.

18 Jesper Nederlof and Karol Węgrzycki. Improving Schroeppel and Shamir's algorithm for subset sum via orthogonal vectors. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1670–1683. ACM, 2021. doi:10.1145/3406325.3451024.

19 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.

20 Tim Randolph. *Exact and Parameterized Algorithms for Subset Sum Problems*. PhD thesis, Columbia University, 2024. doi:10.7916/baym-5m55.

21 Katerina Sotiraki, Manolis Zampetakis, and Giorgos Zirdelis. PPP-completeness with connections to cryptography. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 148–158. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00023.

22 Henk C. A. van Tilborg. An upper bound for codes in a two-access binary erasure channel (corresp.). *IEEE Trans. Inf. Theory*, 24(1):112–116, 1978. doi:10.1109/TIT.1978.1055814.

23 Gerhard J. Woeginger. Open problems around exact algorithms. *Discret. Appl. Math.*, 156(3):397–405, 2008. doi:10.1016/J.DAM.2007.03.023.

24 Gerhard J. Woeginger and Zhongliang Yu. On the equal-subset-sum problem. *Inf. Process. Lett.*, 42(6):299–302, 1992. doi:10.1016/0020-0190(92)90226-L.

25 Stan Zhang. *Pigeonhole Equal Subset Sum in $O^*(2^{n/3})$*. Master's thesis, Massachusetts Institute of Technology, 2024. URL: https://hdl.handle.net/1721.1/156830.

A Proof of Lemma 4 by Jin and Wu

▶ **Lemma 4** ([15, Equation (8)]). *Suppose positive integers $w_1 < w_2 < \cdots < w_n$ satisfy $w([n]) < 2^n - 1$ and Equation (1). Then, for all $i \in [n]$,*

$$-i \cdot F(w_1, \ldots, w_n) \leq w_i - 2^{i-1} \leq F(w_1, \ldots, w_n). \tag{2}$$

The following proof of Lemma 4 was given by [15].

Proof. (Jin and Wu [15]) Recall our notations $w(A) = \sum_{i\in A} w_i$, $f_t = |\{A \subseteq [n] : w(A) = t\}|$. We abbreviate $F = F(w_1, \ldots, w_n) = \sum_{t\in\mathbb{Z}} \max\{f_t - 1, 0\}$. Recall our assumption from Equation (1):

$$w([i]) \geq 2^i - 1 \text{ for all } i \in [n-1]. \tag{1}$$

Since $w([n]) < 2^n - 1$, we know $f_t = 0$ for all $t \notin [0, 2^n - 1]$, and $\sum_{0\leq t<2^n} f_t = 2^n$. Then,

$$F = \sum_{0\leq t<2^n} \max\{f_t - 1, 0\} = \sum_{0\leq t<2^n} (f_t - \mathbf{1}[f_t \geq 1]) = 2^n - \sum_{0\leq t<2^n} \mathbf{1}[f_t \geq 1],$$

and thus

$$F = |\{0 \leq t < 2^n : f_t = 0\}|. \tag{7}$$

Since $f_t = 0$ for all $w([n]) < t < 2^n$, from Equation (7) we know $F \geq 2^n - 1 - w([n])$, and hence $w([n]) \geq 2^n - 1 - F$. Combined with Equation (1) for $i \in [n-1]$, we get

$$w([i]) \geq 2^i - 1 - F \text{ for all } i \in [n]. \tag{8}$$

Now we have the following claim:

▷ Claim 14. For all $i \in [n]$,

$$w_i \leq 2^{i-1} + F. \tag{9}$$

Summing Equation (9) over i gives

$$w([i]) \leq 2^i - 1 + iF \tag{10}$$

for all $i \in [n]$.

Proof of Claim 14. Fix $i \in [n]$. Let M be the number of subsets $S \subseteq [n]$ with $w(S) < w_i$. Since $w_i < w_{i+1} < \cdots < w_n$, any such S must be contained in $[i-1]$, and thus $M \leq 2^{i-1}$. On the other hand, $M = \sum_{t=0}^{w_i-1} f_t \geq w_i - |\{0 \leq t < w_i : f_t = 0\}| \geq w_i - F$ by Equation (7). Hence, $w_i \leq M + F \leq 2^{i-1} + F$. ◁

Comparing Equation (8) with Equation (10) gives

$$w_i = w([i]) - w([i-1]) \geq (2^i - 1 - F) - (2^{i-1} - 1 + (i-1)F) = 2^{i-1} - iF.$$

Together with Equation (9) we get the claimed bound

$$w_i - 2^{i-1} \in [-iF, F]$$

for all $i \in [n]$. ◀

Multicut Problems in Almost-Planar Graphs: the Dependency of Complexity on the Demand Pattern

Florian Hörsch
CISPA Helmholtz Center for Information Security, Saarbrucken, Germany

Dániel Marx
CISPA Helmholtz Center for Information Security, Saarbrückem, Germany

Abstract

Given a graph G, a set T of terminal vertices, and a demand graph H on T, the Multicut problem asks for a set of edges of minimum weight that separates the pairs of terminals specified by the edges of H. The Multicut problem can be solved in polynomial time if the number of terminals and the genus of the graph is bounded (Colin de Verdière [*Algorithmica, 2017*]). Restricting the possible demand graphs in the input leads to special cases of Multicut whose complexity might be different from the general problem.

Focke et al. [SoCG 2024] systematically characterized which special cases of Multicut are fixed-parameter tractable parameterized by the number of terminals on planar graphs. Moreover, extending these results beyond planar graphs, they precisely determined how the parameter genus influences the complexity and presented partial results of this form for graphs that can be made planar by the deletion of π edges. Continuing this line of work, we complete the picture on how this parameter π influences the complexity of different special cases and precisely determine the influence of the crossing number, another parameter measuring closeness to planarity.

Formally, let $\mathcal{H}$ be any class of graphs (satisfying a mild closure property) and let Multicut($\mathcal{H}$) be the special case when the demand graph H is in $\mathcal{H}$. Our first main result is showing that if $\mathcal{H}$ has the combinatorial property of having *bounded distance to extended bicliques*, then Multicut($\mathcal{H}$) on *unweighted* graphs is FPT parameterized by the number t of terminals and π. For the case when $\mathcal{H}$ does not have this combinatorial property, Focke et al. [SoCG 2024] showed that $O(\sqrt{t})$ is essentially the best possible exponent of the running time; together with our result, this gives a complete understanding of how the parameter π influences complexity on unweighted graphs. Our second main result is giving an algorithm whose existence shows that the parameter crossing number behaves analogously if we consider Multicut($\mathcal{H}$) on *weighted* graphs.

2012 ACM Subject Classification Mathematics of computing → Graphs and surfaces; Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases MultiCut, Multiway Cut, Parameterized Complexity, Tight Bounds, Embedded Graph, Planar Graph, Crossing Number

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.87

Related Version *Full Version*: `https://arxiv.org/abs/2504.21624` [9]

Acknowledgements We want to thank Jacob Focke and Shaohua Li for the helpful discussions.

1 Introduction

Most NP-hard problems remain NP-hard when restricted to planar graphs, unless they become trivial or irrelevant on planar graphs. There are very few NP-hard problems that are polynomial-time solvable on planar graphs for some nontrivial and interesting reason. One such case is the Multiway Cut problem: given a graph G and a set $T \subseteq V(G)$ of *terminals*, the task is to find a *multiway cut* S of minimum weight, that is, a set $S \subseteq E(G)$


© Florian Hörsch and Dániel Marx;
licensed under Creative Commons License CC-BY 4.0

such that every component of $G \setminus S$ contains at most one terminal. Dahlhaus et al. [3] showed that Multiway Cut in general graphs is NP-hard even for three terminals, while it is polynomial-time solvable on planar graphs for every fixed number $|T|$ of terminals. The exponent of n in the algorithm of Dahlhaus et al. [3] is $O(|T|)$. Later, the dependence was improved to $O(\sqrt{|T|})$ [2, 10], which was shown to be optimal under the Exponential-Time Hypothesis (ETH) [1, 11]. The results were extended to bounded-genus graphs, again with an essentially best possible form of the exponent [1, 2].

Multicut is a generalization of Multiway Cut where not all terminals need to be separated from each other, but the input specifies which pairs of terminals need to be separated (and we do not care whether other pairs are separated or not).

Multicut	
Input:	A weighted graph G together with a set $T \subseteq V(G)$ of *terminals*; and a *demand graph* H with $V(H) = T$.
Output:	A minimum-weight set $S \subseteq E(G)$ such that u and v are in distinct components of $G \setminus S$ whenever uv is an edge of H.

Let us observe that Multiway Cut is the special case of Multicut when H is a complete graph on T. Colin de Verdière [2] presented an algorithm for Multicut where the exponent of the running time is $O(\sqrt{|T|})$, similarly to Multiway Cut. However, it is no longer clear anymore that this algorithm is a tight, optimal result and cannot be improved in some cases: it could be that there are special cases, special type of demand graphs where significantly better algorithms are possible. For example, if H is a biclique (complete bipartite graph), then Multicut can be solved in polynomial time by a reduction to a minimum weight $s - t$ cut problem. A very natural question to consider is the case when H is a complete 3-partite graph. This special case is the Group 3-Terminal Cut problem: given a graph G with three sets of terminals T_1, T_2, T_3, the task is to find a set S of edges of minimum total weight such that $G \setminus S$ has no path between any vertex of T_i and any vertex of T_j for all distinct $i, j \in [3]$. While a general Multicut algorithm solves Group 3-Terminal Cut with the exponent of the running time being the square root of the number of terminals, the lower bounds for Multiway Cut do not imply that this dependence is optimal.

Focke et al. [5] systematically analyzed which types of demand graphs make the problem more tractable. The question was explored in the following setting. Let $\mathcal{H}$ be a class of graphs (i.e., a set of graphs closed under isomorphism). Then we denote by Multicut($\mathcal{H}$) the special case of Multicut that contains only instances (G, H) where $H \in \mathcal{H}$. For example, if $\mathcal{H}$ is the class of cliques, then Multicut($\mathcal{H}$) is exactly Multiway Cut.

Ideally, for every class $\mathcal{H}$ of graphs, we would like to understand the best possible running time of Multicut($\mathcal{H}$) on planar graphs. In order to make the question more robust, Focke et al. [5] considered only classes $\mathcal{H}$ satisfying a mild closure property. We say that H' is a *projection* of H if H' can be obtained from H by deleting vertices and identifying pairs of independent vertices. Moreover, a class of graphs $\mathcal{H}$ is *projection-closed* if, whenever $H \in \mathcal{H}$ and H' is a projection of H, then $H' \in \mathcal{H}$ also holds. The intuition is that an instance having demand graph H' can be easily simulated by an instance having demand graph H, thus it is reasonable to assume we only consider classes of demand graphs closed under this operation.

Let us consider some easy cases first. As we mentioned before, the Multicut problem can be solved in polynomial time by a reduction to $s - t$ minimum cut if the demand graph is a biclique. Furthermore, if H has only two edges, then a similar reduction is known [8, 12]. Isolated vertices of H clearly do not play any role in the problem. We say that H is a *trivial pattern* if, after removing isolated vertices, it is either a biclique or has at most two edges.

An *extended biclique* is a graph that consists of a complete bipartite graph together with a set of isolated vertices. Let μ be the minimum number of vertices that need to be deleted to make a graph H an extended biclique. Then we say that H has distance μ to extended bicliques. Moreover, we say that a graph class $\mathcal{H}$ has *bounded distance to extended bicliques* if there is a constant μ such that every $H \in \mathcal{H}$ has distance at most μ to extended bicliques.

For planar graphs, Focke et al. [5] give the following characterization:

▶ **Theorem 1.1** ([5]). *Let $\mathcal{H}$ be a computable projection-closed class of graphs. Then the following holds for* Multicut($\mathcal{H}$) *on planar graphs.*

1. *If $\mathcal{H}$ has bounded distance to extended bicliques, then there is an $f(t)n^{O(1)}$ time algorithm.*
2. *Otherwise,*
   a. *There is an $f(t)n^{O(\sqrt{t})}$ time algorithm.*
   b. *Assuming* ETH*, there is a universal constant $\alpha > 0$ such that for any fixed choice of $t \geq 3$, there is no $O(n^{\alpha\sqrt{t}})$ algorithm, even when restricted to unweighted instances with at most t terminals.*

That is, depending on whether $\mathcal{H}$ has bounded distance to extended bicliques, the number t of terminals has to appear in the exponent. Focke et al. [5] precisely characterized how the complexity changes if we move from planar graphs to bounded-genus graphs. As we can see, the parameter genus can appear in the exponent in different ways, depending on $\mathcal{H}$.

▶ **Theorem 1.2** ([5]). *Let $\mathcal{H}$ be a computable projection-closed class of graphs. Then the following holds for* Multicut($\mathcal{H}$).

1. *If every graph in $\mathcal{H}$ is a trivial pattern, then there is a polynomial time algorithm.*
2. *Otherwise, if $\mathcal{H}$ has bounded distance to extended bicliques, then*
   a. *There is an $f(g,t)n^{O(g)}$ time algorithm.*
   b. *Assuming* ETH*, there is a universal constant $\alpha > 0$ such that for any fixed choice of $\vec{g} \geq 0$, there is no $O(n^{\alpha(\vec{g}+1)/\log(\vec{g}+2)})$ algorithm, even when restricted to unweighted instances with orientable genus at most $\vec{g}$ and $t = 3$ terminals.*
3. *Otherwise,*
   a. *There is an $f(g,t)n^{O(\sqrt{g^2+gt+t})}$ time algorithm.*
   b. *Assuming* ETH*, there is a universal constant $\alpha > 0$ such that for any fixed choice of $\vec{g} \geq 0$ and $t \geq 3$, there is no $O(n^{\alpha\sqrt{\vec{g}^2+\vec{g}t+t}/\log(\vec{g}+t)})$ algorithm, even when restricted to unweighted instances with orientable genus at most $\vec{g}$ and at most t terminals.*

Focke et al. [5] also considered a much more restricted extension of planar graphs than bounded-genus graphs: graphs that can be made planar by deleting a few edges. We use $\mathsf{planar} + \pi e$ for the class of graphs that can be made planar by removing at most π edges. Given an instance (G, H), we use π for the smallest integer such that G is in $\mathsf{planar} + \pi e$.

Observe that in Theorem 1.2, the algorithmic results are stated for the edge-weighted version, while the lower bounds are stated for the unweighted version. In fact, edge-weights do not make much difference in Theorem 1.2, as polynomial edge weights can be simulated by parallel edges without changing the genus of the graphs. However, this simulation can easily change the number π of edges that needs to be deleted to make the graph planar. This means that if we consider the influence of the parameter π on the running time, the edge-weighted and the unweighted problem may behave differently. Focke et al. [5] characterized the influence of this parameter in the edge-weighted case:

▶ **Theorem 1.3** ([5]). *Let $\mathcal{H}$ be a computable projection-closed class of graphs. Then the following holds for edge-weighted* Multicut($\mathcal{H}$).

1. *If every graph in $\mathcal{H}$ is a trivial pattern, then there is a polynomial-time algorithm.*
2. *Otherwise, if $\mathcal{H}$ has bounded distance to extended bicliques, then*
   a. *There is an $f(\pi,t)n^{O(\sqrt{\pi})}$ time algorithm.*
   b. *Assuming* ETH*, there is a universal constant $\alpha > 0$ such that for any fixed choice of $\pi \geq 0$, there is no $O(n^{\alpha\sqrt{\pi}})$ algorithm, even when restricted to instances in* planar $+ \pi e$ *with $t = 3$ terminals.*
3. *Otherwise,*
   a. *There is an $f(\pi,t)n^{O(\sqrt{\pi+t})}$ algorithm.*
   b. *Assuming* ETH*, there is a universal constant $\alpha > 0$ such that for any fixed choice of $\pi \geq 0$ and $t \geq 3$, there is no $O(n^{\alpha\sqrt{\pi+t}})$ algorithm, even when restricted to* planar $+ \pi e$ *instances with at most t terminals.*

Observe that π has a different form of influence on the exponent compared to genus: it is $O(\sqrt{\pi})$ instead of $O(g)$ in the case of bounded distance to extended bicliques, and $O(\sqrt{\pi+t})$ instead of $O(\sqrt{g^2+gt+t})$ in the unbounded distance case. Intuitively, having π extra edges is a much more restricted extension of planar graphs than having genus g, and consequently, the parameter π has a much weaker influence on the exponent than genus has.

For unweighted graphs, Focke et al. [5] showed that the problem is easier than the edge-weighted version and the parameter π can be removed from the exponent.

▶ **Theorem 1.4** ([5]). *Unweighted* Multicut *can be solved in time $f(\pi,t)n^{O(\sqrt{t})}$.*

However, this result leaves open the question what happens in unweighted Multicut($\mathcal{H}$) if $\mathcal{H}$ has bounded distance to extended bicliques. Based on Theorems 1.2 and 1.3, one would expect that it is possible to remove the number t of terminals from the exponent as well.

Our contributions

First, we resolve the remaining question about unweighted Multicut($\mathcal{H}$) and show that if $\mathcal{H}$ has bounded distance to extended bicliques, then both t and π can be removed from the exponent. Recall that μ is the number of vertices needed to be deleted to make the demand graph an extended biclique. Our first main result is the following algorithm.

▶ **Theorem 1.5.** *Unweighted* Multicut *can be solved in time $f(\pi,t)n^{O(\sqrt{\mu})}$.*

Observe that, in contrast to Theorem 1.4, only μ is in the exponent of the running time of the algorithm of Theorem 1.5. As $\mu \leq t$, this algorithm gives an independent proof of Theorem 1.4 with a very different proof technique. With Theorem 1.5 at hand, we can complete the picture for the complexity of unweighted Multicut($\mathcal{H}$):

▶ **Theorem 1.6.** *Let $\mathcal{H}$ be a computable projection-closed class of graphs. Then the following holds for unweighted* Multicut($\mathcal{H}$).

1. *If $\mathcal{H}$ has bounded distance to extended bicliques, then there is an $f(\pi,t)n^{O(1)}$-time algorithm.*
2. *Otherwise,*
   a. *There is an $f(\pi,t)n^{O(\sqrt{t})}$ algorithm.*
   b. *Assuming* ETH*, there is a universal constant $\alpha > 0$ such that for any fixed choice of $t \geq 3$, there is no $O(n^{\alpha\sqrt{t}})$-time algorithm, even when restricted to planar instances with at most t terminals.*

We consider yet another, even more restricted parameter measuring the distance from planar graphs: the *crossing number*, the minimum number of crossings needed when drawing the graph in the plane. Clearly, if a graph has a drawing with cr crossings, then we can remove a set of at most cr edges to make the graph planar. Thus the crossing number cr is always at least π, that is, an algorithm parameterized by π immediately gives an algorithm parameterized by cr.

Again, there is no straightforward reduction from the edge-weighted version to the unweighted version without increasing the crossing number. Thus both versions of the problem when parameterizing by crossing number are interesting. Clearly, the unweighted version is no harder than the edge-weighted one. Moreover, both versions parameterized by crossing number are no harder than their respective versions parameterized by π. But which of these versions are *strictly* harder than the others?

It turns out that, when parameterizing by crossing number, the edge weights do not change the hardness of the problem and both the edge-weighted and the unweighted versions parameterized by cr have the same complexity as the *unweighted* version parameterized by π. The following algorithm is our main result on the crossing number parameterization.

▶ **Theorem 1.7.** *Edge-weighted* Multicut *can be solved in time* $f(cr,t)n^{O(\sqrt{\mu})}$.

Together with Theorem 1.1, it implies the following complete classification for the complexity of Multicut($\mathcal{H}$) when parameterizing by cr:

▶ **Theorem 1.8.** *Let $\mathcal{H}$ be a computable projection-closed class of graphs. Then the following holds for edge-weighted* Multicut($\mathcal{H}$).

1. *If $\mathcal{H}$ has bounded distance to extended bicliques, then there is an $f(cr,t)n^{O(1)}$-time algorithm.*
2. *Otherwise,*
   a. *There is an $f(cr,t)n^{O(\sqrt{t})}$ algorithm.*
   b. *Assuming* ETH*, there is a universal constant $\alpha > 0$ such that for any fixed choice of $t \geq 3$, there is no $O(n^{\alpha\sqrt{t}})$-time algorithm, even when restricted to unweighted planar instances with at most t terminals.*

Note that the algorithmic results of Theorems 1.6 and 1.8 are incomparable: Theorem 1.6 considers only unweighted graphs, while Theorem 1.8 considers edge-weighted graphs, but under parameterization by a potentially much larger parameter.

Theorems 1.2–1.8 complete the picture of how different parameters measuring closeness to planarity influence the running time of Multicut($\mathcal{H}$) for different classes $\mathcal{H}$ (see Figure 1). Let us observe that the exact form of the influence is very different for the different parameters. As we explain below, the treewidth of the so-called multicut dual of the optimum solution is the main factor determining the exponent of the running time [2, 5]. The exponent $O(\sqrt{t})$ for planar graphs comes from the fact that a planar graph with $O(t)$ vertices (or faces) has treewidth $O(\sqrt{t})$. Intuitively, for nonplanar graphs, the situation changes as follows:

- **Genus.** If the genus is large, then a graph with t vertices can have treewidth larger than $O(\sqrt{t})$. The maximum possible treewidth is given by the (nonobvious) formula $O(\sqrt{g^2+gt+t})$. Therefore, the maximum treewidth of the multicut dual is $O(\sqrt{g^2+gt+t})$, giving a bound on the exponent of the running time.
- **Edge-weighted** planar $+\,\pi e$**.** With extra edges having infinite (or very large) weight, we can simulate multiple terminals at different parts of a planar graph. More precisely, a planar instance of Group 3-Terminal Cut with t terminals in each of the three groups can be reduced to an instance of Multiway Cut with three terminals if we connect

	genus	weighted planar $+ \pi e$	unweighted planar $+ \pi e$	crossing number
$\mathcal{H}$ with bounded distance to extended bicliques	$f(g,t)n^{O(g)}$	$f(\pi,t)n^{O(\sqrt{\pi})}$	$f(\pi,t)n^{O(1)}$ (*)	$f(cr,t)n^{O(1)}$ (*)
$\mathcal{H}$ with unbounded distance to extended bicliques	$f(g,t)n^{O(\sqrt{g^2+gt+t})}$	$f(\pi,t)n^{O(\sqrt{\pi+t})}$	$f(\pi,t)n^{O(\sqrt{t})}$	$f(cr,t)n^{O(\sqrt{t})}$ (*)

Figure 1 The running time of the algorithms for Multicut($\mathcal{H}$) with the different parameterizations. The new results of the paper are marked by (*). Note that, for every fixed projection-closed class $\mathcal{H}$, the exponents of the running time are tight (up to logarithmic factors).

the terminals in each group with a total of $\pi = 3(t-1)$ infinite-weight edges, creating a planar $+ \pi e$ instance. Thus these extra edges influence the running time exactly as extra terminals would do.

- **Unweighted planar $+ \pi e$.** We can always try to solve the problem by putting the π extra edges into the solution, removing them from G, and then solving the resulting planar instance. Of course, this might not always lead to an optimum solution, but can be up to π edges larger. However, this argument allows us to say that if the solution contains a cut that is "inefficient" in the sense that it is larger then the minimum cut by more than π, then the cut can be replaced by a minimum cut and the π extra edges, resulting in a strictly smaller solution. Using these type of arguments, the problem can be reduced to a planar setting where the number of faces of the multicut dual does not depend on π.
- **Crossing number.** Each crossing can increase the number of vertices of the multicut dual by one, thus the number of vertices and the treewidth of the multicut dual can be $O(t+cr)$ and $O(\sqrt{t+cr})$, respectively. However, all these extra vertices of the multicut dual are at fixed places determined by the crossing. As pointed out by Focke et al. [6] it is sufficient to consider the treewidth of the multicut dual *after removing all the vertices whose locations are fixed* (see Theorem 2.3). Indeed, after removing these vertices, the treewidth of the multicut dual does not depend on the crossing number.

Some remarks on two further notion of almost-planar graphs can be found in the full version.

2 Preliminaries

Some basic notation on sets, graphs, and multicuts can be found in the full version. We now give some less well-known, problem-specific definitions.

An *extended biclique decomposition* of a graph H is a partition (B_1, B_2, I, X) of $V(H)$ such that all vertices of I are isolated in $H - X$, $E(H)$ contains the edge t_1t_2 for all $t_1 \in B_1$ and $t_2 \in B_2$, and $H[B_i]$ is an edgeless graph for $i \in [2]$. We use $\mathcal{H}_\mu$ for the set of graphs H that admit an extended biclique decomposition (B_1, B_2, I, X) with $|X| \leq \mu$.

We now review the concept of so-called multicut duals, which will play a crucial role throughout the article. Recall that an instance (G, H) of Multicut is *planar* if G is given in the form of a graph cellularly embedded in the plane. Given a graph C that is embedded in the plane and in general position with G, we denote by $e_G(C)$ those edges of G that are crossed by at least one edge of C. The *weight* $w(C)$ of C is defined to be $w(e_G(C))$

where w is the weight function associated with G. A *multicut dual* for (G, H) is a graph C embedded in the plane that is in general position with G and has the property that, for every $t_1, t_2 \in V(H)$ with $t_1t_2 \in E(H)$, the terminals t_1 and t_2 are contained in different faces of C.

The following result is an immediate consequence of Lemmas 2.3 and 2.4 in [6].

▶ **Lemma 2.1.** *Let (G, H) be a planar instance of* Multicut *and C a minimum weight multicut dual for (G, H). Then $e_G(C)$ is a minimum weight multicut for (G, H).*

In order to make use of the topology of certain multicut duals, we need two results related to treewidth. The first one of the following two results provides a sufficient criterion for the treewidth of certain graphs being bounded and the second one allows to exploit bounded treewidth algorithmically.

The following result can be obtained from Theorem 3.2 in [4] and the fact that the treewidth of any graph can only be larger than its branchwidth by a constant factor.

▶ **Proposition 2.2.** *Let C be a planar graph that admits a 5-dominating set of size k for some positive integer k. Then $\mathrm{tw}(C) = O(\sqrt{k})$.*

An *extended* planar instance (G, H, F^*) of Multicut is a planar instance (G, H) of Multicut together with a set F^* of faces of G which is given with the input. Given a set F^* of faces of G and a multicut dual C for (G, H), let $C - F^*$ be the graph obtained from C by removing every vertex that is in a face of F^* and removing every edge that intersects a face of F^*. The following result is the restriction of Theorem 1.11 in [6] to the planar setting.

▶ **Theorem 2.3.** *Let (G, H, F^*) be an extended planar instance of* Multicut *such that every subcubic inclusionwise minimal minimum-weight multicut dual C satisfies $\mathrm{tw}(C - F^*) \leq \beta$. Then an optimum solution of (G, H) can be found in time $f(|F^*|, t)n^{O(\beta)}$.*

3 Edge-deletion distance to planarity

In this section, we give an overview of the proof of the following result, whose complete proof can be found in the full version.

▶ **Theorem 1.5.** *Unweighted* Multicut *can be solved in time $f(\pi, t)n^{O(\sqrt{\mu})}$.*

Throughout this section, we say that an instance (G, H) of Multicut is planar $+ \pi e$ if a set $E_\pi \subseteq E(G)$ is given with the instance such that $|E_\pi| = \pi$ and $G \setminus E_\pi$ is planar. We further use T for $V(H)$, W for $V(E_\pi)$, and G_0 for $G \setminus E_\pi$. Also, for some $S \subseteq E(G)$, we use S_0 for $S \setminus E_\pi$. We say that (G, H) is *connected* if G is connected.

The overall algorithmic strategy for Theorem 1.5 is to run an algorithm on the given instance that either produces a minimum multicut for the instance or computes a collection of instances for which the parameters in consideration are smaller and such that from minimum multicuts of all those instances, a minimum multicut for the original instance can be computed. We need the following definitions.

Given instances (G, H) and (G', H') of Multicut, (G', H') is a *subinstance* of (G, H) if G' is a labelled subgraph of G and H' is a labelled induced sungraph of H. Next, an *extended subinstance* of (G, H) consists of a set $S' \subseteq E(G)$ and a subinstance (G', H') of (G, H) with $E(G') \cap S' = \emptyset$. When we say that an extended subinstance $(S', (G', H'))$ has certain properties which can only be associated to an instance of Multicut, we refer to (G', H'). Next, we say that an extended subinstance $(S', (G', H'))$ of (G, H) is *optimumbound* for (G, H) if $S' \cup S''$ is a minimum multicut for (G, H) for every minimum multicut S'' for (G', H').

The main technical difficulty is to prove the following result, which shows that the above described reduction to smaller instances exists for connected instances. With this result at hand, Theorem 1.5 follows readily. In particular, if disconnected instances are created, they can easily be reduced to instances corresponding to the components of the graph. Further observe that μ cannot increase when moving to a subinstance as the new demand graph is an induced subgraph of the previous one.

▶ **Lemma 3.1.** *Let (G, H) be a connected* planar $+ \pi e$ *instance of* Multicut. *Then, in $f(\pi, t)n^{O(\sqrt{\mu})}$, we can run an algorithm that returns a set $S \subseteq E(G)$ and a collection $(S_i, (G_i, H_i))_{i\in[k]}$ of extended subinstances of (G, H) for some $k = f(\pi, t)$ such that for all $i \in [k]$, we have that $\pi(G_i) + |V(H_i)| < \pi(G) + |V(H)|$ holds and either S is a minimum multicut for (G, H) or there exists some $i \in [k]$ such that $(S_i, (G_i, H_i))$ is optimumbound for (G, H).*

We now give an overview of the proof of Lemma 3.1. Let (G, H) be a planar $+ \pi e$ instance of Multicut. The idea is to have a branching algorithm that gradually guesses more and more crucial information about a minimum multicut for (G', H') while the number of directions it branches into remains bounded. The current status of this procedure is displayed by a subpartition of the vertices of $V(G)$ with some extra properties. More formally, a *state* is a subpartition $\mathcal{P}$ of $V(G)$ with $W \cup T \subseteq \bigcup \mathcal{P}$ and $Y \cap (W \cup T) \neq \emptyset$ for all $Y \in \mathcal{P}$. Given a certain state, the objective is to understand whether there exists a minimum multicut S for (G, H) such that the structure of the components of $G_0 \setminus S$ reflects the structure of $\mathcal{P}$. In order to make this more formal, we need the following definitions: Given a set V and two subpartitions $\mathcal{P}^1$ and $\mathcal{P}^2$ of V, we say that $\mathcal{P}^2$ *extends* $\mathcal{P}^1$ if $|\mathcal{P}^1| = |\mathcal{P}^2|$ and for every $Y \in \mathcal{P}^1$, there exists some $\overline{Y} \in \mathcal{P}^2$ with $Y \subseteq \overline{Y}$ such that for any distinct $Y_1, Y_2 \in \mathcal{P}^1$, we have $\overline{Y_1} \neq \overline{Y_2}$. Given a set $S \subseteq E(G)$, we use S_0 for $S \cap E(G_0)$ and $\mathcal{K}(G_0 \setminus S_0)$ for the unique partition of $V(G)$ in which two vertices are in the same partition class if and only if they are in the same component of $G_0 \setminus S_0$. For a state $\mathcal{P}$, we say that a multicut S for (G, H) *respects* $\mathcal{P}$ if $\mathcal{K}(G_0 \setminus S_0)$ extends $\mathcal{P}$ and we say that a state is *valid* if there exists a minimum multicut for (G, H) respecting it. Further, $\mathcal{P}$ is *maximum valid* if $|\mathcal{P}|$ is maximum among all valid states.

The idea of our algorithm now consists of considering a valid state, choosing a vertex that is not contained in any class of the state and making the classes larger by adding this vertex to one of them. As we do not know which class this vertex should be added to, we try all classes of the state and branch in these directions. The difficulty is now to carefully choose the vertex in consideration and make some additional modifications on the states so that this procedure terminates sufficiently fast.

In order to be able to explain how this works, we first need some more definitions. Given disjoint sets $Y_1, Y_2 \subseteq V(G)$, we use $\lambda_{G_0}(Y_1, Y_2)$ to denote the size of the smallest set of edges in $E(G_0)$ whose deletion from G_0 results in a graph in which there exists no component containing both a vertex of Y_1 and a vertex of Y_2. Next, for a state $\mathcal{P}$ and some $Y \in \mathcal{P}$, we define $\alpha_{G,\mathcal{P}}(Y) = \lambda_{G_0}(Y, \bigcup \mathcal{P} \setminus Y) - \lambda_{G_0}(Y \cap (W \cup T), W \cup T \setminus Y)$. Intuitively speaking, this measure describes how much more costly it is to separate the class from the other classes of the state in comparison to only separating the corresponding elements of $W \cup T$. Given a state $\mathcal{P}$, we now classify its classes according to their value of α and the value of α of their neighboring classes. More formally, given a state $\mathcal{P}$ and some $Y \in \mathcal{P}$, we say that Y is *fat* in $\mathcal{P}$ if $\alpha_{G_0,\mathcal{P}}(Y) \geq \pi(t + 1) + 1$. For some $Y \in \mathcal{P}$ that is not fat, we say that Y is *fat-neighboring* in $\mathcal{P}$ if there exists some fat $Y' \in \mathcal{P}$, some $u \in Y$, and some $v \in Y'$ such that $uv \in E(G_0)$. Finally, for some $Y \in \mathcal{P}$ which is neither fat nor fat-neighboring in $\mathcal{P}$, we say that Y is *thin* in $\mathcal{P}$. We refer by $\mathcal{P}_{\text{fat}}, \mathcal{P}_{\text{f-n}}$, and $\mathcal{P}_{\text{thin}}$ to the fat, fat-neighboring, and thin classes of $\mathcal{P}$, respectively.

In order to make use of this distinction, we crucially need a structural property of minimum multicuts for (G, H). Namely, the following result shows that, for any minimum multicut S for (G, H) and any vertex set of a component of $G_0 \setminus S_0$, one of two things holds: either the value of α is not too large or a terminal of X is close to this vertex set in a certain sense, where (B_1, B_2, I, X) is an extended biclique decomposition of H. In order to measure this proximity of certain vertex sets with respect to a multicut, we introduce the following notion: Let $S \subseteq E(G)$, and $Y_1, Y_2 \subseteq V(G)$. Then we denote by $\text{dist}_{G_0,S_0}(Y_1, Y_2)$ the smallest integer q such that G_0 contains a Y_1Y_2-path P with $|E(P) \cap S_0| = q$. We are now ready to state the following structural result for minimum multicuts.

▶ **Lemma 3.2.** *Let (G, H) be a* planar $+ \pi e$ *instance of* Multicut*, let S be a minimum multicut for (G, H) and let $Y \in \mathcal{P}_{\text{fat}}$ for $\mathcal{P} = \mathcal{K}(G_0 \setminus S_0)$. Then $\text{dist}_{G_0,S_0}(Y, X) \leq 1$.*

We now explain the strategy of our algorithm. Given a state with an extra property that will be explained later, we choose a vertex that is not contained in any class of the state and has a neighbor in a thin class of the state in G_0. We then guess a partition class which this vertex will be added to and modify the state accordingly. In order to show the functionality of this algorithm, there are two points that need further explanation. First, we need to explain how to conclude when the described modification is no longer possible and second, we need to explain why this procedure terminates sufficiently fast.

We first explain the scenario that we can no longer execute the described modification, so there does not exist any vertex in the graph that is not contained in any class of the state $\mathcal{P}$ in consideration and that has a neighbor in a thin class of $\mathcal{P}$ in G_0. We call such a state *complete*, all other states are called *incomplete*. For complete states, we need to consider two different cases, depending on whether $\mathcal{P}$ contains a thin class.

The case that there exists a thin class Y can be handled with elementary means. Observe that for this thin class, all its neighbors in G_0 are contained in a different class of $\mathcal{P}$. Hence, if $\mathcal{P}$ is valid, then all edges in $\delta_{G_0}(Y)$ are contained in a minimum multicut for (G, H). We can hence delete these edges and solve the remaining instance, which turns out to be significatly smaller in a certain sense. This is subsumed in the following result.

▶ **Lemma 3.3.** *Let (G, H) be a connected* planar$+\pi e$ *instance of* Multicut *and let a complete valid state $\mathcal{P}$ of (G, H) with $\mathcal{P}_{\text{thin}} \neq \emptyset$ be given. Then in polynomial time, we can compute an optimumbound extended subinstance $(S', (G', H'))$ of (G, H) such that $\pi(G') + |V(H')| < \pi(G) + |V(H)|$.*

We now turn to the case that $\mathcal{P}$ does not contain a thin set. In order to handle this case, geometrical arguments on multicut duals will be needed. Namely, we show that if $\mathcal{P}$ is valid and does not contain a thin set, then the problem of finding a minimum multicut for (G, H) can be reduced to solving a planar instance of multicut, which can be solved sufficiently fast due to Theorem 2.3 and an argument on the treewidth of the multicut duals for this instance. Formally, we prove the following result, where $\tau(\mathcal{P})$ denotes the sum of the number of components of $G_0[Y]$ over all $Y \in \mathcal{P}$.

▶ **Lemma 3.4.** *Let (G, H) be a connected* planar $+ \pi e$ *instance of* Multicut *and let a complete maximum valid state $\mathcal{P}$ of (G, H) with $\mathcal{P}_{\text{thin}} = \emptyset$ be given. Then a minimum multicut for (G, H) can be computed in $f(\pi, \tau(\mathcal{P}))n^{O(\sqrt{\mu})}$.*

It remains to show that we can design our procedure so that we reach a complete state sufficiently fast. In order to keep track of the progress of our algorithm, we need a measure of how far advanced a given state is. To this end, given a state $\mathcal{P}$, we first define an integer $\kappa(\mathcal{P}, Y)$ for every $Y \in \mathcal{P}$ by

$$\kappa(\mathcal{P}, Y) = \left\{ \begin{array}{ll} 2(\pi(t+1)+1), & \text{for } Y \in \mathcal{P}_{\text{fat}}, \\ \pi(t+1)+1+\alpha_{G_0,\mathcal{P}}(Y), & \text{for } Y \in \mathcal{P}_{\text{f-n}}, \\ \alpha_{G_0,\mathcal{P}}(Y), & \text{for } Y \in \mathcal{P}_{\text{thin}}. \end{array} \right\}$$

Now, we define $\kappa(\mathcal{P}) = \sum_{Y \in \mathcal{P}} \kappa(\mathcal{P}, Y)$.

Our objective is to make the value of κ increase in each iteration of the algorithm. In order to make this possible, we need that the addition of a vertex to a given set in the state makes the value of α of this set increase. In order for that to hold, we need the classes in the state to be as large as possible in a certain sense. More precisely, given a graph G and two disjoint sets $Y_1, Y_2 \subseteq V(G)$, we say that a set Y_3 is *relevant* for (Y_1, Y_2) in G if $Y_1 \subseteq Y_3, Y_2 \cap Y_3 = \emptyset, d_G(Y_3) = \lambda_G(Y_1, Y_2)$ and Y_3 is maximum among all sets with these properties. Further, we say that a state $\mathcal{P}$ is *relevant* if for all $Y \in \mathcal{P}$, we have that Y is a relevant set for $(Y, \bigcup \mathcal{P} \setminus Y)$. Luckily, the following result whose proof needs some careful reconfiguration arguments, allows us to turn arbitrary states in consideration into relevant ones without dropping any important properties.

▶ **Lemma 3.5.** *Let a connected* planar $+ \pi e$ *instance* (G, H) *of* Multicut *and a state* $\mathcal{P}$ *of* (G, H) *be given. Then, in polynomial time, we can compute a relevant state* $\mathcal{P}'$ *of* (G, H) *such that* $|\mathcal{P}'| = |\mathcal{P}|, \tau(\mathcal{P}') \leq \tau(\mathcal{P})$, $\kappa(\mathcal{P}') \geq \kappa(\mathcal{P})$ *and, if* $\mathcal{P}$ *is maximum valid, then so is* $\mathcal{P}'$.

With this result at hand, we can handle incomplete states. As pointed out before, given an incomplete relevant state $\mathcal{P}$, we choose a vertex $u_0 \in V(G) \setminus \bigcup \mathcal{P}$ that has a neighbor in a thin class Y of $\mathcal{P}$. We now consider all states that are obtained by adding u_0 to one of these partition classes. If u_0 is added to a thin or a fat-neighboring partition class, then, as $\mathcal{P}$ is relevant, the value of α increases for this class and hence κ increases. If u_0 is added to a fat class, then Y becomes fat-neighboring and hence κ also increases. This shows that a complete state can be reached sufficiently fast. In order to apply Lemma 3.4, we also need to make sure that $\tau(\mathcal{P})$ does not increase too fast. However, this follows readily by construction. Our handling of relevant incomplete states is subsumed in the following result.

▶ **Lemma 3.6.** *Let* (G, H) *be a connected* planar $+ \pi e$ *instance of* Multicut *and let a relevant incomplete state* $\mathcal{P}$ *of* (G, H) *be given. Then, in polynomial time, we can compute a collection* $(\mathcal{P}^i)_{i \in [q]}$ *of* q *states of* (G, H) *for some* $q \leq \pi + t$ *such that* $\tau(\mathcal{P}^i) \leq \tau(\mathcal{P}) + 1$ *and* $\kappa(\mathcal{P}^i) > \kappa(\mathcal{P})$ *hold for all* $i \in [q]$ *and if* $\mathcal{P}$ *is maximum valid, then there exists some* $i \in [q]$ *such that* $\mathcal{P}^i$ *is maximum valid.*

The overall strategy for handling Lemma 3.1 is now to apply Lemma 3.5 and Lemma 3.6 until we are left with a collection of complete states. These can then be handled by Lemma 3.3 and Lemma 3.4.

4 Crossing number

In this section, a *drawing* of a graph G consists of a mapping of the vertices of G to points in the plane and a mapping of the edges of G to curves in the plane connecting their endpoints such that all vertices and edges of G are in general position except the required adjacencies and such that in any point not corresponding to a vertex, at most two edges intersect. An *edge crossing* refers to a point that does not correspond to a vertex and in which two edges intersect. Given an instance (G, H) of Multicut we use *cr* for the *crossing number* of a graph G, which is defined to be the smallest integer k such that there exists a drawing of G with exactly k edge crossings. The following is the main result of this section.

▶ **Theorem 1.7.** *Edge-weighted* Multicut *can be solved in time* $f(cr,t)n^{O(\sqrt{\mu})}$.

In order to make our algorithm work, rather than just the crossing number of a graph, we also need to have an optimal drawing available. This can be achieved through the following result due to Grohe [7].

▶ **Proposition 4.1.** *Given a graph* G*, there exists an algorithm that computes a drawing of* G *in the plane minimizing the number of crossings and that runs in time* $f(cr)n^{O(1)}$.

Formally, an instance (G,H) of Multicut is called *crossing-planar* if G is given as a drawing of a graph. Given a crossing-planar instance, we denote by E_{cr} the set of edges in $E(G)$ crossing at least one other edge of $E(G)$ and we set $\overline{cr} = |E_{cr}|$. Observe that $\overline{cr}$ can be arbitrarily large in comparison to cr as $\overline{cr}$ refers to the given drawing and cr to an optimal drawing of the abstract graph G. However, when considering an optimal drawing of G, we have $\overline{cr} \leq 2cr$ as every crossing involves 2 edges.

For our algorithm, it will be convenient to assume that the instances in consideration satisfy a technical extra condition. Namely, a crossing-planar instance (G,H) of Multicut is called *normalized* if every $e \in E_{cr}$ is of infinite weight. Observe that for any normalized crossing-planar instance (G,H) of Multicut, any finite multicut S of (G,H) and any $e \in E_{cr}$, we have that $V(e)$ is contained in one component of $G \setminus S$. Our main technical contribution is proving the restriction of Theorem 1.7 to normalized instances.

▶ **Theorem 4.2.** *There exists an algorithm that solves every normalized crossing-planar instance of* Multicut *and runs in* $f(\overline{cr},t)n^{O(\sqrt{\mu})}$.

The simple proof that Theorem 4.2 implies Theorem 1.7 is postponed to the full version. It remains to prove Theorem 4.2.

Given a normalized crossing-planar instance (G,H) of Multicut, we denote by $\mathcal{H}$ the set of graphs H' on $V(H) \cup V(E_{cr})$ and we use G' for $G \setminus E_{cr}$. Further, we let F^* denote the faces of G' which contain a pair of crossing edges in G. The main technical difficulty for the proof of Theorem 4.2 is contained in the following result.

▶ **Lemma 4.3.** *Let* (G,H) *be a normalized crossing-planar instance of* Multicut *admitting a multicut of finite weight. Then there exists* $H' \in \mathcal{H}$ *such that every minimum weight multicut for* (G',H') *is a minimum weight multicut for* (G,H) *and for every subcubic, inclusion-wise minimal minimum weight multicut dual* C *for* (G',H')*, we have* $\mathrm{tw}(C - F^*) = O(\sqrt{\mu})$.

Due to space restrictions, the full proof of Lemma 4.3 is postponed to the full version. We now give an overview of the proof of Lemma 4.3.

Let (B_1,B_2,I,X) be an extended biclique decomposition of H with $|X| = \mu$. Further, let S^* be a minimum weight multicut for (G,H) such that S^* minimizes the number of elements of $V(H) \cup V(E_{cr})$ that are in a component of $G \setminus S^*$ that also contains a terminal of X. In the following, we use $\overline{X}$ for the set of elements of $V(H) \cup V(E_{cr})$ that are in a component of $G \setminus S^*$ that also contains a terminal of X, we use $\overline{B_1}$ for the set of elements of $(V(H) \cup V(E_{cr})) \setminus \overline{X}$ that are in a component of $G \setminus S^*$ that also contains a terminal of B_1, and we use $\overline{B_2}$ for $(V(H) \cup V(E_{cr})) \setminus (\overline{X} \cup \overline{B_1})$. Observe that $(\overline{B_1}, \overline{B_2}, \overline{X})$ is a partition of $V(H) \cup V(E_{cr})$. We now define H' to be the complete multipartite graph on $V(H) \cup V(E_{cr})$ such that $\overline{B_1}$ and $\overline{B_2}$ are partition classes and such that for every component of $G \setminus S^*$ that contains at least one terminal of X, all the elements of $\overline{X}$ contained in this component form a partition class.

This finishes the description of H'. Observe that $H' \in \mathcal{H}$. Further observe that H is a subgraph of H'.The following two claims directly imply that every minimum weight multicut for (G', H') is also a minimum weight multicut for (G, H'). Claim 4.4 will be reused for the proof of the treewidth bound. These claims mainly follow from the construction of (G', H').

▷ Claim 4.4. Let S' be a multicut for (G', H'). Then S' is also a multicut for (G, H').

▷ Claim 4.5. S^* is a multicut for (G', H').

We are now ready to conclude that any minimum weight multicut S for (G', H') is a minimum weight multicut for (G, H). First observe that S is a multicut for (G, H) by Claim 4.4 and as H is a subgraph of H'. Further, by Claim 4.5 and the minimality of S, we obtain $w(S) \leq w(S^*)$ where w is the weight function associated to G. By the minimality of S^*, the statement follows.

In the following, let C be a subcubic, inclusion-wise minimal, minimum weight multicut dual for (G', H'). We will give the desired treewidth bound on C. Let $S' = E_G(C)$ and observe that S' is a minimum weight multicut for (G', H') by Lemma 2.1. It hence follows from the first part of the lemma that S' is also a minimum weight multicut for (G, H). Further, we use C' for $C - F^*$.

Our strategy for proving the second part of the lemma is to show that the faces of C' containing a terminal of $\overline{X}$ form a small set of faces that is close to every face of C' in a certain sense. We first give the following result showing that the number of such faces is small indeed. Its proof uses the fact that S^* is chosen so that the size of $\overline{X}$ is minimized and the fact that S' is a minimum weight multicut for (G, H).

▷ Claim 4.6. Let f be a face of C' that contains a terminal of $\overline{X}$. Then f also contains a terminal of X.

The next result uses the structure of H.

▷ Claim 4.7. Let $v \in V(C')$ with $d_{C'}(v) = 3$. Then v is incident to a face of C' that contains a terminal of $\overline{X}$.

We are now ready to conclude the desired treewidth bound on C'. Let C'' be a component of C'. For every face f of C'', we now introduce a vertex z_f that shares an edge with every vertex that is incident to f. Let C_0'' be the resulting graph. It turns out that C_0'' is a planar graph and the vertices corresponding to faces containing terminals of $\overline{X}$ form a 3-dominating set in C_0''. By Proposition 2.2 and $|A| \leq |X| \leq \mu$, we obtain that $\mathrm{tw}(C_0'') = O(\sqrt{\mu})$ and hence $\mathrm{tw}(C') = O(\sqrt{\mu})$. This allows to conclude Lemma 4.3.

With Lemma 4.3 at hand, it is easy to conclude Theorem 4.2. The proof can be found in the full version.

References

1 Vincent Cohen-Addad, Éric Colin de Verdière, Dániel Marx, and Arnaud de Mesmay. Almost tight lower bounds for hard cutting problems in embedded graphs. *J. ACM*, 68(4):30:1–30:26, 2021. doi:10.1145/3450704.

2 Éric Colin de Verdière. Multicuts in planar and bounded-genus graphs with bounded number of terminals. *Algorithmica*, 78:1206–1224, 2017. doi:10.1007/S00453-016-0258-0.

3 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994. doi:10.1137/S0097539792225297.

4 Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, July 2005. doi:10.1145/1077464.1077468.

5 Jacob Focke, Florian Hörsch, Shaohua Li, and Dániel Marx. Multicut Problems in Embedded Graphs: The Dependency of Complexity on the Demand Pattern. In Wolfgang Mulzer and Jeff M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 57:1–57:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2024.57.

6 Jacob Focke, Florian Hörsch, Shaohua Li, and Dániel Marx. Multicut problems in embedded graphs: The dependency of complexity on the demand pattern, 2025. arXiv:2312.11086.

7 Martin Grohe. Computing crossing numbers in quadratic time. *Journal of Computer and System Sciences*, 68(2):285–302, 2004. Special Issue on STOC 2001. doi:10.1016/j.jcss.2003.07.008.

8 T. C. Hu. Multi-commodity network flows. *Operations Research*, 11(3):344–360, 1963.

9 Florian Hörsch and Dániel Marx. Multicut problems in almost-planar graphs: The dependency of complexity on the demand pattern, 2025. doi:10.48550/arXiv.2504.21624.

10 Philip N. Klein and Dániel Marx. Solving planar k-terminal cut in $O(n^{c\sqrt{k}})$ time. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 569–580. Springer, 2012. doi:10.1007/978-3-642-31594-7_48.

11 Dániel Marx. A tight lower bound for planar multiway cut with fixed number of terminals. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 677–688. Springer, 2012. doi:10.1007/978-3-642-31594-7_57.

12 Paul D. Seymour. A short proof of the two-commodity flow theorem. *J. Comb. Theory, Ser. B*, 26(3):370–371, 1979. doi:10.1016/0095-8956(79)90012-1.

Parameterized Approximability for Modular Linear Equations

Konrad K. Dabrowski
Newcastle University, UK

Peter Jonsson
Linköping University, Sweden

Sebastian Ordyniak
University of Leeds, UK

George Osipov
Linköping University, Sweden
University of Oxford, UK

Magnus Wahlström
Royal Holloway, University of London, UK

Abstract

We consider the Min-r-Lin($\mathbb{Z}_m$) problem: given a system S of length-r linear equations modulo m, find $Z \subseteq S$ of minimum cardinality such that $S - Z$ is satisfiable. The problem is NP-hard and UGC-hard to approximate in polynomial time within any constant factor even when $r = m = 2$. We focus on parameterized approximation with solution size as the parameter. Dabrowski, Jonsson, Ordyniak, Osipov and Wahlström [SODA-2023] showed that Min-2-Lin($\mathbb{Z}_m$) is in FPT if m is prime (i.e. $\mathbb{Z}_m$ is a field), and it is W[1]-hard if m is not a prime power. We show that Min-2-Lin($\mathbb{Z}_{p^n}$) is FPT-approximable within a factor of 2 for every prime p and integer $n \geq 2$. This implies that Min-2-Lin($\mathbb{Z}_m$), $m \in \mathbb{Z}^+$, is FPT-approximable within a factor of $2\omega(m)$ where $\omega(m)$ counts the number of distinct prime divisors of m. The high-level idea behind the algorithm is to solve tighter and tighter relaxations of the problem, decreasing the set of possible values for the variables at each step. When working over $\mathbb{Z}_{p^n}$ and viewing the values in base-p, one can roughly think of a relaxation as fixing the number of trailing zeros and the least significant nonzero digits of the values assigned to the variables. To solve the relaxed problem, we construct a certain graph where solutions can be identified with a particular collection of cuts. The relaxation may hide obstructions that will only become visible in the next iteration of the algorithm, which makes it difficult to find optimal solutions. To deal with this, we use a strategy based on shadow removal [Marx & Razgon, STOC-2011] to compute solutions that (1) cost at most twice as much as the optimum and (2) allow us to reduce the set of values for all variables simultaneously. We complement the algorithmic result with two lower bounds, ruling out constant-factor FPT-approximation for Min-3-Lin(R) over any nontrivial ring R and for Min-2-Lin(R) over some finite commutative rings R.

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

Keywords and phrases parameterized complexity, approximation algorithms, linear equations

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.88

Related Version *Full Version*: https://arxiv.org/abs/2509.04976

Funding *Peter Jonsson*: Supported by the Swedish Research Council (VR) under grant 2021-04371.
Sebastian Ordyniak: Supported by the Engineering and Physical Sciences Research Council (EPSRC, project EP/V00252X/1).
George Osipov: Supported by the Swedish Research Council (VR) under grant 2024-00274.



© Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, and Magnus Wahlström; licensed under Creative Commons License CC-BY 4.0



1 Introduction

Systems of linear equations are ubiquitous in computer science and mathematics [16] and methods like Gaussian elimination can efficiently solve linear systems over various rings. Equations and congruences over the ring of integers modulo m ($\mathbb{Z}_m$) are of central importance in number theory, but also have many applications in computer science, including complexity theory, coding theory, cryptography, hash functions and pseudorandom generators, see e.g. [3, 9, 11, 33]. Linear equations over modular rings can be solved in polynomial time, but the methods are not suited to dealing with inconsistent systems of equations. We consider the Min-r-Lin(R) problem which asks to find an assignment to a system of linear equation over the ring R that violates the minimum number of equations, and where each equation contains at most r distinct variables. This problem is NP-hard even when $r = 2$ and R is the simplest nontrivial ring $\mathbb{Z}_2$ [23]. We note that Min-r-Lin(R) for $r \in \mathbb{N}$ and finite ring R is a special case of MinCSP(Γ) for a finite constraint language Γ. This, and the more general Valued CSP, have been widely studied from many different perspectives, e.g. [4, 6, 22, 23, 31, 32].

Two common ways of coping with NP-hardness are approximation and parameterized algorithms, but neither of them seems sufficient in isolation to deal with Min-r-Lin($\mathbb{Z}_m$). Even Min-2-Lin over finite fields such as $\mathbb{Z}_2$ is conjectured to be NP-hard to approximate within any constant factor under the Unique Games Conjecture (UGC) [20]: see Definition 3 in [21] and the discussion that follows. The natural parameter for Min-r-Lin($\mathbb{Z}_m$) is the cost of the optimal solution (i.e. the number of equations not satisfied by it), which we denote by k. Under this parameterization, Min-2-Lin($\mathbb{Z}_m$) is fixed-parameter tractable when m is a prime, i.e. $\mathbb{Z}_m$ is a field, but W[1]-hard when m is not a prime power. Moreover, the problem Min-3-Lin is W[1]-hard for every nontrivial ring [8]. This motivates us to study *parameterized approximation* algorithms [14, 27]. This approach has received rapidly increasing interest (see, for instance, [13, 17, 18, 25, 26, 30]). Let $c \geq 1$ be a constant. A factor-c *FPT-approximation algorithm* takes an instance (I, k), runs in $O^*(f(k))$[1] time for an arbitrary computable function f, either returns that there is no solution of size at most k or returns that there is a solution of size at most $c \cdot k$. Thus, there is more time to compute the solution (compared to polynomial-time approximation) and the algorithm may output an oversized solution (unlike an exact FPT algorithm). [2] Our main result is the following. Let $\omega(m)$ be the number of distinct prime factors of m.

▶ **Theorem 1.** *For every $m \in \mathbb{Z}_+$, Min-2-Lin($\mathbb{Z}_m$) is* FPT*-approximable within $2\omega(m)$.*

We complement the result with two lower bounds. First, we show that allowing three or more variables per equation leads to W[1]-hardness of constant-factor approximation.

▶ **Theorem 2.** *Min-3-Lin(R) over every nontrivial ring R is W[1]-hard to approximate within any constant factor.*

[1] The notation $O^*(\cdot)$ hides polynomial factors in the input size.

[2] A decision c-approximation procedure for Min-2-Lin($\mathbb{Z}_m$) can be turned into an algorithm that returns a c-approximate solution using self-reducibility: if (S, k) is a yes-instance, then there exists a subset $S' \subseteq S$, $|S'| \leq c$, such that $(S - S', k - 1)$ is a yes-instance; moreover, such S' can be found by iterating over all subsets of size at most c, which incurs a polynomial overhead on the running time of the algorithm.

This result strengthens two previously known hardness results: (i) Min-3-Lin(R) is W[1]-hard [8] and (ii) Min-3-Lin(R) is NP-hard to approximate within any constant (which can easily be derived from [19]). While we focus on rings of the form $\mathbb{Z}_m$, the result of Theorem 1 begs the questions whether Min-2-Lin(R) is FPT-approximable within a constant factor for every finite commutative ring R. We answer this question in the negative.

▶ **Theorem 3** (See Theorem 17 for a more detailed statement.). *There exist finite commutative rings R such that Min-2-Lin(R) is W[1]-hard to approximate within any constant factor.*

Theorems 5.2 and 6.2 in [8] leave open the question of whether Min-2-Lin($\mathbb{Z}_{p^n}$) is FPT or W[1]-hard for a prime p and $n \geq 2$. The answer is unknown even for the smallest such ring – $\mathbb{Z}_4$. While our result implies that Min-2-Lin($\mathbb{Z}_{p^n}$) is FPT-approximable within a factor of 2, its exact parameterized complexity remains an intriguing open problem.

Full proofs for the results marked with a $\star$ can be found in the full version of the paper. Another version of this paper is available on arXiv [7]; it considers a broader class of rings, but gives worse approximation factors.

2 Preliminaries

For the basics of graph theory and parameterized complexity, we refer to [10, 12, 15, 29].

An expression $c_1 \cdot x_1 + \cdots + c_r \cdot x_r = c$ is a *(linear) equation over R* if $c_1, \ldots, c_r, c \in R$ and $x_1, \ldots, x_r$ are variables with domain R. This equation is *homogeneous* if $c = 0$. Let S be a set (or equivalently a system) of equations over R. Let $V(S)$ denote the variables in S, and we say that S is *consistent* if there is an assignment $\varphi : V(S) \to R$ satisfying all equations in S. An instance of the computational problem r-Lin(R) is a system S of equations in at most r variables over R, and the question is whether S is consistent. Linear equation systems over $\mathbb{Z}_m$ are solvable in polynomial time and the well-known procedure is outlined, for instance, in [1, p. 473]. We now define the computational problem when we allow some equations in an instance to be soft (i.e. deletable at unit cost) and crisp (i.e. undeletable).

Min-r-Lin(R)	
Instance:	A (multi)set S of equations over R with at most r variables per equation, a subset $S^\infty \subseteq S$ of crisp equations and an integer k.
Parameter:	k.
Question:	Is there a set $Z \subseteq S \setminus S^\infty$ such that $S - Z$ is consistent and $\lvert Z\rvert \leq k$?

We use crisp equations for convenience since they can be modelled by $k+1$ copies of the same soft equation. For an assignment $\alpha : V(S) \to R$, let $\mathrm{cost}_S(\alpha)$ be ∞ if α does not satisfy a crisp equation and the number of unsatisfied soft equations otherwise. We drop the subscript S when it is clear from context. We write $\mathrm{mincost}(S)$ to denote the minimum cost of an assignment to S.

3 FPT-Approximation Algorithm

3.1 Algorithm Summary

Let $p_1^{n_1} \cdots p_\ell^{n_\ell}$ be the prime factorization of $m \in \mathbb{Z}_+$. It is well known that $\mathbb{Z}_m$ is isomorphic to the direct sum $\bigoplus_{i=1}^{\ell} \mathbb{Z}_{p_i^{n_i}}$, and we can reduce the problem to the prime power case.

▶ **Proposition 4** (⋆). *Suppose that the ring R is isomorphic to a direct sum $\bigoplus_{i=1}^{\ell} R_i$. If* Min-2-Lin$(R_i)$ *is* FPT*-approximable within a factor c_i for all $i \in [\ell]$, then* Min-2-Lin(R) *is* FPT*-approximable within a factor $\sum_{i=1}^{\ell} c_i$.*

Now, consider the ring $\mathbb{Z}_{p^n}$ for a prime p and positive integer n. We start with a simplification step. An equation over a ring R is *simple* if it is either a binary equation of the form $u = rv$ for some $r \in R$ or a crisp unary equation $u = r$ for some $r \in R$. An instance S of 2-Lin(R) is simple if every equation in S is simple.

▶ **Lemma 5** (⋆). *Let m be positive integer and c be a constant. If* Min-2-Lin$(\mathbb{Z}_m)$ *restricted to simple instances is* FPT*-approximable within a factor c, then* Min-2-Lin$(\mathbb{Z}_m)$ *on general instances is* FPT*-approximable within a factor c.*

Now, suppose $n = 1$, i.e. we are working over a field $\mathbb{Z}_p$. Min-2-Lin$(\mathbb{Z}_p)$ can be solved exactly in FPT time (see. [8]); we will show a 2-approximation because it is illustrative of our approach. For an instance S of 2-Lin$(\mathbb{Z}_p)$, we construct a graph $G = G(S)$ with vertices x_i for every $x \in V(G)$ and $1 \leq i \leq p-1$ and special vertices s and t. For unary equations $x = r$ in S, add crisp edges sx_r if $r \neq 0$, and crisp edges $x_r t$ for every $1 \leq r \leq p-1$ if $r = 0$. For every binary equation e of the form $x = r \cdot y$, construct an edge bundle $B_e = \{x_{ri}y_i : 1 \leq i \leq p-1\}$ and add these edges to G. This completes the construction. We establish a correspondence between conformal st-cuts in $G(S)$ and assignments to S. Formally, for a set of vertices $X \subseteq V(G)$, let $\delta(X)$ be the set of edges with exactly one endpoint in X, i.e. $\delta(X)$ is the cut separating X and $\overline{X}$. If $U \subseteq V(G)$ is such that $s \in U$, $t \notin U$ and there is at most one vertex $x_i \in U$ for any $x \in V(S)$, we say that $\delta(U)$ is a *conformal st-cut*. To construct an assignment from a conformal cut U, map $x \mapsto i$ whenever $x_i \in U$ for some $i \in \{1, \ldots, p-1\}$, and otherwise $x \mapsto 0$; the reverse direction of the correspondence is obvious.

Let $b(U)$ be the number of edge bundles B_e intersected by $\delta(U)$. Then R being a field implies that $b(U)$ is exactly the cost of the assignment corresponding to U. More specifically, consider an equation $e = (x = r \cdot y)$, suppose $x_i, y_j \in U$ and $B_e \cap \delta(U) = \emptyset$. Then $i = rj$ because the edge $x_{rj}y_j$ is uncut and both its endpoints are reachable from s, so the assignment corresponding to U satisfies e. This guarantee is represented by B_e being a matching: view an edge $x_i y_j$ as encoding "$x = i$ if and only if $y = j$". To complete the algorithm for fields, we compute a conformal cut $\delta(U)$ with $b(U) \leq k$. Observe that $b(U) \leq k$ implies $\delta(U) \leq 2k$ because U contains at most one vertex per variable so $\delta(U)$ may intersect at most two edges of any bundle. Thus, for 2-approximation it suffices to compute a conformal cut of size $2k$, which is guaranteed to exist for yes-instances. However, when translating it back into a set of equations, we may delete $2k$ equations because all edges of our conformal cut may intersect distinct bundles. Finally, to compute a conformal cut in FPT time, we may use branching in the style of Digraph Pair Cut [24]: compute the closest st-cut (which is unique by submodularity of cuts), and if it is not conformal, then branch.

If we use the same approach naïvely over $\mathbb{Z}_{p^n}$, $n \geq 2$, then the bundles B_e stop being matchings. Consider the equation $e = (x = 2y)$ over $\mathbb{Z}_8$ (second graph from the left in Figure 1). Note that both y_2 and y_6 are adjacent to x_4 in B_e. Moreover, if $x = 4$ then either $y = 2$ or $y = 6$ so the dependencies cannot be captured by binary edges (even if one were to use directed graphs). Thus, we lose the connection between the number of bundles intersected by a conformal cut and the cost of the corresponding assignment. One idea for solving Min-2-Lin$(\mathbb{Z}_{p^n})$ is to retain the "if and only if" semantics of an edge by matching *sets* of values rather than individual values. More specifically, let us partition $\{1, \ldots, p^n - 1\}$ into classes $C_1, \ldots, C_\ell$ and build $G(S)$ with vertices s, t and $x_{C_1}, \ldots, x_{C_\ell}$ for all $x \in V(S)$. To keep the matching structure for an equation e, we want an edge $x_{C_i}y_{C_j}$ in B_e to mean

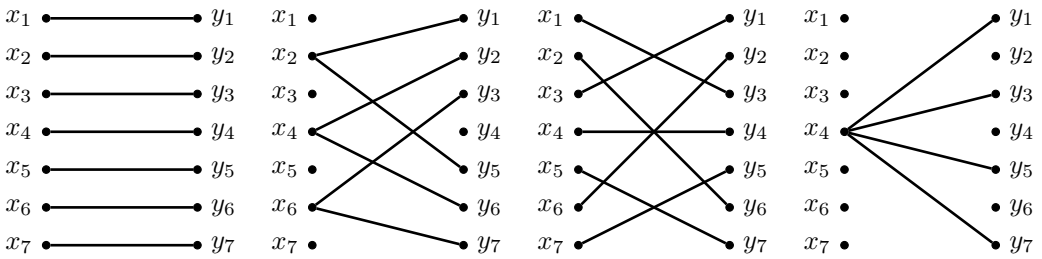


Figure 1 Graphs B_e corresponding to equations $x = 1 \cdot y$, $x = 2 \cdot y$, $x = 3 \cdot y$ and $x = 4 \cdot y$.

"$x \in C_i$ if and only if $y \in C_j$". For fields, we have used the most refined partition (every nonzero element is a class of its own). For other rings, a coarser partition is needed: e.g. if $R = \mathbb{Z}_8$, then 2 and 6 have to be in the same class (think of $x = 2y$). However, simply taking a coarser partition is not sufficient: indeed, the coarsest partition (putting all non-zero elements in the same class) has the required structure, but it only distinguishes between zero and nonzero values, and is not very useful algorithmically. Intuitively, we want to a partition such that a class assignment over $\mathbb{Z}_{p^n}$ allows us to rewrite our input as a set of equations over $\mathbb{Z}_{p^{n-1}}$ without increasing the cost too much.

A useful partition is obtained by viewing the elements of $\mathbb{Z}_{p^n}$ represented in base-p. Formally, every element $a \in \mathbb{Z}_{p^n}$ equals $\sum_{i=0}^{p-1} a_i p^i$, where the coefficients $a_0, \ldots, a_{p-1} \in \mathbb{Z}_p$ uniquely define a. Let $\vec{a} = (a_0, \ldots, a_{p-1})$, and for every $a \neq 0$, define $\mathrm{ord}(a) = \min\{i : a_i \neq 0\}$ to be the index of the first nonzero coordinate in $\vec{a}$, and $\mathrm{lsu}(a) = r_{\mathrm{ord}(a)}$ to be the least significant unit in $\vec{a}$. For completeness, let $\mathrm{ord}(0) = \mathrm{lsu}(0) = 0$. Let

$$a \equiv b \iff \mathrm{ord}(a) = \mathrm{ord}(b) \text{ and } \mathrm{lsu}(a) = \mathrm{lsu}(b).$$

This equivalence relation has two important properties. First, it is *matching*, i.e. $\{0\}$ is an equivalence class, and for every $i, j \in \mathbb{Z}_{p^n}$ and $r \in \mathbb{Z}_{p^n}$,

- if $i \equiv j$ then $ri \equiv rj$,
- if $i \not\equiv j$, then either $ri \not\equiv rj$ or $ri = rj = 0$.

Moreover, it is *absorbing*, meaning that

- $i \equiv j \implies p$ divides $i - j$ for all $i, j \in \mathbb{Z}_{p^n}$.

Let Γ_{p^n} denote the set of equivalence classes of $\equiv$, and $\Gamma_{p^n}^{\neq 0} = \Gamma_{p^n} \setminus \{\{0\}\}$. We will drop the subscript when it is clear from the context. The name "matching" comes from considering bipartite graphs $G_r^{\neq 0}$ defined by binary equations $u = rv$ for every $r \in R$ as follows: let $V(G_r) = \Gamma^{\neq 0} \uplus \Gamma^{\neq 0}$ and let there be an edge between two classes C_1 on the left and C_2 on the right if and only if $i = rj$ for some $i \in C_1$ and $j \in C_2$. Then $\equiv$ being matching implies that $G_r^{\neq 0}$ is a matching for every r, i.e. every vertex has degree at most 1.

▶ **Example 6.** Partition Γ_{3^2} (with elements written in base-3) has classes

$$\{01_3, 11_3, 21_3\}, \{02_3, 12_3, 22_3\}, \{10_3\}, \{20_3\}, \{00_3\}.$$

For another example, Γ_{2^3} has classes

$$\{001_2, 011_2, 101_2, 111_2\}, \{010_2, 110_2\}, \{100_2\}, \{000_2\}.$$

For a non-example, a coarser partition of $\mathbb{Z}_{2^3}$ into three classes

$$\{001_2, 011_2, 101_2, 111_2\}, \{010_2, 110_2, 100_2\}, \{000_2\}.$$

lacks the matching property (as is evident from the equation $x = 2y$ in Figure 1).

The matching property of $\equiv$ is crucial for the main algorithmic lemma, which we state below and prove in Section 3.2. A value assignment $\alpha : V(S) \to \mathbb{Z}_{p^n}$ *agrees with* a class assignment $\tau : V(S) \to \Gamma$ if $\alpha(v) \in \tau(v)$ for all $v \in V(S)$. A class assignment τ *respects* an equation e if it admits a satisfying assignment that agrees with τ, otherwise we say that τ *violates* e. Define the cost of a class assignment τ to be the number of equations in S that it violates. Note that every value assignment α uniquely defines a class assignment $\tau_\alpha : V(S) \to \Gamma_{p^n}$, so we say that α *strongly violates* an equation e if τ_α violates e. Clearly, an optimal assignment can violate at most k equations in S, and we can guess the number $q \in \{0, \ldots, k\}$ of strongly violated equations.

▶ **Lemma 7.** *Let p be a prime, n be a positive integer, and let k and q be integers with $k \geq q$. There is a randomized algorithm that takes a simple instance S of 2-Lin($\mathbb{Z}_{p^n}$) and integers k and q as input, and in $O^*(2^{O(k)})$ time returns a class assignment $\tau : V(S) \to \Gamma_{p^n}$ such that the following holds. Let Y be the set of equations in S violated by τ. Then have $|Y| \leq 2q$. Moreover, if S admits an assignment of cost at most k that strongly violates at most q equations, then with probability at least $2^{-O(k+q^2)}$, $S - Y$ admits an assignment of cost at most $k - |Y|/2$ that agrees with τ.*

While technical details are deferred to Section 3.2, we remark that we can no longer use branching in the style of Digraph Pair Cut when working over non-fields since some conformal cuts of low cost correspond to class assignments of high cost. For an extreme example over $\mathbb{Z}_8$, consider the system of equations $\triangle = \{x = 4, 2a = x, 3a = b, 3b = c, 3c = a\}$. By the construction of $G(\triangle)$, the vertex t is isolated and the connected component U of s contains vertices s, $x_{\{4\}}$, $a_{\{2,6\}}$, $b_{\{2,6\}}$ and $c_{\{2,6\}}$. Hence, $\delta(U)$ is empty and conformal, but the cost of $\triangle$ is at least 1 because the system is inconsistent (this follows from considering both possible values for a, which are 2 and 6). In fact, the cost is exactly 1 because it is sufficient to delete $2a = x$. To mitigate this issue of "invisible" future costs, we use shadow removal in the class assignment graph followed by branching on the shadow components.

Now, to explain how the absorbing property of $\equiv$ is used, we need some definitions. Choose an arbitrary representative element $C^\sim$ from every equivalence class $C \in \Gamma$. Consider a simple equation e and a class assignment $\tau : V(e) \to \Gamma$ to its variables that respects e. For unary equations $e = (u = r)$, define $e' = \text{next}(e, \tau)$ to be

$$u' = \frac{r - \tau(u)^\sim}{p}.$$

For binary equations $e = (u = rv)$, define $e' = \text{next}(e, \tau)$ to be

$$u' = rv' + \frac{r\tau(v)^\sim - \tau(u)^\sim}{p}.$$

The absorbing property implies that $\text{next}(e, \tau)$ is defined in both cases. Indeed, if $e = (u = r)$ and τ respects e, then $r \in \tau(u)$ and $r \equiv \tau(u)^\sim$, so p divides $r - \tau(u)^\sim$. If $e = (u = rv)$ and τ respects e, then there is an assignment $\alpha : \{u, v\} \to \mathbb{Z}_{p^n}$ such that $\alpha(u) = r\alpha(v)$, $\alpha(u) \in \tau(u)$ and $\alpha(v) \in \tau(v)$. Equivalently, $\alpha(u) - r\alpha(v) = 0$ and p divides both $\tau(u)^\sim - \alpha(u)$ and $\tau(v)^\sim - \alpha(v)$, hence p also divides any linear combination of these two values, particularly

$$r(\tau(v)^\sim - \alpha(v)) - (\tau(u) - \alpha(u)^\sim) = r\tau(v)^\sim - \tau(u)^\sim + (\alpha(u) - r\alpha(v)) = r\tau(u)^\sim - \tau(v)^\sim.$$

▶ **Lemma 8 (⋆).** *Let p be a prime and $n \in \mathbb{Z}_+$. Let e be a simple equation over $\mathbb{Z}_{p^n}$, and $\tau : V(e) \to \Gamma$ be a class assignment. Then τ respects e if and only if* $\text{next}(e, \tau)$ *is satisfiable over $\mathbb{Z}_{p^{n-1}}$.*

Now we combine all ingredients to prove the main theorem.

▶ **Theorem 1.** *For every $m \in \mathbb{Z}_+$, Min-2-Lin($\mathbb{Z}_m$) is* FPT*-approximable within $2\omega(m)$.*

Proof. Let $m = p_1^{n_1} \cdots p_\ell^{n_\ell}$ be the prime factorization of m. Note that $\omega(m) = \ell$, so by Proposition 4, it suffices to show that Min-2-Lin($\mathbb{Z}_{p^n}$) is FPT-approximable within factor 2 for every prime p and positive integer n. We proceed by induction on n. If $n = 1$, then we can use the algorithm of [8] for Min-2-Lin over fields to solve the problem exactly. Otherwise, let (S, k) be an instance of Min-2-Lin($\mathbb{Z}_{p^n}$). By Lemma 5, we may assume without loss of generality that S is simple.

Select a value $q \in \{0, \ldots, k\}$ uniformly at random and run the algorithm from Lemma 7 on (S, k, q) to produce a class assignment $\tau : V(S) \to \Gamma_{p^n}$. Let Y be the set of equations in S violated by τ. If $|Y| > 2q$, then reject (S, k). Otherwise, create an instance S' of 2-Lin($\mathbb{Z}_{p^{n-1}}$) with $V(S') = \{v' : v \in V(S)\}$ and $S' = \{\text{next}(e, \tau) : e \in S - Y\}$. Set $k' = k - \lceil |Y|/2 \rceil$ and pass (S', k') as input to the algorithm for Min-2-Lin($\mathbb{Z}_{p^{n-1}}$), and return the same answer.

Towards correctness, first assume (S, k) is a yes-instance, and let q be the number of equations strongly violated by an optimal assignment to S. Our guess for q is correct with probability $\frac{1}{k+1}$. By Lemma 7, with probability $2^{-O(q^2)}$, we have $|Y| \leq 2q$ and $S - Y$ admits an assignment of cost at most $k - |Y|/2$ that agrees with τ. By Lemma 8, the instance S' of 2-Lin($\mathbb{Z}_{p^{n-1}}$) we create has cost at most k', so we accept.

For the other direction, suppose the algorithm accepts. Then (S', k') is a yes-instance of Min-2-Lin($\mathbb{Z}_{p^{n-1}}$), and by Lemma 8, mincost$(S - Y) \leq k - |Y|/2$. Let Z be an optimal solution to $S - Y$. Then $Y \cup Z$ is a solution to S, and $|Y \cup Z| \leq |Y| + |Z| \leq 2(k - |Y|/2) + |Y| \leq 2k$.

We remark that all steps in the algorithm can be derandomized either by exhaustive branching or standard methods in [28]. ◀

3.2 Computing Class Assignments

This subsection is devoted to a proof of Lemma 7. To achieve this we introduce the class assignment graph (Section 3.2.1) and show that certain cuts in this graph correspond to class assignments (Section 3.2.2), which themselves correspond to solutions of Min-2-Lin($\mathbb{Z}_{p^n}$). We then use shadow removal and branching to compute these cuts (Section 3.2.3). Throughout this section, we consider the ring $\mathbb{Z}_{p^n}$ for prime number p and integer n.

3.2.1 The Class Assignment Graph

In what follows, let (S, k) be a simple instance of Min-2-Lin($\mathbb{Z}_{p^n}$) and let $\equiv$ be the matching and absorbing equivalence relation on $\mathbb{Z}_{p^n}$ defined in Section 3.1. Without loss of generality, we assume that every equation of S is consistent (otherwise we can remove the equation and decrease k by 1) and non-trivial (otherwise we can remove the equation). We first use the matching property of $\equiv$ to define the mapping $\pi_e : \Gamma_{p^n}^{\neq 0} \to \Gamma_{p^n}$ between equivalence classes for any equation $e = (ax = y)$ with $a \in \mathbb{Z}_{p^n} \setminus \{0\}$, as follows. For every $C \in \Gamma_{p^n}^{\neq 0}$, we set:

- $\pi_e(C) = 0$ if $ar = 0$ for every $r \in C$ and
- $\pi_e(C) = D$ otherwise, where D is the unique equivalence class such that e maps C to D. This is uniquely defined since $\equiv$ has the matching property.

For an equation $e = (x = a)$ with $a \in \mathbb{Z}_{p^n}$, we let π_e be the unique equivalence class $\Gamma_{p^n}(a)$ consistent with e. We can now define the *class assignment* graph $G = G(S)$; see Figure 2 for an illustration. The graph G has two distinguished vertices s and t together with vertices x_C

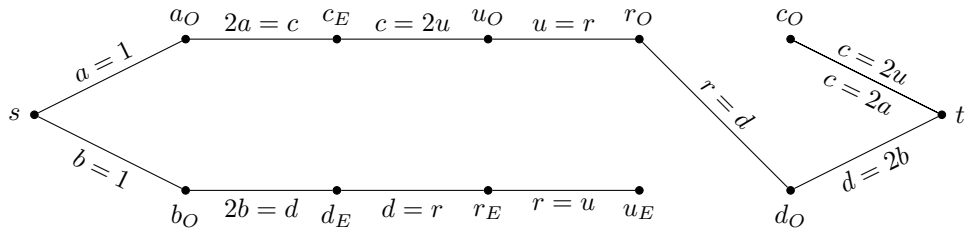


Figure 2 Let S be the instance of 2-Lin($\mathbb{Z}_4$) with variables a, b, c, d, u, r and equations $a = 1$, $b = 1$, $2a = c$, $c = 2u$, $u = r$, $2b = d$, and $d = r$. The figure illustrate the class assignment graph $G = G(S)$. Note that $\equiv$ has only two non-zero equivalence classes, namely, $E = \{2\}$ and $O = \{1, 3\}$. Every edge of G is annotated with the equation that implies it. A minimum conformal st-cut is given by the two edges that correspond to the equation $u = r$ and corresponds to the class assignment $a = O$, $b = O$, $c = E$, $d = E$, $u = O$, and $r = E$. Note that G has only one minimal conformal st-cut closest to s, namely $\{a_O c_E, b_O d_E\}$. This st-cut corresponds to the class assignment $a = O$, $b = O$, and $c = d = u = r = 0$. Therefore, the optimum solution for S only removes the equation $u = r$, however, any solution that corresponds to a minimum conformal st-cut closest to s has to remove the equations $2a = c$ and $2b = d$.

for every $x \in V(S)$ and every non-zero class $C \in \Gamma_{p^n}^{\neq 0}$. Moreover, G contains the following edges for each equation.

- For an equation $e = (ax = y)$ do the following. For every $C \in \Gamma_{p^n}^{\neq 0}$, add the edge $x_C y_{\pi_e(C)}$ if $\pi_e(C) \neq 0$. For every $D \in \Gamma_{p^n}^{\neq 0}$ such that $\pi_e^{-1}(D)$ is undefined, add the edge $y_D t$. If the equation is crisp, then the edges are crisp as well.
- For a crisp equation $e = (x = 0)$, add the crisp edge $x_C t$ for every class $C \in \Gamma_{p^n}^{\neq 0}$.
- For a crisp equation $e = (x = b)$, where $b \neq 0$, add the crisp edge $s x_{\pi_e}$.

Intuitively, every node of G corresponds to a Boolean variable and every edge e of G corresponds to an "if and only if" between the two Boolean variables connected by e. Moreover, every assignment φ of the variables of S naturally corresponds to a Boolean assignment, denoted by φ_G of the vertices in G by setting $s = 1$, $t = 0$, and:

- if $\varphi(x)$ belongs to the non-zero class C, then we set $x_C = 1$ and $x_{C'} = 0$ for every non-zero class C' not equal to C,
- if $\varphi(x) = 0$, we set $x_C = 0$ for every non-zero class C.

We say that an edge e of G is satisfied by φ if φ_G satisfies the "if and only if" Boolean constraint represented by that edge.

▶ **Observation 9.** *Let S be a simple instance of 2-Lin($\mathbb{Z}_{p^n}$), let φ be an assignment of S and pick $e \in S$. If φ satisfies e, then φ_G satisfies all edges corresponding to e in $G(S)$.*

3.2.2 Cuts in the Class Assignment Graph

In this section, we introduce *conformal cuts* and show how they relate to class assignments and solutions to Min-2-Lin($\mathbb{Z}_{p^n}$) instances. Let S be a simple instance of 2-Lin($\mathbb{Z}_{p^n}$) and $G = G(S)$. An st-cut Y in G is *conformal* if for every variable $x \in V(S)$ at most one vertex x_C for some $C \in \Gamma_{p^n}^{\neq 0}$ is connected to s in $G - Y$. Please refer to Figure 2 for an illustration of conformal cuts in the class assignment graph. If Y is a conformal st-cut in G, then we say that a variable x is *decided* with respect to Y if (exactly) one vertex x_C is reachable from s in $G - Y$; and otherwise we say that x is *undecided* with respect to Y. Moreover, we denote by τ_Y the assignment of variables of S to classes in Γ_{p^n} implied by Y, i.e. $\tau_Y(x) = 0$ if x is undecided and otherwise $\tau_Y(x) = C$, where C is the unique non-zero class in Γ_{p^n} such that

x_C is reachable from s in $G - Y$. We say that an assignment φ of S *agrees with* Y if $\varphi(x)$ is in the class $\tau_Y(x)$ for every variable x of S. Note that if some assignment agrees with Y, then Y is conformal. The following auxiliary lemma characterizes which edges of G are satisfied by an assignment φ of S after removing a set Y of edges from G.

▶ **Lemma 10.** *Let Y be a set of edges of G and let φ be an assignment of S. Then, φ_G satisfies all edges reachable from s in $G - Y$ if and only if φ_G sets all Boolean variables reachable from s in $G - Y$ to 1. Similarly, φ_G satisfies all edges reachable from t in $G - Y$ if and only if φ_G sets all Boolean variables reachable from t in $G - Y$ to 0.*

Proof. This follows because $\varphi_G(s) = 1$ and $\varphi_G(t) = 0$ for any φ and every edge of G corresponds to an "if and only if" between the variables corresponding to its two endpoints. ◀

For a set Z of equations of S, we let $\mathsf{ed}(Z)$ denote the set of all edges of G corresponding to an equation of Z. Conversely, for a set Y of edges of G, we let $\mathsf{eqn}(Y)$ denote all equations of S having a corresponding edge in Y. Moreover, if Y is an st-cut in G, we let $\mathsf{sep}(Y)$ denote the unique minimal st-cut contained in Y that is *closest* to s in G. Finally, for an optimal solution Z of S, we let $\overline{Z}$ be the set of equations $Z \setminus \mathsf{eqn}(\mathsf{sep}(\mathsf{ed}(Z)))$, i.e. all equations in Z that do not have an edge in $\mathsf{sep}(\mathsf{ed}(Z))$. We can now establish the connection between solutions of Min-2-Lin($\mathbb{Z}_{p^n}$) instances and conformal st-cuts in the class assignment graph.

▶ **Lemma 11** ($\star$)**.** *Let Z be a set of equations such that $S - Z$ is satisfiable and let $Y = \mathsf{ed}(Z)$. Then, $Y' = \mathsf{sep}(Y)$ satisfies:*

1. *Y' is a conformal st-cut.*
2. *$|Y'| \leq 2|\mathsf{eqn}(Y')| = 2|Z \setminus \overline{Z}|$.*
3. *There is a satisfying assignment for $S - Z$ that agrees with Y'.*

Proof Sketch. Let φ be a satisfying assignment of $S - Z$. Observation 9 implies that φ_G satisfies all edges of $G - Y$. Therefore, it follows from Lemma 10 that φ_G sets all vertices reachable from s in $G - Y$ to 1 and all vertices reachable from t in $G - Y$ to 0. Thus, Y is an st-cut, because otherwise t would have to be set to 1 by φ_G since it would be reachable from s in $G - Y$. Therefore, $Y' = \mathsf{sep}(Y)$ exists. Because Y' is closest to s, it holds that a vertex is reachable from s in $G - Y$ if and only if it is reachable from s in $G - Y'$. Therefore, if at least two vertices x_C and $x_{C'}$ for some distinct non-zero classes C and C' are reachable from s in $G - Y'$ for some variable x, then all of them must be set to 1 by φ_G, which is not possible due to the definition of φ_G. We conclude that Y' is conformal.

Towards showing that $|Y'| \leq 2|\mathsf{eqn}(Y')|$, it suffices to show that $|Y' \cap \mathsf{ed}(e)| \leq 2$ for every equation $e \in Z$. Note that because Y' is a minimal st-cut, it holds that one of the endpoints of every $y \in Y'$ is reachable from s in $G - Y'$. Therefore, because Y' is conformal, Y' can contain at most two edges in $\mathsf{ed}(e) = \{\, x_C y_{\pi_e(C)} \mid C \in \Gamma_{p^n}^{\neq 0} \wedge \pi_e(C) \neq 0 \,\} \cup \{\, y_D t \mid D \in \Gamma_{p^n}^{\neq 0} \wedge \pi_e^{-1}(D) \text{ is undefined} \,\}$ for every binary equation e of the form $ax = y$. Similarly, Y' can contain at most one edge in $\mathsf{ed}(e) = \{\, x_C t \mid C \in \Gamma_{p^n}^{\neq 0} \,\}$ for every unary equation e of the form $x = 0$. Finally, $|\mathsf{ed}(e)| = |\{s x_{\pi_e}\}| = 1$ for every unary equation e of the form $x = b$. Therefore, $|Y'| \leq 2|\mathsf{eqn}(Y')|$, and $Y' = Z \setminus \overline{Z}$ by definition.

Let D be the set of all variables of S such that no vertex x_C is reachable from s in $G - Y'$. Let φ' be the assignment for S such that $\varphi'(x) = 0$ if $x \in D$ and $\varphi'(x) = \varphi(x)$ otherwise. Clearly, φ' agrees with Y', because φ agrees with all variables not in D and all other variables are correctly set to 0 by φ'. It therefore only remains to show that φ' still satisfies $S - Z$, which we leave to the full version of the paper. ◀

3.2.3 Shadow Removal

We show how shadow removal (introduced in [28] and improved in [5]) can be used for computing conformal cuts that correspond to solutions of a MIN-2-LIN($\mathbb{Z}_{p^n}$) instance. We follow [5] and begin by importing some definitions, which we translate from directed graphs to undirected graphs to fit our setting; to get back to directed graphs one simply has to think of an undirected graph as the directed graph obtained after replacing each undirected edge with two directed arcs in both directions. Let G be an undirected graph. Let $\mathcal{F}$ be a set of connected subgraphs of G. A set $T \subseteq V(G)$ is an *$\mathcal{F}$-transversal* if T intersects every subgraph in $\mathcal{F}$. Conversely, if T is an $\mathcal{F}$-transversal, we say that $\mathcal{F}$ is *T-connected*.

▶ **Theorem 12** ([5]). *Let G be an undirected graph, $T \subseteq V(G)$ and $k \in \mathbb{N}$. There is a randomized algorithm that takes (G, T, k) as input and returns in $O^*(4^k)$ time a set $W \subseteq V(G) \setminus T$ such that the following holds with probability $2^{-O(k^2)}$. For every T-connected family of connected subgraphs $\mathcal{F}$ in G, if there is an $\mathcal{F}$-transversal of size at most k in $V(G) \setminus T$, then there is an $\mathcal{F}$-transversal $Y \subseteq V(G) \setminus (W \cup T)$ of size at most k such that every vertex $v \notin W \cup Y$ is connected to T in $G - Y$.*

The following lemma is crucial for the application of shadow removal (Theorem 12). Informally, it shows that if Z is a solution, i.e. a set of equations such that $S - Z$ is satisfiable, then we can obtain a (not too large) new solution $Z' = (\overline{Z} \cup \mathsf{eqn}(Y'))$ by replacing the corresponding conformal minimal sA-cut $Y = \mathsf{sep}(\mathsf{ed}(Z))$, where A is the set of vertices in G not reachable from s in $G - Y$, by any minimal sA-cut Y'.

▶ **Lemma 13.** *Let S be a simple instance of 2-LIN($\mathbb{Z}_{p^n}$) and $G = G(S)$. Moreover, let Z be a set of equations such that $S - Z$ is satisfiable, $Y = \mathsf{sep}(\mathsf{ed}(Z))$, A be the set of all vertices in G that are not reachable from s in $G - Y$, and let Y' be an sA-cut in G. Then, there is an assignment $\varphi : V(S) \to \mathbb{Z}_{p^n}$ of S that satisfies $S - Z'$ and agrees with Y', where $Z' = (\overline{Z} \cup \mathsf{eqn}(Y'))$.*

Proof. Lemma 11 implies that Y is conformal and there is a satisfying assignment φ for $S - Z$ that agrees with Y. Because Y' is also an sA-cut in G, if no vertex x_C is reachable from s in $G - Y$ for some variable x of S, then the same applies in $G - Y'$. Let D be the set of all variables x of S such that some vertex x_C is reachable from s in $G - Y$ but that is not the case in $G - Y'$. Let φ' be the assignment obtained from φ by setting all variables in D to 0. Then, φ' agrees with Y'. We claim that φ' also satisfies $S - Z'$, where $Z' = (\overline{Z} \cup \mathsf{eqn}(Y'))$.

Consider a unary equation e of $S - Z'$ on variable x. If $x \notin D$, then $\varphi'(x) = \varphi(x)$ and therefore φ' satisfies e (because e is crisp and therefore $e \notin Z$). So suppose that $x \in D$. If e is of the form $x = 0$, then φ' satisfies e. Otherwise, e is of the form $x = b$ for some $b \neq 0$ and $G - Y'$ contains the edge sx_{π_e}. Therefore, x_{π_e} is reachable from s in $G - Y'$ contradicting our assumption that $x \in D$.

Now, consider a binary equation $e = (ax = y)$ of $S - Z'$ on variables x and y and first consider the case when $e \in Z$. Clearly, if neither a vertex x_C nor a vertex y_C is reachable from s in $G - Y'$, then $\varphi'(x) = \varphi'(y) = 0$, so e is satisfied by φ'. We next show that either no vertex x_c or no vertex y_C is reachable from s in $S - Y'$. Suppose for a contradiction that x_{C_x} and y_{C_y} are reachable from s in $S - Y'$. Let h be an arbitrary edge in $\mathsf{ed}(e) \cap Y$; such an edge h exists because $e \in Z \setminus Z'$. Because Y is a minimal st-cut, it follows that exactly one endpoint of h is reachable from s in $G - Y$ and either x_C or y_C (endpoint of h) for some $C \in \Gamma_{p^n}$ must be reachable from s in $G - Y$. We assume without loss of generality that x_C is reachable from s in $G - Y$. Because Y' is an sA-cut and Y' does not contain h, x_C is not reachable from s in $G - Y'$. But then $C \neq C_x$ and both x_C and x_{C_x} are reachable from s in $G - Y$, which contradicts that Y is conformal. It remains to consider the case when there

is a vertex x_C that is reachable from s in $G - Y'$ but no vertex y_C is reachable from s in $G - Y'$; the case when there is a vertex y_C reachable from s in $G - Y'$ but no vertex x_C reachable from s in $G - Y'$ is analogous. Since $Y' \cap \mathsf{ed}(e) = \emptyset$, we obtain that $\pi_e(C) = 0$ since otherwise either t or some $y_{C'}$ would be reachable from s in $G - Y'$. Because $\varphi'(y) = 0$, it follows that e is satisfied by φ'. This completes the proof for the case when $e \in Z$.

Suppose instead that $e \notin Z$. In this case φ satisfies e and therefore φ' also satisfies e unless exactly one of x and y is not in D. We distinguish the following cases:

- $x \notin D$ and $y \in D$. If there is no vertex x_C that is reachable from s in $G - Y'$, then the same holds in $G - Y$ so $\varphi'(x) = \varphi(x) = \varphi'(y) = 0$, which shows that φ' satisfies e. Otherwise, let x_C be reachable from s in $G - Y'$. Then, $\pi_e(C) = 0$ since otherwise either t or $y_{\pi_e(C)}$ is also reachable from s in $G - Y'$ (because $Y' \cap \mathsf{ed}(e) = \emptyset$), which in the former case contradicts our assumption that Y' is an st-cut and which in the latter case contradicts our assumption that $y \in D$. Therefore, φ' satisfies e (because $\varphi'(y) = 0$).
- $x \in D$ and $y \notin D$. We first show that there is no vertex y_C that is reachable from s in $G - Y'$. Suppose there is such a vertex y_C. Then, $\pi_e^{-1}(C)$ is undefined since otherwise $x_{\pi_e^{-1}(C)}$ is reachable from s in $G - Y'$ (because $Y' \cap \mathsf{ed}(e) = \emptyset$), which contradicts our assumption that $x \in D$. But then $y_C t \in E(G - Y')$ and t is reachable from s in $G - Y'$, which contradicts our assumption that Y' is an st-cut. Hence, there is no vertex y_C that is reachable from s in $G - Y'$, which implies that the same holds in $G - Y$ so $\varphi'(y) = \varphi(y) = \varphi'(x) = 0$, which shows that φ' satisfies e. ◀

Let $G = G(S)$ and for a set $W \subseteq V(G)$, let $\delta(W)$ be the set of edges incident to a vertex in W and a vertex in $V(G) \setminus W$. The forthcoming Lemma 14 provides a version of shadow removal adopted to our problem. Informally, it provides us with a set $W \subseteq V(G)$ such that we only have to look for conformal st-cuts that are subsets of $\delta(W)$ to obtain our class assignment; in fact it even shows that for every component C of $G[W]$ either all edges in $\delta(C)$ are part of the cut or no edge of $\delta(C)$ is part of the cut. We will use this fact in Lemma 15 to find a conformal st-cut by branching on which components of $G[W]$ are reachable from s.

More formally, if Z is a set of equations such that $S - Z$ is satisfiable and A is the set of vertices not reachable from s in G minus the conformal st-cut $\mathsf{sep}(\mathsf{ed}(Z))$ (see Lemma 11), then the lemma provides us with a set $W \subseteq V(G)$ such that there is a conformal sA-cut Y' within $\delta(W)$ of size at most $2|Z \setminus \overline{Z}|$ such that there is an assignment $\varphi : V(S) \to \mathbb{Z}_{p^n}$ for the variables in S that satisfies $S - (\overline{Z} \cup \mathsf{eqn}(Y'))$ and agrees with Y'. The main idea behind the proof is the application of Theorem 12 to the set of all walks from s to A in G to obtain the set W and to employ Lemma 13 to obtain the new solution that corresponds to the minimum sA-cut $Y' \subseteq \delta(W)$.

▶ **Lemma 14 (⋆).** *Let S be a simple instance of 2-Lin($\mathbb{Z}_{p^n}$) and let $G = G(S)$. Moreover, let Z be a set of equations such that $S - Z$ is satisfiable, $Y = \mathsf{sep}(\mathsf{ed}(Z))$, let A be the set of all vertices in G that are not reachable from s in $G - Y$, and let $q = |Z \setminus \overline{Z}|$. There is a randomized algorithm that in $\mathcal{O}^*(4^{2q})$ time takes (G, q) as input and returns a set $W \subseteq V(G) \setminus \{s\}$ such that the following holds with probability $2^{-O(q^2)}$. There is a (minimal) sA-cut Y' of size at most $2q$ satisfying:*

1. *every vertex $v \notin W$ is connected to s in $G - Y'$,*
2. *$Y' \subseteq \delta(W)$,*
3. *there is satisfying assignment for $S - (\overline{Z} \cup \mathsf{eqn}(Y'))$ that agrees with Y'.*

Moreover, for every component C of $G[W]$ the following holds:

resume *either $Y' \cap \delta(C) = \emptyset$ or $\delta(C) \subseteq Y'$,*
resume *if $t \in C$, then $\delta(C) \subseteq Y'$,*
resume *if $x_\alpha, x_{\alpha'} \in C$ for some variable x and $\alpha \neq \alpha'$, then $\delta(C) \subseteq Y'$,*
resume *if C contains some x_α for some decided variable x w.r.t. Y', then $\delta(C) \subseteq Y'$.*

The following lemma now uses the set $W \subseteq V(G)$ computed in Lemma 14 to compute a set of at most $2^{\mathcal{O}(k)}$ conformal cuts $\mathcal{Y}$ which contains a cut Y with the following property. If S has a solution Z of size at most k such that $|Z \setminus \overline{Z}| = q$, then there is a subset $Z_2 \subseteq (Z \setminus \overline{Z})$ with for which Y serves as a 2-approximation: more precisely, $|Y| \leq 2|Z_2|$, $(Z \setminus Z_2) \cup Y$ is a solution to S, and $S - ((Z \setminus Z_2) \cup Y)$ admits a satisfying assignment that agrees with Y. Observe that the cost of $S - Y$ is at most $|Z \setminus Z_2| \leq k - |Y|/2$. Now Lemma 7 is an immediate consequence of Lemma 15, i.e. instead of returning the set $\mathcal{Y}$ of conformal cuts, we choose one conformal cut $Y \in \mathcal{Y}$ uniformly at random and output the class assignment corresponding to Y. The main idea behind computing $\mathcal{Y}$ is to use the fact that we only need to consider conformal cuts that are within $\delta(W)$ and this allows us to branch on which components of $G[W]$ are reachable from s (see also Property **4.** in Lemma 14).

▶ **Lemma 15** ($\star$). *Let S be a simple instance of 2-Lin($\mathbb{Z}_{p^n}$) and let k and q with $k \geq q$ be integers. There is a randomized algorithm that takes (S, k, q) as input and returns in $\mathcal{O}^*(2^{\mathcal{O}(k)})$ time a set $\mathcal{Y}$ of at most $2^{\mathcal{O}(k)}$ conformal cuts (each of size at most $2q$), such that if S has a solution Z of size at most k such that $q = |Z \setminus \overline{Z}|$ then with probability at least $2^{-O(k+q^2)}$ there is a cut $Y \in \mathcal{Y}$ with the following property: there is a partition $Z = Z_1 \cup Z_2$ where $\overline{Z} \subseteq Z_1$ and $|Y| \leq 2|Z_2|$, and an assignment that satisfies $S - (Z_1 \cup \mathsf{eqn}(Y))$ and agrees with Y.*

4 Hardness of FPT-Approximation

We complement the approximation algorithm with hardness results: we prove (1) that for finite, commutative, non-trivial rings R, Min-r-Lin(R) is W[1]-hard to FPT-approximate within any constant when $r \geq 3$ and (2) the existence of finite commutative rings R such that Min-2-Lin(R) is W[1]-hard to FPT-approximate within any constant.

Let G denote an arbitrary Abelian group. An expression $x_1 + \cdots + x_r = c$ is an *equation over* G if $c \in G$ and $x_1, \ldots, x_r$ are either variables or inverted variables with domain G. We say that it is an *r-variable equation* if it contains at most r distinct variables. A *cyclic* group is generated by a single element and every finite cyclic group C_n of order n is isomorphic to the additive group of $\mathbb{Z}_n$. Cyclic groups are the building blocks of more complex Abelian groups: the fundamental theorem of finite Abelian groups asserts that every finite Abelian group is a direct sum of cyclic groups whose orders are prime powers.

We consider the natural group-based variant Min-r-Lin(G) of the Min-r-Lin(R) problems in what follows. We first prove that Min-3-Lin(C_p), with p a prime, is not FPT-approximable within any constant if FPT $\neq$ W[1]. Our result is based on a reduction from a fundamental problem in coding theory: the Maximum Likelihood Decoding problem over $\mathbb{Z}_p$ with p prime. Here we are given a matrix $A \in \mathbb{Z}_p^{n \times m}$, a vector $b \in \mathbb{Z}_p^m$, and the goal is to find $x \in \mathbb{Z}_p^n$ such that $Ax = b$ with minimum Hamming weight, i.e. the one that minimizes $k = |\{i \in [n] : x_i \neq 0\}|$. The parameter is k. Theorem 5.1 in [2] proves that for every prime p, the problem MLD_p is W[1]-hard to approximate within any constant factor.

Intuitively, row i in $Ax = b$ is a linear equation $\sum_{j=1}^{n} a_{ij} x_j = b_j$, where $a_{ij}, b_j \in \mathbb{Z}_p$ are coefficients and x_j are variables. There is a straightforward way to subdivide long equations into ternary equations: for example, if we have an equation $x_1 + x_2 + x_3 + x_4 = 1$, we can introduce auxiliary variables y_1, y_2, y_3 and write

$$x_1 + x_2 - y_1 = 0, \quad y_1 + x_3 - y_2 = 0, \quad y_2 + x_4 - y_3 = 0 \quad \text{and} \quad y_3 = 1.$$

When summing up these equations, auxiliary variables cancel out and we obtain $x_1 + x_2 + x_3 + x_4 = 1$. Using this trick, we encode the constraints implied by the row equations of

$Ax = b$ as crisp ternary and unary equations. To encode the objective function, i.e. the fact that we are minimizing the Hamming weight of x, we add soft equations $x_j = 0$ for all $j \in [n]$. This way, breaking a soft equation corresponds to increasing the Hamming weight by 1. This hardness result for C_p, with p prime, can be lifted into the general case via two simple steps: algebraic manipulations allow us to show hardness for C_{p^l}, $l \geq 1$, and this implies hardness for Min-r-Lin(G) by recalling that G is a direct sum of cyclic groups of prime power order. We obtain the following due to the additive group of every ring being Abelian.

▶ **Theorem 16** (⋆). *Let R be a non-trivial finite ring. Min-r-Lin(R) is* W*[1]-hard to* FPT*-approximate within any constant factor when $r \geq 3$.*

We use Theorem 16 to demonstrate that there exist finite commutative rings such that Min-2-Lin(R) is W[1]-hard to FPT-approximate within any constant factor. Let R denote the 16-element polynomial ring $\mathbb{Z}_2[\mathsf{x},\mathsf{y}]/(\mathsf{x}^2,\mathsf{y}^2)$, i.e. the ring with coefficients from $\mathbb{Z}_2$ and indeterminates x,y with x^2 and y^2 factored out. An element $r \in R$ is thus a sum $r_{\mathsf{unit}} + r_x\mathsf{x} + r_y\mathsf{y} + r_{xy}\mathsf{xy}$, where $r_{\mathsf{unit}}, r_x, r_y, r_{xy} \in \mathbb{Z}_2$. The idea is to express equations of length 3 over $\mathbb{Z}_2$ using equations of length 2 over R. We illustrate this by considering an equation $a + b + c = 0$ over $\mathbb{Z}_2$. To express it using binary equations over R, we introduce a fresh variable v and three equations (1) $\mathsf{x}v = \mathsf{xy}b$, (2) $\mathsf{y}v = \mathsf{xy}a$, and (3) $(\mathsf{x}+\mathsf{y})v = -\mathsf{xy}c$. Summing up the first two equations, we obtain $(\mathsf{x}+\mathsf{y})v = \mathsf{xy}(a+b)$. Together with the third one, this implies $\mathsf{xy}(a+b+c) = 0$. On the other hand, any assignment that satisfies $\mathsf{xy}(a+b+c) = 0$ can be extended as $v = \mathsf{x}a + \mathsf{y}b$ to satisfy all three binary equations. With this in mind, it is not too difficult to prove the following with the aid of Theorem 16.

▶ **Theorem 17** (⋆). *Min-2-Lin(R) is* W*[1]-hard to* FPT*-approximate within any constant factor when $R = \mathbb{Z}_p[\mathsf{x}_1,\ldots,\mathsf{x}_k]/(\mathsf{x}_1^2,\ldots,\mathsf{x}_k^2)$, p prime, and $k \geq 2$.*

References

1. Vikraman Arvind and T. C. Vijayaraghavan. The complexity of solving linear equations over a finite ring. In *Proc. 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS-2005)*, pages 472–484, 2005. doi:10.1007/978-3-540-31856-9_39.
2. Arnab Bhattacharyya, Édouard Bonnet, László Egri, Suprovat Ghoshal, Bingkai Lin, Pasin Manurangsi, and Dániel Marx. Parameterized intractability of even set and shortest vector problem. *Journal of the ACM*, 68(3):1–40, 2021. doi:10.1145/3444942.
3. Ian F. Blake. Codes over certain rings. *Information and Control*, 20(4):396–404, 1972. doi:10.1016/S0019-9958(72)90223-9.
4. Édouard Bonnet, László Egri, and Dániel Marx. Fixed-parameter approximability of Boolean MinCSPs. In *Proc. 24th Annual European Symposium on Algorithms (ESA-2016)*, pages 18:1–18:18, 2016. doi:10.4230/LIPICS.ESA.2016.18.
5. Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Transactions on Algorithms*, 11(4):1–28, 2015. doi:10.1145/2700209.
6. Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, Marcin Pilipczuk, and Roohani Sharma. Parameterized complexity classification for interval constraints. In *Proc. 18th International Symposium on Parameterized and Exact Computation (IPEC-2023)*, pages 11:1–11:19, 2023. doi:10.4230/LIPICS.IPEC.2023.11.
7. Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, and Magnus Wahlström. Towards a parameterized approximation dichotomy of mincsp for linear equations over finite commutative rings. *CoRR*, abs/2410.09932, 2024. doi:10.48550/arXiv.2410.09932.
8. Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, and Magnus Wahlström. Almost consistent systems of linear equations. In *Proc. 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-2023)*, pages 3179–3217, 2023. doi:10.1137/1.9781611977554.CH121.

9 Anuj Dawar, Erich Grädel, Bjarki Holm, Eryk Kopczynski, and Wied Pakusa. Definability of linear equation systems over groups and rings. *Logical Methods in Computer Science*, 9(4), 2013. doi:10.2168/LMCS-9(4:12)2013.

10 Reinhard Diestel. *Graph Theory*. Springer, Berlin, Heidelberg, 6th edition, 2025.

11 Cunsheng Ding, Dingyi Pei, and Arto Salomaa. *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific, 1996.

12 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

13 Eduard Eiben, Clément Rambaud, and Magnus Wahlström. On the parameterized complexity of symmetric directed multicut. In *Proc. 17th International Symposium on Parameterized and Exact Computation (IPEC-2022)*, pages 11:1–11:17, 2022. doi:10.4230/LIPICS.IPEC.2022.11.

14 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/A13060146.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.

16 Joseph F. Grcar. How ordinary elimination became Gaussian elimination. *Historia Mathematica*, 38(2):163–218, 2011.

17 Venkatesan Guruswami, Bingkai Lin, Xuandi Ren, Yican Sun, and Kewen Wu. Parameterized inapproximability hypothesis under exponential time hypothesis. In *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC-2024)*, pages 24–35, 2024. doi:10.1145/3618260.3649771.

18 Venkatesan Guruswami, Xuandi Ren, and Sai Sandeep. Baby PIH: parameterized inapproximability of MinCSP. In *Proc. 39th Computational Complexity Conference (CCC-2024)*, pages 27:1–27:17, 2024. doi:10.4230/LIPICS.CCC.2024.27.

19 Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

20 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proc. 24th Annual ACM Symposium on Theory of Computing (STOC-2002)*, pages 767–775, 2002. doi:10.1145/509907.510017.

21 Subhash Khot and Dana Moshkovitz. Candidate hard unique game. In *Proc. 48th Annual ACM Symposium on Theory of Computing (STOC-2016)*, pages 63–76, 2016. doi:10.1145/2897518.2897531.

22 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Flow-augmentation III: complexity dichotomy for boolean CSPs parameterized by the number of unsatisfied constraints. In *Proc. 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA-2023)*, pages 3218–3228, 2023. doi:10.1137/1.9781611977554.CH122.

23 Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolínek. The complexity of general-valued CSPs. *SIAM Journal on Computing*, 46(3):1087–1110, 2017. doi:10.1137/16M1091836.

24 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *Journal of the ACM*, 67(3):1–50, 2020. doi:10.1145/3390887.

25 Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. FPT-approximation for FPT problems. In *Proc. 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA-2021)*, pages 199–218, 2021. doi:10.1137/1.9781611976465.14.

26 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurab, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In *Proc. 40th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-2020)*, pages 2181–2200, 2020. doi:10.1137/1.9781611975994.134.

27 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008. doi:10.1093/COMJNL/BXM048.

28 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM Journal on Computing*, 43(2):355–388, 2014. doi:10.1137/110855247.

29 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. doi:10.1093/ACPROF:OSO/9780198566076.001.0001.

30 George Osipov, Marcin Pilipczuk, and Magnus Wahlström. Parameterized complexity of MinCSP over the point algebra. In *Proc. 32nd Annual European Symposium on Algorithms (ESA-2024)*, volume 308, pages 93:1–93:15, 2024. doi:10.4230/LIPICS.ESA.2024.93.

31 George Osipov and Magnus Wahlström. Parameterized complexity of equality MinCSP. In *Proc. 31st Annual European Symposium on Algorithms (ESA-2023)*, pages 86:1–86:17, 2023. doi:10.4230/LIPICS.ESA.2023.86.

32 Prasad Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC-2008)*, pages 245–254, 2008. doi:10.1145/1374376.1374414.

33 Song Yan. *Number Theory for Computing*. Springer, 2002.

A Simple Algorithm for Trimmed Multipoint Evaluation

Nick Fischer
INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria

Melvin Kallmayer
Goethe University Frankfurt, Germany

Leo Wennmann
University of Southern Denmark, Odense, Denmark

Abstract

Evaluating a polynomial on a set of points is a fundamental task in computer algebra. In this work, we revisit a particular variant called *trimmed* multipoint evaluation: given an n-variate polynomial with bounded individual degree d and total degree D, the goal is to evaluate it on a natural class of input points. This problem arises as a key subroutine in recent algorithmic results [Dinur; SODA '21], [Dell, Haak, Kallmayer, Wennmann; SODA '25]. It is known that trimmed multipoint evaluation can be solved in near-linear time [van der Hoeven, Schost; AAECC '13] by a clever yet somewhat involved algorithm. We give a *simple* recursive algorithm that avoids heavy computer-algebraic machinery, and can be readily understood by researchers without specialized background.

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

Keywords and phrases Algebraic Algorithms, Multipoint Evaluation, Interpolation, LU Decomposition

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.89

Related Version *Full Version*: `https://arxiv.org/abs/2507.00196`

Funding *Nick Fischer*: Partially funded by the Ministry of Education and Science of Bulgaria's support for INSAIT as part of the Bulgarian National Roadmap for Research Infrastructure. Part of this work was done while the author was affiliated with the Weizmann Institute of Science.
Leo Wennmann: Supported by Dutch Research Council (NWO) project "The Twilight Zone of Efficiency: Optimality of Quasi-Polynomial Time Algorithms" [grant number OCEN.W.21.268].

1 Introduction

One of the most fundamental problems in computer algebra is to efficiently evaluate a polynomial P on some set of points, known as the *multipoint evaluation* problem. Besides its importance as one of the most basic algebraic primitives, this problem finds many further applications in computer algebra (such as modular composition and polynomial factorization) and in algorithm design beyond (in diverse fields such as computational geometry, coding theory and cryptography). The inverse task, to *interpolate* a polynomial from a given set of evaluations, is an equally important primitive.

For univariate polynomials, a textbook algorithm [10, 26] solves the multipoint evaluation problem in near-linear time. This algorithm generalizes to multivariate polynomials [20] (via a simple divide-and-conquer method sometimes referred to as *Yates' algorithm* [28]), however, only in the restricted setting where the evaluation points form a cartesian *grid*. Specifically, Yates' algorithm evaluates an n-variate polynomials with individual degree d on all points of a grid

$$Z = \{z_{1,0}, \dots, z_{1,d}\} \times \cdots \times \{z_{n,0}, \dots, z_{n,d}\}$$


© Nick Fischer, Melvin Kallmayer, and Leo Wennmann;
licensed under Creative Commons License CC-BY 4.0



in near-optimal time[1] $O^*((d+1)^n)$. Lifting this restriction to grid points has been the focus of a long and active line of research [18, 23, 15, 25, 6, 24, 2, 1] which, following breakthroughs by Umans [23] and Kedlaya and Umans [15], only recently culminated in an algorithm with almost-optimal running time $(d+1)^{(1+o(1))n}\operatorname{poly}(n, d, \log|\mathbb{F}|)$, for all finite fields $\mathbb{F}$ and for $(d+1)^n$ arbitrary evaluation points, due to Bhargava, Ghosh, Guo, Kumar and Umans [1]. This fully settles the multipoint evaluation problem for *dense* polynomials (over finite fields), but leaves open whether almost-linear time can also be achieved for (some classes of) *sparse* polynomials and evaluation points.

In this paper we focus on one natural such class, called *trimmed* multipoint evaluation, with important applications in the design of exact and parameterized algorithms. Trimmed multipoint evaluation can be solved in near-linear time by an algorithm due to van der Hoeven and Schost [25]. Our contribution is that we make this result accessible to modern algorithm design (beyond computer algebra) by distilling a particularly *simple* recursive algorithm.

Trimmed Multipoint Evaluation

In the trimmed multipoint evaluation problem we focus on the class of n-variate polynomials P with individual degree d *and total degree*[2] D. This is a very natural class of polynomials which, for $D \ll nd$, is exponentially sparser than polynomials with just an individual degree bound. As evaluation points we consider triangular subsets of grids Z defined by

$$\{(z_{1,\ell_1}, \ldots, z_{n,\ell_n}) : \ell \in \{0, \ldots, d\}^n,\ \ell_1 + \cdots + \ell_n \leq D\} \subseteq Z,$$

to which we will informally refer as *trimmed* grids. This is arguably the most naturally matching class of evaluation points. To see this, first observe that the number of relevant (i.e., possibly nonzero) coefficients of P equals exactly the number of grid points. We denote this number by $\binom{n}{\leq D}_d$ (which can be seen as an appropriate generalization of a binomial coefficient called an *extended* binomial coefficient). More importantly, it turns out that the polynomial P is *uniquely* determined by the evaluations on an appropriate trimmed grid, i.e., we can interpolate P given only these evaluations. Trimmed grids are, in a sense, the only sets of evaluation points satisfying this property; see [22].

Applications

Besides being a natural problem in its own right, our interest in trimmed multipoint evaluation stems mainly from its applications in the context of exponential-time algorithms, most notably in a line of research on solving systems of polynomial equations [17, 7, 13, 12]. In this problem the input consists of n-variate polynomials $P_1, \ldots, P_m$ over some finite field $\mathbb{F}_q$ with (total) degree Δ, and the task is to test if all polynomials simultaneously vanish at some point $x \in \mathbb{F}_q^n$, i.e., $P_1(x) = \cdots = P_m(x) = 0$. The initial breakthrough due to Lokshtanov, Paturi, Tamaki, Williams and Yu [17] established that this problem can be solved exponentially faster than brute-force, in time $(q-\epsilon)^n$ for some $\epsilon > 0$, whenever q and Δ are constant. This inspired several follow-up papers aiming to optimize the precise exponential running time [7, 13, 12]. Trimmed multipoint evaluation shows up as a critical subroutine in the algorithms due to

[1] Here and throughout, we write $O^*(\cdot)$ to omit polynomial factors in n and d.

[2] Recall that the *individual* degree d of a polynomial is the largest exponent of a variable in a monomial, whereas the *total* degree D is the largest sum of exponents in a monomial. E.g., $X_1^2X_2$ has individual degree $d = 2$ and total degree $D = 3$.

Dinur [13] (for multilinear polynomials, $d = q - 1 = 1$) and due to Dell, Haak, Kallmayer and Wennmann [12] (for general $d = q - 1 \geq 1$). Solving systems of polynomial equations in turn has many more applications on both the theoretical side – e.g., parity-counting directed Hamiltonian cycles [4] – and the practical side – e.g., the security of several so-called multivariate cryptosystems is based on the hardness of solving quadratic equations [13, 21, 16]. Especially for the latter it could be interesting to achieve simple, implementable algorithms.

In another closely related work, Björklund, Husfeldt, Kaski and Koivisto [5] considered trimmed[3] variants of the Zeta and Möbius transforms to develop fast algorithms for various exponential-time graph problems, such as computing the chromatic number for constant-degree-bounded graphs. The Zeta and Möbius transforms can be regarded as special cases of polynomial multipoint evaluation.[4]

State of the Art

The state of the art for trimmed multipoint evaluation, as mentioned before, is a clever algorithm with near-linear running time $O^*(\binom{n}{\leq D}_d)$ due to van der Hoeven and Schost [25]. Their algorithm is based on the classical concept of Newton interpolation (see e.g. [3]), suitably tailored to the trimmed problem (see also [27, 9, 22, 11, 14] for some more references with a more mathematical point of view). The algorithm also offers two additional benefits: (1) It even solves a strictly more general multipoint evaluation problem on arbitrary downward-closed sets of relevant coefficients and grid points. (2) van der Hoeven and Schost have optimized the lower-order factors achieving an algebraic algorithm with $O(nN \log^2 N \log\log N)$ field operations where $N = \binom{n}{\leq D}_d$. The potentially remaining lower-order improvements even persist in the univariate setting. Thus, all in all, the trimmed multipoint evaluation problem has already been satisfyingly resolved.

The only downside is that van der Hoeven and Schost's algorithm is arguably somewhat intricate – both in the sense that it can be technically demanding to understand, particularly for researchers outside the computer algebra community, and in that it relies on several textbook algebraic primitives, such as efficient conversions between polynomial bases [3] and the use of truncated Fourier transforms as an implementation detail to achieve further improvements.

Our Contribution

Our focus here is *not* to optimize the running time or generality of this state of the art. Instead, given the many exciting algorithmic applications our contribution is to make van der Hoeven and Schost's result for trimmed multipoint evaluation accessible to the algorithms community. We design a *simple* recursive algorithm that is teachable to researchers without any background in computer algebra. Moreover, our algorithm does not rely on any black-box algebraic primitives other than Gaussian elimination. As in [25], we also obtain an equally simple algorithm for the interpolation problem.

Our emphasis on simplicity clashes, however, with the additional benefits (1) and (2): (1) It seems hard to obtain a recursive algorithm for the more general problem, and (2) optimizing lower-order factors would involve dealing with more details. Besides, for exponential-time

[3] In fact, they consider a more general definition of "trimmed" allowing arbitrary downward-closed sets.

[4] Indeed, recall that the zeta transform of a function $f : \{0,1\}^n \to \mathbb{F}$ is defined as $(f\zeta)(X) = \sum_{Y \subseteq X} f(Y)$, where we identify sets $X \subseteq [n]$ with their indicator vectors $X \in \{0,1\}^n$. Consider the n-variate polynomial $P(x_1, \ldots, x_n) = \sum_{Y \subseteq [n]} f(Y) \prod_{i \in Y} x_i$, and observe that $(f\zeta)(X) = P(X)$. Thus, the zeta transform $f\zeta$ can be read off the evaluations of P on the grid $\{0,1\}^n$.

algorithms we are typically anyway not bothered with keeping track of polynomial factors. For these reasons we have decided to stick to the simplest version, resulting in a pleasingly simple 8-line algorithm.

Remark on Terminology

The term "trimmed" multipoint evaluation is inspired by the related algorithms for Zeta and Möbius transforms in [5]. It is not standard in the mathematical literature, where one more often encounters terms like "triangular subsets of tensor product grids". We adopt the "trimmed" terminology here as this paper is primarily intended for the algorithms community.

2 Preliminaries

We write $[n] = \{1, \dots, n\}$. Throughout, let $\mathbb{F}$ be a field and assume that we can evaluate field operations in unit time. For integers n, k, d with $n \geq 0$ and $d \geq 1$ we define the *extended binomial coefficient*

$$\binom{n}{k}_d = |\{\ell \in \{0, \dots, d\}^n : \ell_1 + \cdots + \ell_n = k\}|.$$

That is, $\binom{n}{k}_d$ counts the number of multisubsets of $[n]$ with size k and multiplicity at most d, or equivalently, the number of monomials with total degree k and individual degree at most d in an n-variate polynomial. In the same spirit we define the set

$$\binom{[n]}{k}_d = \{\ell \in \{0, \dots, d\}^n : \ell_1 + \cdots + \ell_n = k\}.$$

As a shorthand, we write $\binom{n}{\leq k}_d = \sum_{i=0}^{k} \binom{n}{i}_d$, and similarly define $\binom{[n]}{\leq k}_d = \bigcup_{i=0}^{k} \binom{n}{i}_d$. Further, we rely on the following generalization of Pascal's triangle; see [8].

▶ **Lemma 1** (Extended Pascal Triangle). *For $n, d \geq 1$ it holds that*

$$\binom{n}{k}_d = \sum_{j=0}^{d} \binom{n-1}{k-j}_d.$$

Throughout we refer to a set $Z = \{z_{1,0}, \dots, z_{1,d}\} \times \cdots \times \{z_{n,0}, \dots, z_{n,d}\}$ of field elements $z_{i,j}$ as a *grid*. To conveniently refer to the grid points we regularly write $Z_\ell = (z_{1,\ell_1}, \dots, z_{n,\ell_n})$ for $\ell \in \{0, \dots, d\}^n$. In particular, the subset of grid points Z_ℓ where ℓ ranges over $\binom{[n]}{\leq D}_d$ constitutes exactly a trimmed grid as introduced before.

3 Trimmed Multipoint Evaluation and Interpolation

In this section, we present a simple, algebraic algorithm for trimmed multipoint evaluation. As it is usually the case, the same algorithmic approach yields a simple algorithm for trimmed interpolation as well.

3.1 Key Ideas

The problem of evaluating univariate polynomials can be defined via the Vandermonde matrix. For $z_0, \dots, z_d \in \mathbb{F}$, define the (square) Vandermonde matrix $V \in \mathbb{F}^{(d+1)\times(d+1)}$ as

$$V = V(z_0, \dots, z_d) = \begin{pmatrix} 1 & z_0 & z_0^2 & \cdots & z_0^d \\ 1 & z_1 & z_1^2 & \cdots & z_1^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_d & z_d^2 & \cdots & z_d^d \end{pmatrix}.$$

Let $a \in \mathbb{F}^{d+1}$ be the coefficient vector of the univariate polynomial $P(X) = \sum_{i=0}^{d} a_i X^i$ of degree d. The evaluation of P at all points z_j with $j \in \{0, \dots, d\}$ can be expressed as the matrix-vector product $V \cdot a = y$ (indeed, each entry is of the form $y_j = \sum_{i=0}^{d} a_i z_j^i = P(z_j)$). Similarly, we can interpolate P from its evaluations $P(z_j)$ using the matrix-vector product $V^{-1} \cdot y = V^{-1} \cdot V \cdot a = a$. The generalization to multivariate polynomials is simple: Let $a \in \mathbb{F}^{(d+1)^n}$ be the coefficient vector of an n-variate polynomial P with individual degree d that we want to evaluate on all grid points in Z. Then the Kronecker product $V(z_{1,0}, \dots, z_{1,d}) \otimes \cdots \otimes V(z_{n,0}, \dots, z_{n,d}) \cdot a = y$ yields the vector $y \in \mathbb{F}^{(d+1)^n}$ of all grid point evaluations. The classical algorithm of Yates [28] allows to compute these evaluations recursively. Notably, this computation does *not* account for the total degree D and automatically results in a running time of $(d+1)^n$ for both multipoint evaluation and interpolation – even in our setting where the total degree D is much smaller than its maximal value nd. Consequently, we need more insights to tailor this approach to the trimmed requirements.

Idea 1: Recursion Scheme. Our goal is to achieve a running time that is linear in $\binom{n}{\leq D}_d$, i.e., in the number of grid points on which we want to evaluate a multivariate polynomial P. As a baseline, we start with a short explanation of Yates' algorithm: write $P(X_1, \dots, X_n) = \sum_{i=0}^{d} P_i(X_1, \dots, X_{n-1}) \cdot X_n^i$ to obtain $d+1$ many $(n-1)$-variate polynomials P_i of degree $D-i$, where each P_i can be seen as a coefficient of a univariate polynomial in X_n. By making $d+1$ many recursive calls of size $\binom{n-1}{\leq D}_d$, we compute the evaluations $P_i(Z_{\ell'})$ for all $\ell' \in \binom{[n-1]}{\leq D}_d$. We obtain all $P(Z_\ell)$ for $\ell \in \binom{[n]}{\leq D}_d$ by evaluating the univariate polynomials $\sum_{i=0}^{d} P_i(Z_{\ell'}) \cdot X_n^i$ at all grid points $z_{n,i}$. However, this does *not* exploit the degrees $D-i$ of the polynomials P_i in the recursive calls, and hence results in the running time of $(d+1)^n$.

Consider the following identity of the extended binomial coefficient that can be derived from Lemma 1

$$\binom{n}{\leq D}_d = \binom{n-1}{\leq D}_d + \binom{n-1}{\leq D-1}_d + \cdots + \binom{n-1}{\leq D-d}_d.$$

In light of this identity, in order to achieve a running time of $O^*\left(\binom{n}{\leq D}_d\right)$ we aim to design an algorithm with $d+1$ recursive calls on $n-1$ variables with total degrees $D, D-1, \dots, D-d$.

Idea 2: LU Decomposition. Our goal is an interleaving of the P_i's into polynomials Q_j satisfying two properties: (1) All evaluations $Q_j(Z_{\ell'})$ can be recursively computed in a call of size $\binom{n-1}{\leq D-j}_d$; in particular, Q_j must have degree $D-j$. (2) Simultaneously, we need to recover all evaluations $P(Z_\ell)$ from the recursively computed evaluations $Q_j(Z_{\ell'})$ for $\ell' \in \binom{[n-1]}{\leq D-j}_d$. Due to the degree restriction for the polynomials Q_j in (1) and evaluations $Q_j(Z_{\ell'})$ in (2), this coincides with a matrix factorization of $V = L \cdot U$ where U is upper-triangular and L lower-triangular, i.e., a LU decomposition.

Algorithm 1 `TrimmedEvaluation`(P).

Input : n-variate polynomial P of individual degree d and total degree D
Output : Evaluations $P(Z_\ell)$ for all $\ell \in \binom{[n]}{\leq D}_d$

if $n = 0$ **then return** the constant P.
Write $P(X_1, \ldots, X_n) = \sum_{i=0}^{d} P_i(X_1, \ldots, X_{n-1}) \cdot X_n^i$.
Compute the LU decomposition $V(z_{n,0}, \ldots, z_{n,d}) = L \cdot U$.
Compute the vector of polynomials

$$\begin{pmatrix} Q_0 \\ \vdots \\ Q_d \end{pmatrix} = \begin{pmatrix} U_{0,0} & \cdots & U_{0,d} \\ & \ddots & \vdots \\ & & U_{d,d} \end{pmatrix} \cdot \begin{pmatrix} P_0 \\ \vdots \\ P_d \end{pmatrix}.$$

for $j = 0, \ldots, d$ **do**
  Recursively call `TrimmedEvaluation`(Q_j) to evaluate $Q_j(Z_{\ell'})$ for $\ell' \in \binom{[n-1]}{\leq D-j}_d$.
for $\ell' \in \binom{[n-1]}{\leq D}_d$ **do**
  Let $k = \min\{d, D - \sum_{j=1}^{n-1} \ell'_j\}$ and compute all evaluations

$$\begin{pmatrix} P\big(Z_{(\ell',0)}\big) \\ \vdots \\ P\big(Z_{(\ell',k)}\big) \end{pmatrix} = \begin{pmatrix} L_{0,0} & & \\ \vdots & \ddots & \\ L_{k,0} & \cdots & L_{k,k} \end{pmatrix} \cdot \begin{pmatrix} Q_0(Z_{\ell'}) \\ \vdots \\ Q_k(Z_{\ell'}) \end{pmatrix}.$$

In more detail: Why is the LU decomposition useful in keeping the degrees of the polynomials Q_j "small"? Let $p = (P_0, \ldots, P_d)^T$ be the vector of polynomials P_i and consider the product $U \cdot p = (Q_1, \ldots, Q_d)^T = q$. Here, the upper triangular structure of U guarantees that each $Q_j = \sum_{i=j}^{d} U_{j,i} \cdot P_i$ has at most degree $D - j$, since the first non-zero entry in the j-th row of U corresponds to polynomial P_j of degree $D-j$ while all other P_i's with (possible) non-zero coefficients have a smaller degree. Additionally, we can also recover $P(Z_\ell)$ from only the recursively computed $Q_j(Z_{\ell'})$ as L has lower triangular structure – appropriately exploiting the fact that $L \cdot q = L \cdot U \cdot p = V \cdot p$.[5] With these ideas in mind, we are now in the position to present the algorithm in detail.

3.2 Trimmed Multipoint Evaluation

Throughout this section, we will prove the following theorem.

▶ **Theorem 2** (Trimmed Multipoint Evaluation). *Let P be an n-variate polynomial over $\mathbb{F}$ with individual degree d and total degree D, and let Z be a grid. The evaluations $P(Z_\ell)$ can be computed for all $\ell \in \binom{[n]}{\leq D}_d$ in time $O^*\big(\binom{n}{\leq D}_d\big)$.*

[5] The analogy to van der Hoeven and Schost's algorithm is as follows: Broadly speaking, their algorithm is based on efficient transformations between not only the coefficient and evaluation-based representations of polynomials, but also involving a third representation based on so-called *Newton polynomials*. In this terminology, the matrix U performs a basis change from the monomial basis to the Newton basis, and the matrix L performs a basic change from the Newton basis to the evaluation basis.

For the proof of Theorem 2, consider the algorithm `TrimmedEvaluation`. Throughout we assume that the algorithm has access to the grid Z and for simplicity, we omit Z in the recursive calls.

▶ **Lemma 3** (Correctness of `TrimmedEvaluation`). *Given an n-variate polynomial P over $\mathbb{F}$ with individual degree d and total degree D,* `TrimmedEvaluation` *correctly computes the evaluations $P(Z_\ell)$ for all $\ell \in \binom{[n]}{\leq D}_d$.*

Proof. Given a polynomial P and a grid Z, we show that `TrimmedEvaluation` correctly computes all evaluations $P(Z_\ell)$ by induction on n.

For $n = 0$, the polynomial P does not depend on any variable. Thus, we correctly return the constant P. For $n \geq 1$, we write $P(X_1, \ldots, X_n) = \sum_{i=0}^{d} P_i(X_1, \ldots, X_{n-1}) \cdot X_n^i$ to obtain $d+1$ many $(n-1)$-variate polynomials $P_0, \ldots, P_d$ of degrees $D, \ldots, D-d$, respectively. Formally, each polynomial P_i is the coefficient of P viewed as a univariate polynomial in X_n. Let $V := V(z_{n,0}, \ldots, z_{n,d})$ be a Vandermonde matrix. It is well-known that V is invertible if and only if all grid points $z_{n,i}$ are distinct, and in this case the LU decomposition $V = L \cdot U$ in Line 3 always exists (see e.g. [19]). Further, let $Q_0, \ldots, Q_d$ as computed in Line 4 of `TrimmedEvaluation`. Note that each polynomial $Q_j(X_1, \ldots, X_{n-1}) = \sum_{i=j}^{d} U_{j,i} \cdot P_i(X_1, \ldots, X_{n-1})$ has degree $D - j$. Indeed, the upper triangular structure of U guarantees that Q_j has at most degree $D - j$, because the first non-zero entry in the j-th row of U corresponds to the polynomial P_j of degree $D - j$ while all other P_i's with (possible) non-zero coefficients have a smaller degree.

Calling `TrimmedEvaluation`(Q_j), we recursively compute the evaluations $Q_j(Z_{\ell'})$ for all $\ell' \in \binom{[n-1]}{\leq D-j}_d$ in Lines 5 and 6. Next, we focus on any iteration ℓ' of the loop in Line 7. Let $k = \min\{d, D - \sum_{j=1}^{n-1} \ell'_j\}$, then we show that we correctly compute the evaluations $P(Z_\ell)$. For each $j \in \{0, \ldots, k\}$, we have

$$\begin{aligned}
\sum_{i=0}^{k} L_{j,i} \cdot Q_i(Z_{\ell'}) &= \sum_{i=0}^{d} L_{j,i} \cdot Q_i(Z_{\ell'}) \\
&= \sum_{i=0}^{d} L_{j,i} \cdot \sum_{m=0}^{d} U_{i,m} \cdot P_m(Z_{\ell'}) \\
&= \sum_{i=0}^{d} z_{n,j}^i \cdot P_i(Z_{\ell'}) \\
&= P\big(Z_{(\ell',j)}\big),
\end{aligned}$$

where the first equality follows from the fact that $L_{j,i} = 0$ whenever $i > k \geq j$. Notably, restricting the computation to k is crucial to ensure that we only use the evaluations $Q_j(Z_{\ell'})$ that we actually computed. This proves that `TrimmedEvaluation` correctly computes the evaluations $P\big(Z_{(\ell',0)}\big), \ldots, P\big(Z_{(\ell',k)}\big)$ in Line 8. Since each Z_ℓ can be expressed as some $Z_{(\ell',j)}$ as above, the loop in Line 7 recovers all evaluations $P(Z_\ell)$. As a result, the algorithm `TrimmedEvaluation` correctly computes all evaluations $P(Z_\ell)$. ◀

Lastly, we prove the running time of `TrimmedEvaluation`.

▶ **Lemma 4** (Running Time of `TrimmedEvaluation`). *The algorithm* `TrimmedEvaluation` *runs in time $O^*\big(\binom{n}{\leq D}_d\big)$.*

Proof. Let $T(n,d,D)$ be the running time of `TrimmedEvaluation`. The time to compute the LU decomposition in Line 4 and the matrix-vector products in Lines 5 and 9 can be bounded by $\mathrm{poly}(n,d)$. Consequently, the running time of the algorithm *without* the recursive calls in Line 7 is $\binom{n}{\leq D}_d \cdot M(n,d)$ for some function[6] $M(n,d) = \mathrm{poly}(n,d)$. Thus, the algorithm admits the following recurrence

$$T(n,d,D) \leq \binom{n}{\leq D}_d \cdot M(n,d) + \sum_{i=0}^{d} T(n-1,d,D-i).$$

By induction on n, we show that $T(n,d,D) \leq \binom{n}{\leq D}_d \cdot M(n,d) \cdot n$. Indeed, it holds that

$$\begin{aligned}
T(n,d,D) &\leq \binom{n}{\leq D}_d \cdot M(n,d) + \sum_{i=0}^{d} T(n-1,d,D-i) \\
&\leq \binom{n}{\leq D}_d \cdot M(n,d) + \sum_{i=0}^{d} \binom{n}{\leq D-i}_d \cdot M(n,d) \cdot (n-1) \\
&= \binom{n}{\leq D}_d \cdot M(n,d) + \sum_{i=0}^{d} \sum_{j=0}^{D-i} \binom{n-1}{j}_d \cdot M(n,d) \cdot (n-1) \\
&\leq \binom{n}{\leq D}_d \cdot M(n,d) + \sum_{j=0}^{D} \binom{n}{j}_d \cdot M(n,d) \cdot (n-1) \\
&= \binom{n}{\leq D}_d \cdot M(n,d) + \binom{n}{\leq D}_d \cdot M(n,d) \cdot (n-1) \\
&= \binom{n}{\leq D}_d \cdot M(n,d) \cdot n
\end{aligned}$$

Therefore, `TrimmedEvaluation` runs in time $O^*\big(\binom{n}{\leq D}_d\big)$. ◀

Combining Lemma 3 and Lemma 4 concludes the proof of Theorem 2.

3.3 Trimmed Interpolation

Throughout this section, we show that a polynomial can be (uniquely) interpolated from its evaluations on the trimmed grid points – using essentially the same algorithmic approach as in Section 3.2.

▶ **Theorem 5** (Trimmed Interpolation). *Let $\alpha_\ell \in \mathbb{F}$ for $\ell \in \binom{[n]}{\leq D}_d$, and let Z be a grid. The unique n-variate polynomial P with individual degree d and total degree D that satisfies $P(Z_\ell) = \alpha_\ell$ for all $\ell \in \binom{[n]}{\leq D}_d$ can be computed in time $O^*\big(\binom{n}{\leq D}_d\big)$.*

Consider the algorithm `TrimmedInterpolation` for the proof of Theorem 5. As before, we assume that the algorithm has access to the grid Z and for simplicity, we omit Z in the recursive calls.

[6] In our context, we are content with bounding $M(n,d) = \mathrm{poly}(n,d)$; however, let us remark that following van der Hoeven and Schost [25] we could achieve a dependence on d that is only polylogarithmic. The main insight is that the L- and U-factors of a Vandermonde matrix are sufficiently structured to support matrix-vector operations in time $O(d(\log d)^{O(1)})$.

Algorithm 2 `TrimmedInterpolation`$((\alpha_\ell)_\ell)$.

Input : Evaluations α_ℓ for all $\ell \in \binom{[n]}{\leq D}_d$
Output : Unique n-variate polynomial P of individual degree d and total degree D such that $P(Z_\ell) = \alpha_\ell$ for all $\ell \in \binom{[n]}{\leq D}_d$

1 **if** $n = 0$ **then return** the constant P.
2 Compute the LU decomposition $V(z_{n,0}, \dots, z_{n,d})^{-1} = L \cdot U$.
3 **for** $\ell' \in \binom{[n-1]}{\leq D-j}_d$ **do**
4 Let $k = \min\{d, D - \sum_{j=1}^{n-1} \ell'_j\}$ and compute all evaluations

$$\begin{pmatrix} \beta_{(\ell',0)} \\ \vdots \\ \beta_{(\ell',k)} \end{pmatrix} = \begin{pmatrix} U_{0,0} & \cdots & U_{0,k} \\ & \ddots & \vdots \\ & & U_{k,k} \end{pmatrix} \cdot \begin{pmatrix} \alpha_{(\ell',0)} \\ \vdots \\ \alpha_{(\ell',k)} \end{pmatrix}.$$

5 **for** $j = 0, \dots, d$ **do**
6 Recursively call `TrimmedInterpolation`$\big((\beta_{(\ell',j)})_{\ell'}\big)$ for $\ell' \in \binom{[n-1]}{\leq D-j}_d$ to interpolate the $(n-1)$-variate polynomial Q_j.
7 Compute the vector of polynomials

$$\begin{pmatrix} P_0 \\ \vdots \\ P_d \end{pmatrix} = \begin{pmatrix} L_{0,0} & & \\ \vdots & \ddots & \\ L_{d,0} & \cdots & L_{d,d} \end{pmatrix} \cdot \begin{pmatrix} Q_0 \\ \vdots \\ Q_d \end{pmatrix}.$$

8 Compute $P(X_1, \dots, X_n) = \sum_{i=0}^{d} P_i(X_1, \dots, X_{n-1}) \cdot X_n^i$.

▶ **Lemma 6** (Correctness of `TrimmedInterpolation`). *Given evaluations α_ℓ for $\ell \in \binom{[n]}{\leq D}_d$,* `TrimmedInterpolation` *correctly interpolates the unique n-variate polynomial P with individual degree d and total degree D such that $P(Z_\ell) = \alpha_\ell$.*

Proof. Given the evaluations α_ℓ for all $\ell \in \binom{[n]}{\leq D}_d$, we show that `TrimmedInterpolation` correctly interpolates the unique n-variate polynomial P by induction on n. If $n = 0$, simply return the constant P as it does not depend on any variable.

For the remainder of the proof we assume $n \geq 1$. Let $V := V(z_{n,0}, \dots, z_{n,d})$. It is well-known that V is invertible if and only if the grid points $z_{n,0}, \dots, z_{n,d}$ are distinct. Additionally, it implies that the LU decomposition in Line 2, namely $V^{-1} = L \cdot U$, always exists (see e.g. [19]).

In the following, we show that $P(Z_\ell) = \alpha_\ell$ for all $\ell \in \binom{[n]}{\leq D}_d$. Note that each Z_ℓ can be expressed as some $Z_{(\ell',j)}$ with $\ell' \in \binom{[n-1]}{\leq D-j}_d$ and $\sum_{i=1}^{n-1} \ell'_i + j \leq D$. Fix such a pair ℓ' and j, and let $k = \min\{d, D - \sum_{i=1}^{n-1} \ell'_i\}$. It holds that

$$\begin{aligned} P\big(Z_{(\ell',j)}\big) &= \sum_{i=0}^{d} z_{n,j}^i \cdot P_i(Z_{\ell'}) \\ &= \sum_{i=0}^{d} V_{j,i} \cdot \sum_{m=0}^{d} L_{i,m} \cdot Q_m(Z_{\ell'}) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=0}^{d} V_{j,i} \cdot \sum_{m=0}^{d} L_{i,m} \cdot \beta_{(\ell',m)} \\
&= \sum_{i=0}^{d} V_{j,i} \cdot \sum_{m=0}^{d} L_{i,m} \cdot \sum_{h=0}^{k} U_{m,h} \cdot \alpha_{(\ell',h)} \\
&= \sum_{i=0}^{d} V_{j,i} \cdot \sum_{h=0}^{k} (V^{-1})_{i,h} \cdot \alpha_{(\ell',h)} \\
&= \sum_{h=0}^{k} I_{j,h} \cdot \alpha_{(\ell',h)} \\
&= \alpha_{(\ell',j)},
\end{aligned}
$$

where $I \in \mathbb{F}^{(d+1)\times(d+1)}$ is the identity matrix. The last equality uses the fact that $j \leq k$. Consequently, `TrimmedInterpolation` correctly interpolates the unique n-variate polynomial P with individual degree d and total degree D such that $P(Z_\ell) = \alpha_\ell$. ◀

As the recursion scheme of `TrimmedInterpolation` and `TrimmedEvaluation` are exactly the same, we refer to Lemma 4 for a proof of the running time of `TrimmedInterpolation`. This concludes the proof of Theorem 5.

References

1 Vishwas Bhargava, Sumanta Ghosh, Zeyu Guo, Mrinal Kumar, and Chris Umans. Fast multivariate multipoint evaluation over all finite fields. In *63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2022)*, pages 221–232. IEEE, 2022. doi:10.1109/FOCS54457.2022.00028.

2 Vishwas Bhargava, Sumanta Ghosh, Mrinal Kumar, and Chandra Kanta Mohapatra. Fast, algebraic multivariate multipoint evaluation in small characteristic and applications. In Stefano Leonardi and Anupam Gupta, editors, *54th Annual ACM Symposium on Theory of Computing (STOC 2022)*, pages 403–415. ACM, 2022. doi:10.1145/3519935.3519968.

3 Dario Bini and Victor Y. Pan. *Polynomial and matrix computations, 1st Edition*, volume 12 of *Progress in theoretical computer science*. Birkhäuser, 1994. URL: https://www.worldcat.org/oclc/312012822.

4 Andreas Björklund and Thore Husfeldt. The parity of directed hamiltonian cycles. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 727–735. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.83.

5 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Trimmed moebius inversion and graphs of bounded degree. *Theory Comput. Syst.*, 47(3):637–654, 2010. doi:10.1007/S00224-009-9185-7.

6 Andreas Björklund, Petteri Kaski, and Ryan Williams. Generalized kakeya sets for polynomial evaluation and faster computation of fermionants. *Algorithmica*, 81(10):4010–4028, 2019. doi:10.1007/S00453-018-0513-7.

7 Andreas Björklund, Petteri Kaski, and Ryan Williams. Solving systems of polynomial equations over GF(2) by a parity-counting self-reduction. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *LIPIcs*, pages 26:1–26:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.26.

8 Richard C. Bollinger. Extended pascal triangles. *Mathematics Magazine*, 66(2):87–94, 1993. doi:10.1080/0025570x.1993.11996088.

9 Carl De Boor and Amos Ron. Computational aspects of polynomial interpolation in several variables. *Mathematics of Computation*, 58(198):705, 1992. doi:10.2307/2153210.

10 Allan Borodin and R. Moenck. Fast modular transforms. *J. Comput. Syst. Sci.*, 8(3):366–386, 1974. doi:10.1016/S0022-0000(74)80029-2.

11 Abdellah Chkifa, Albert Cohen, and Christoph Schwab. High-dimensional adaptive sparse polynomial interpolation and applications to parametric pdes. *Found. Comput. Math.*, 14(4):601–633, 2014. doi:10.1007/S10208-013-9154-Z.

12 Holger Dell, Anselm Haak, Melvin Kallmayer, and Leo Wennmann. Solving polynomial equations over finite fields. In Yossi Azar and Debmalya Panigrahi, editors, *36th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2025)*, pages 2779–2803. SIAM, 2025. doi:10.1137/1.9781611978322.90.

13 Itai Dinur. Improved algorithms for solving polynomial systems over GF(2) by multiple parity-counting. In Dániel Marx, editor, *32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 2550–2564. SIAM, 2021. doi:10.1137/1.9781611976465.151.

14 Nira Dyn and Michael S. Floater. Multivariate polynomial interpolation on lower sets. *J. Approx. Theory*, 177:34–42, 2014. doi:10.1016/J.JAT.2013.09.008.

15 Kiran S. Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011. doi:10.1137/08073408X.

16 Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In Jacques Stern, editor, *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT 1999)*, volume 1592 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 1999. doi:10.1007/3-540-48910-X_15.

17 Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In Philip N. Klein, editor, *28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 2190–2202. SIAM, 2017. doi:10.1137/1.9781611974782.143.

18 Michael Nüsken and Martin Ziegler. Fast multipoint evaluation of bivariate polynomials. In Susanne Albers and Tomasz Radzik, editors, *12th Annual European Symposium on Algorithms (ESA 2004)*, volume 3221 of *Lecture Notes in Computer Science*, pages 544–555. Springer, 2004. doi:10.1007/978-3-540-30140-0_49.

19 Halil Oruç and George M. Phillips. Explicit factorization of the vandermonde matrix. *Linear Algebra and its Applications*, 315(1–3):113–123, 2000. doi:10.1016/s0024-3795(00)00124-5.

20 Victor Y. Pan. Simple multivariate polynomial multiplication. *J. Symb. Comput.*, 18(3):183–186, 1994. doi:10.1006/JSCO.1994.1042.

21 Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In Ueli M. Maurer, editor, *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT 1996)*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 1996. doi:10.1007/3-540-68339-9_4.

22 Tomas Sauer. Lagrange interpolation on subgrids of tensor product grids. *Math. Comput.*, 73(245):181–190, 2004. doi:10.1090/S0025-5718-03-01557-6.

23 Christopher Umans. Fast polynomial factorization and modular composition in small characteristic. In Cynthia Dwork, editor, *40th Annual ACM Symposium on Theory of Computing (STOC 2008)*, pages 481–490. ACM, 2008. doi:10.1145/1374376.1374445.

24 Joris van der Hoeven and Grégoire Lecerf. Fast amortized multi-point evaluation. *J. Complex.*, 67:101574, 2021. doi:10.1016/J.JCO.2021.101574.

25 Joris van der Hoeven and Éric Schost. Multi-point evaluation in higher dimensions. *Appl. Algebra Eng. Commun. Comput.*, 24(1):37–52, 2013. doi:10.1007/S00200-012-0179-3.

26 Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3rd edition, 2013. doi:10.1017/CBO9781139856065.

27 Helmut Werner. Remarks on newton type multivariate interpolation for subsets of grids. *Computing*, 25(2):181–191, 1980. doi:10.1007/BF02259644.

28 Frank Yates. The design and analysis of factorial experiments, 1937.

Separating Two Points with Obstacles in the Plane: Improved Upper and Lower Bounds

Jack Spalding-Jamieson
Independent, Vancouver, Canada

Anurag Murty Naredla
University of Bonn, Germany

Abstract

Given two points in the plane, and a set of "obstacles" given as curves through the plane with assigned weights, we consider the **point-separation** problem, which asks for a minimum-weight subset of the obstacles separating the two points. A few computational models for this problem have been previously studied. We give a unified approach to this problem in all models via a reduction to a particular shortest-path problem, and obtain improved running times in essentially all cases. In addition, we also give fine-grained lower bounds for many cases.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases obstacle separation, point separation, geometric intersection graph, Z_2-homology, fine-grained lower bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.90

Related Version *Full Version*: `https://arxiv.org/abs/2504.17289` [17]

Acknowledgements This work originated from a problem discussion group in the Fall of 2021, whose members collectively contributed to the discussion and development of some of this work's preliminary results in the "arrangement model", as well as a better understanding of the problem as a whole. In addition to the authors, this group included (in alphabetical order) Sam Barr, Therese Biedl, Reza Bigdeli, Anna Lubiw, Jayson Lynch, Karthik Murali, and Graeme Stroud.
In addition, we want to thank Jeff Erickson, Elizabeth Xiao, and Da Wei Zheng for helpful discussions about homology.

1 Introduction

Given points s and t in the plane, and a weighted set of **obstacles** $\mathcal{C}$ defined by simple closed curves (possibly also including their interiors), the (s,t) **point-separation problem** asks for a minimum-weight subset C of $\mathcal{C}$ such that any path from s to t intersects some obstacle in C. Equivalently, s and t are in different connected components of $\mathbb{R}^2 \setminus (\cup_{\gamma \in C} \gamma)$. We say such a subset **separates** s and t. An example of this problem can be found in Figure 1.

This is a natural problem that arises in various scenarios. As a toy application of this problem: Suppose every night your dog runs from his doghouse to your backpack to eat your homework, taking an unpredictable route (making use of windows and doors). You have noticed that your dog will forget about your homework if it smells a treat. You have a number of candidate locations to place treats, and you'd like to ensure your dog is distracted from your homework every day using the fewest treats possible. See Figure 2 for this example. Similar applications also arise when considering security (e.g., replacing the batteries in the fewest number of your dead security cameras to cover all possible paths from the entrance to your bank vault). Additionally, (s,t) point-separation has found an application as a tool for constant-factor approximation of the well-studied APX-hard problem "barrier-resilience" [13].


© Jack Spalding-Jamieson and Anurag Murty Naredla;
licensed under Creative Commons License CC-BY 4.0



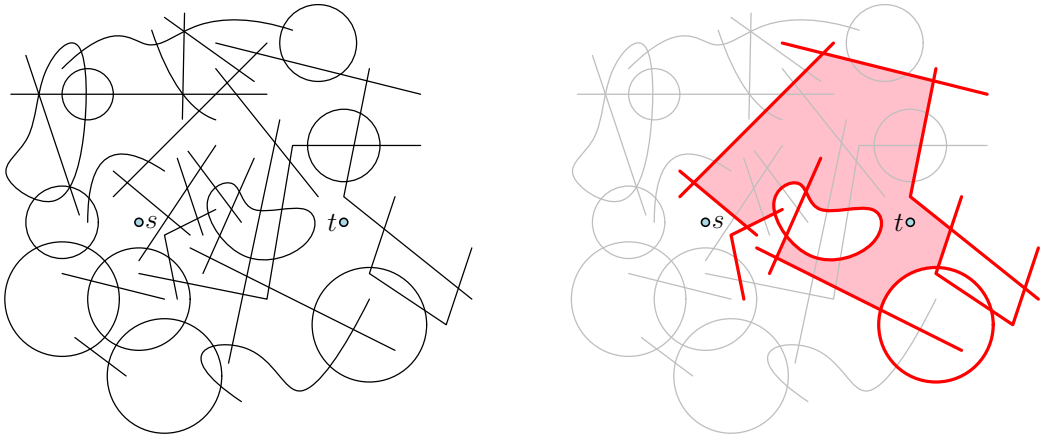


Figure 1 An instance of the (s, t) point-separation problem.

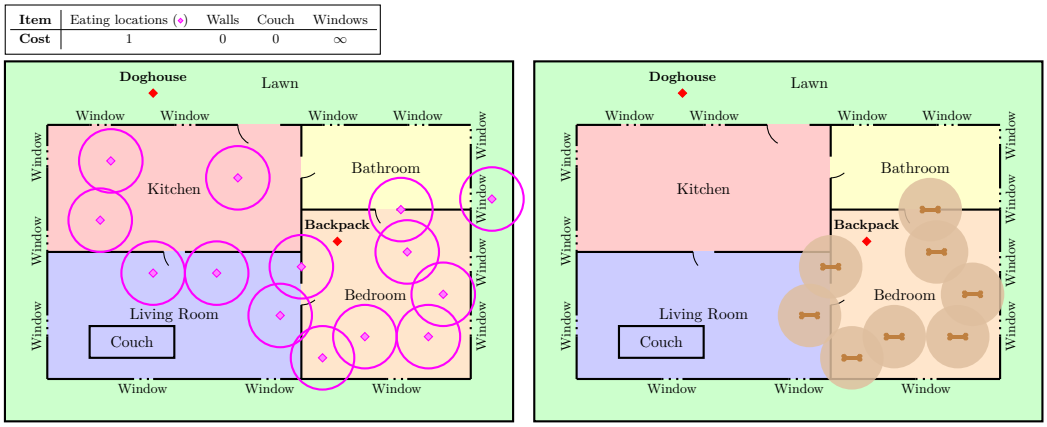


Figure 2 An example of the point-separation problem applied to placing distracting doggy treats around a house, with the minimum-weight solution on the right.

For brevity, we will henceforth say "curve" to mean a simple closed curve in $\mathbb{R}^2 \setminus \{s, t\}$. The algorithmic complexity of the (s, t) point-separation problem depends on the chosen computational model of the obstacles. Previous works use a few different models. We categorize and name the different classes of models used as follows:

- **Specific Obstacle-Type Models**: Assume the set of curves $\mathcal{C}$ takes on a special form with a standard representation, such as a set of disks, circles, or line segments.
- **The Oracle-based Intersection Graph Model**: Assume the existence of several $O(1)$-time oracles that would allow the computation of the **intersection graph** G of $\mathcal{C}$ (the graph whose vertices are exactly the curves $\mathcal{C}$ and whose edges correspond to pairwise intersections of these curves), as well as some additional information for each edge related to s and t (detailed in Section 2).
- **The Arrangement Model**: Assume the arrangement of $\mathcal{C}$ is provided as input, in the form of a plane (multi-)graph, with the faces corresponding to s and t labelled. This is a more graph-theoretic formulation. See Figure 3 for an example of such an arrangement.

Prior works discussing this problem each focused on only one of these paradigms, and the names for each paradigm are our own. We will not assume general position, so no paradigm is strictly more general than the others, since there are arrangements of n obstacles with $\Theta(n^2)$ pairwise intersections, but only $O(n)$ unique intersection points.

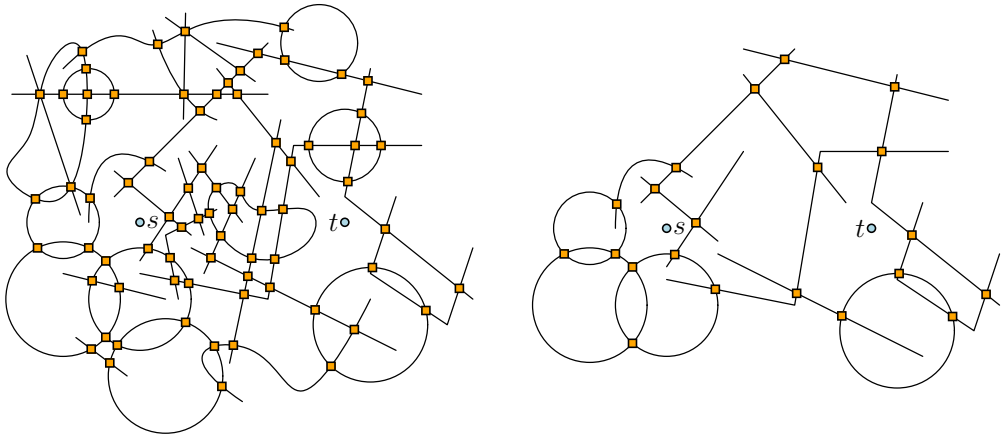


Figure 3 An arrangement representation of the instance in Figure 1, as well as an arrangement of a smaller instance.

In our work, our key method is to frame all models of this problem in terms of a "homology cover". Homology is a very broad topic in algebraic topology, which we will not attempt to summarize in this paper. We aim our paper at a typical computational geometry audience, and we will not assume prior knowledge of homology. We will present the necessary aspects in Section 2.

1.1 Prior Work (Brief)

In this subsection, we very briefly outline some key aspects of prior work. A significantly extended form of this section is in the appendix of the full version.

Most importantly for our methods, Kumar, Lokshtanov, Saurabh and Suri [13, Section 6] describe an algorithm that runs in polynomial time in the *arrangement model*. Their algorithm implicitly makes use of the homology cover (perhaps unintentionally), in the same sense that we will use it. In fact, their algorithm is in some ways analogous to an algorithm of Chambers, Erickson, Fox, and Nayyeri [5] for minimum-cuts (and maximum-flow) in surface graphs. These two algorithms are the main inspiration for our approach to the (s, t) point-separation at a high-level, in *all* models.

1.2 Results and Organization

We obtain improved algorithms for the (s, t)-point separation problem in several cases, which we outline in Table 1. As mentioned earlier, all of our positive results make use of a reduction to a shortest-path problem in the so-called "homology cover". We discuss this formulation in Section 2, and then again more rigorously in the appendix of the full version. Using this reduction, the upper bounds are then given in Section 3.

We also obtain several fine-grained lower bounds in Section 4. Our lower bounds also all have a shared foundation, which will be stated in Theorem 13. The resulting lower bounds are based on a few different hypotheses, and we give their details in the appendix of the full version.

Table 1 The time complexities of our algorithms for various obstacle models. In all cases, $n := |\mathcal{C}|$. For the arrangement model, k denotes the vertex count of the arrangement, and m is the number of obstacle-vertex incidences (so $m \geq k$). The notation $\Omega^*(\cdot)$ hides sub-polynomial factors. The notation † denotes that this is only true for one particular mutual dependence of k, m, and n (and is a slight abuse of notation). ω is the matrix-multiplication exponent ($\omega < 2.371339$ [2]). The lower bound results are given in the full version.

Model	Weights	Old	New	Cond. L.B.
Oracle	Yes	$O(n^3)$ [3]	$\widetilde{O}(n^{(3+\omega)/2})$ (Thm. 4)	$\Omega^*\left(n^2\right)$
Oracle	No	$O(n^3)$ [3]	$O(n^\omega \log n)$ (Thm. 5)	$\Omega^*(n^{3/2})$
Arrangement	Yes	$O(km^2 \lg k)$ [13]	$O(km + k^2 \lg k)$ (Thm. 6)	$\Omega^*\left(km\right)^\dagger$
Segments	No	$O(n^3)$ [3]	$O(n^{7/3} \log^{1/3} n)$ (Cor. 9)	$\Omega^*(n^{3/2})$
Axis-aligned segments	No	$O(n^3)$ [3]	$O(n^2 \log\log n)$ (Cor. 9)	None
Unit Disks	No	$O(n^2 \log^3 n)$ [4]	$O(n^2 \log n)$ (Cor. 9)	None
Disks	No	$O(n^3)$ [3]	$O(n^2 \log n)$ (Cor. 9)	None
$O(1)$-length polylines	No	$O(n^3)$ [3]	$O(n^{7/3} \log^{1/3} n)$ (Cor. 9)	$\Omega^*\left(n^{3/2}\right)$
$O(1)$-length rectilinear polylines	No	$O(n^3)$ [3]	$O(n^2 \log\log n)$ (Cor. 9)	$\Omega^*(n^{3/2})$
Segments	Yes	$O(n^3)$ [3]	$\widetilde{O}(n^{7/3})$ (Cor. 12)	$\Omega^*\left(n^2\right)$
Axis-aligned segments	Yes	$O(n^3)$ [3]	$\widetilde{O}(n^2)$ (Cor. 12)	None
$O(1)$-length polylines	Yes	$O(n^3)$ [3]	$\widetilde{O}(n^{7/3})$ (Cor. 12)	$\Omega^*\left(n^2\right)$
$O(1)$-length rectilinear polylines	Yes	$O(n^3)$ [3]	$\widetilde{O}(n^2)$ (Cor. 12)	$\Omega^*(n^2)$

2 Homology and Obstacles [Informal]

In this section, we will explain the main topological tools we will use to approach the (s, t) point-separation problem. However, this will be an information section, not requiring prior knowledge of any aspect of topology. Rather, we will give an equivalent formulation of the important pieces using simpler tools from computational geometry. We give a more formal treatment in the appendix of the full version. Throughout this section, we will make (implicit) assumptions of general position and such, but the deferred formal treatment does not need these.

At a high-level, there are two main steps to the constructions of our approach. First, we will discuss what it means for a *simple curve* to separate s and t, and what tools exist to classify such curves. Second, we will discuss what it means for a *set of obstacles* to separate s and t. That is, when does the union of the obstacles *contain* a simple curve separating s and t?

The simplest possible case of asking whether a simple curve separates s and t is characterized by the **point-in-polygon** problem, which asks: Given a point p and a (simple) polygon P, is p inside P? In fact, this is equivalent to asking if P separates p and the point

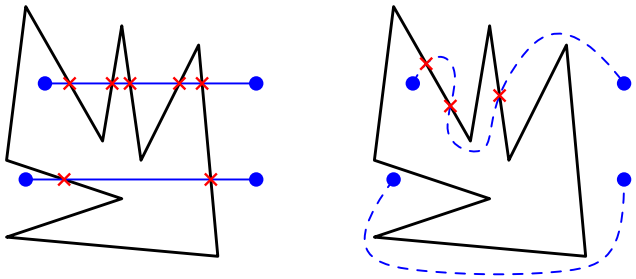


Figure 4 A demonstration of the algorithm for the point-in-polygon problem (left), as well as the problem of testing whether or not a polygon separates two points (middle). On the right, alternative paths for the separation problem are given.

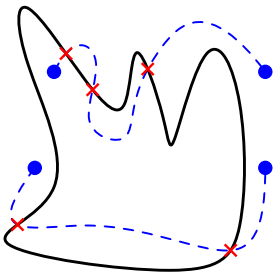
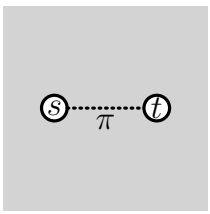


Figure 5 Examples of point-pairs separated or not separated by a given closed path between the pair, and the corresponding intersections of those curves.

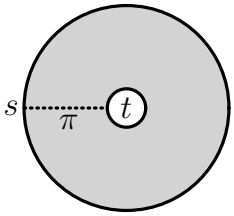
at infinity. There is a folklore algorithm for this problem that takes any ray r starting at p, and counts the number of intersections between r and P. Then, P contains p if and only if the number of intersections is odd. In fact, essentially the same algorithm can be used to test whether or not P separates two points p_1 and p_2. Rather than using a ray, take the segment $\overline{p_1p_2}$. Count the number of intersections between $\overline{p_1p_2}$ and P. The count is odd if and only if P separates p_1 and p_2. Moreover, there is in fact nothing special about rays or line segments in this problem. *Any* (closed) path between p_1 and p_2 would cross P the same number of times, modulo 2. See Figure 4 for examples for these algorithms.

All three of these algorithms are equivalent in a sense. In fact, there is a further generalization: Given two points s and t on the sphere, a simple and closed curve C (not covering s or t), and any $s-t$ path π, C separates s and t if and only if π crosses C an odd number of times (regardless of the choice of π). Since the extended plane is homeomorphic to the sphere, a ray in the plane corresponds to a (simple) path in the sphere. See Figure 5 for an example.

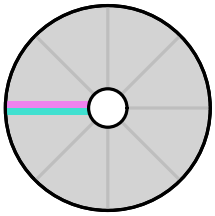
The problems we have discussed so far are all completely static, and the methods do not provide much structure for solving more difficult problems. One of the most common ways to extend the point-in-polygon problem is to fix the polygon (or curve) P, and aim to support fast queries of points. This problem is known as "point-location", and it is well-studied in computational geometry [12]. However, we want a different sort of structure: We have a fixed pair of points s and t, and we wish to classify the curves that separate them. Since we have fixed s and t, we can also fix the path π between them – in most cases, we will use the line segment $\overline{st}$. Then, the problem of classifying curves that separate s and t becomes the problem of classifying curves that cross $\overline{st}$ an odd number of times.



(a) s and t are punctures, π is a path between them.

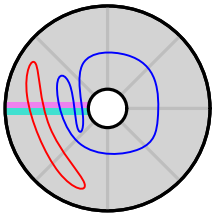


(b) s and t are enlarged to become holes, and the space is an annulus.

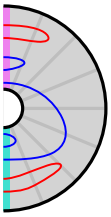


(c) The two sides of π can be coloured to aid later notation.

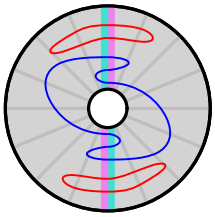
Figure 6 A demonstration of how the (extended) plane with two punctures is homeomorphic to the annulus.



(a) An annulus with a specified path π.



(b) The annulus is sliced open along the specified path π.



(c) Two copies of the sliced annulus are glued together.

Figure 7 The "cut and glue" construction of the homology cover, as well as how it maps a closed curve that separates the two boundaries (blue) and one that does not (red).

It will be helpful for demonstration to make some transformations to the space we work in. That is, we will perform a sequence of homeomorphisms. We start with the extended plane with the two marked points s and t, and the line segment $\overline{st}$ (see Figure 6a). No curves we will be classifying cross s or t, so we may assume there are punctures at s and t. We can enlarge these punctures into holes with a homeomorphism. Next, since we have the extended plane, we can make one of the punctures the outer face, obtaining an annulus (see Figure 6b and Figure 6c). In performing these steps, the line segment $\overline{st}$ becomes a path between the inner and outer boundaries of the annulus.

We now present a method for constructing an important space called the **homology cover**[1]. Take the specified path π between the two boundaries (see Figure 7a) and slice it open (see Figure 7b). Then, create a second copy of the sliced annulus, and glue them together in a way that matches up the orientations of the sliced ends (see Figure 7c).

Alternatively, an equivalent construction is to create two copies of the original space (whether that be the extended plane or the annulus), and use each side of the path from s to t as a (separate) "portal" between the two copies. A more general form of this "portal" idea has been studied in the form of "portalgons" [14, 15], of which the homology cover is essentially a special case. See Figure 8 for an example of this construction.

One important aspect of this construction is that it involves the connection of two identical copies of the annulus (i.e., the extended plane with punctures s and t). In Figure 7, we also show how this construction transforms two curves – one separating the two boundaries, and one not separating them. The structure we will make use of to study curves separating s and t in the plane ultimately stems from an important set of facts:

[1] We use the term "homology cover" to refer to the one-dimensional $\mathbb{Z}_2$-homology cover of the annulus.

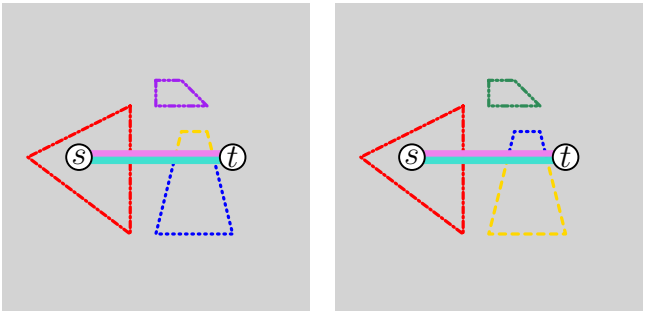


Figure 8 The "portal" construction of the homology cover. Each colour (or dot/dash pattern) is a single closed curve in the homology cover. The two solid lines are the portals.

▶ **Fact 1.** *For a simple curve C in the annulus (or the plane) that gets mapped to the set C' in the homology cover, and a point p along C that gets mapped to corresponding points p_1 and p_2 in the homology cover, the following are all equivalent:*

- *C separates s and t.*
- *C' separates the two boundaries in the homology cover.*
- *C' has one connected component (i.e., it is one closed curve instead of two).*
- *The points p_1 and p_2 are connected by a path through C'.*

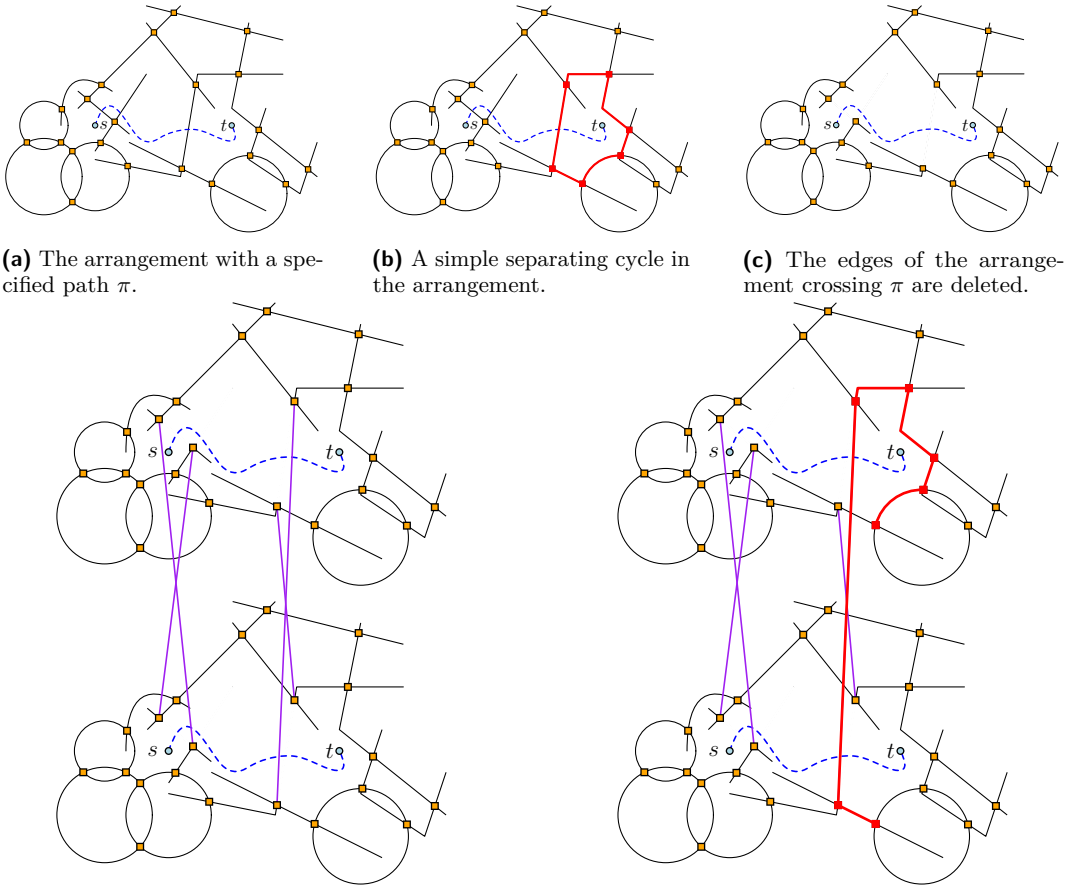
It is this last characterization that will be the most critical for obstacles. In particular, a consequence of this characterization is that if two corresponding points p_1 and p_2 in the homology cover (corresponding in the sense that they are identical points in different copies of the annulus/plane) have a simple path π between them, then that path can be mapped to a closed curve separating s and t.

2.1 Obstacles and the Homology Cover

We now have tools for characterizing *closed curves* that separate s and t. We'd now like to characterize *sets of obstacles* that separate s and t. In other words, we'd like to characterize unions of closed curves (obstacles) that contain a closed curve separating s and t. We'll give two different tools for this, first for the arrangement model, then for the oracle model. For simplicity, we will work exclusively with obstacles that do not *individually* separate s and t (that is, obstacles that get mapped to *two* separate closed curves in the homology cover). We will reduce to this case algorithmically at the start of Section 3.

In the arrangement model of the (s,t) point-separation problem, we are given a plane graph D (the arrangement) with specified faces s and t (can be obtained from points via point-location), and a set of obstacles σ, each given as a connected subgraph. The homology cover has a simple graph-theoretic interpretation in this framework: Take any (simple) dual-path π of faces from s to t – this is will serve as the "portal". Next, create two copies D_1, D_2 of D, each with the faces s and t removed. We will create a combined graph $\overline{D}$ starting from the union of D_1 and D_2. For each edge $e = uv$ used in the dual-path π, delete e from both D_1 and D_2 (inside $\overline{D}$). Denote the corresponding vertices of u and v in each of D_1 and D_2 as u_1, v_1 and u_2, v_2, respectively. Then create new edges u_1v_2 and u_2v_1 in $\overline{D}$. This form of the homology cover is visualized in Figure 9.

With this particular arrangement structure Kumar, Lokshtanov, Saurabh and Suri [13, Section 6] built a graph using one vertex per obstacle-vertex incidence (in each copy of the plane graph). We will build a smaller graph of similar form to theirs. First, for each obstacle



(a) The arrangement with a specified path π.

(b) A simple separating cycle in the arrangement.

(c) The edges of the arrangement crossing π are deleted.

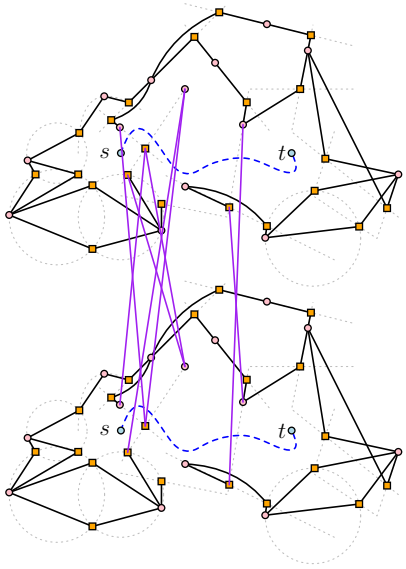
(d) Two copies of are created, and the deleted edges are added as "crossing" edges (in purple).

(e) A path between two copies of the same original vertex that would project onto the simple separating cycle in the original space from Figure 9b.

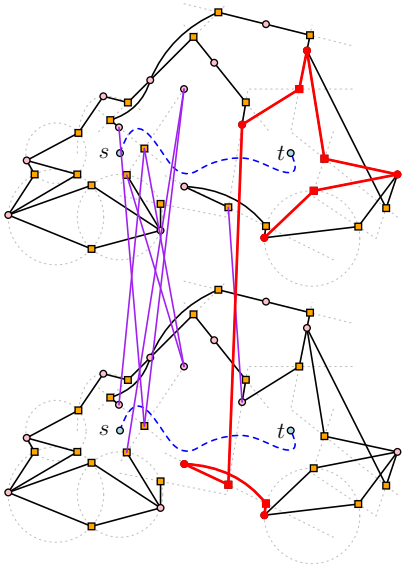
Figure 9 The construction of the homology cover for the second arrangement given in Figure 3.

$C \in \sigma$ (which induces a subgraph of D), pick some arbitrary **canonical point** $x \in C$. Since this choice is arbitrary, it will sometimes be useful to assume it is a specific point – this will primarily be useful for specific obstacle types. Assume that x is also a vertex in the subgraph of D induced by C (and if it is not, modify D so that it is, with either a subdivision or a new degree-1 vertex). Note that since C is connected in D, every other vertex $x' \in C$ is connected to x by only edges in C. Given a plane graph D with faces s and t, a dual-path π, obstacles σ, and canonical points for each obstacle, the **auxiliary graph** H is a bipartite graph constructed as follows:

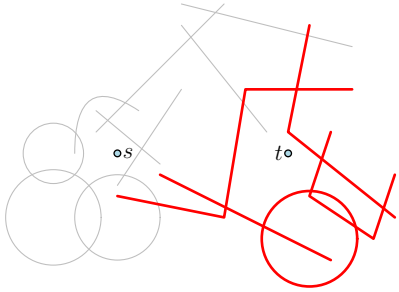
- The first set of vertices are the copies of arrangement vertices in the homology cover. For a vertex $v \in V(D)$, we denote these two copies v^+ and v^-.
- The second set of vertices are the copies of the obstacles/canonical points in the homology cover. For an obstacle $\gamma \in \sigma$, we denote these two copies as $(\gamma, -)$ and $(\gamma, +)$.
- The values $-$ and $+$ in each of the above are **indicator bits**, and can be stored as 0 and 1, respectively.
- A vertex copy v^{b_1} has an edge connecting it to a canonical point copy (γ, b_2) when:
  - $v \in \gamma$ (that is, the point v belongs to the obstacle γ in the arrangement), and...
  - The canonical point x of γ has a path through the edges of γ to v whose intersection



(a) A canonical point is added along each obstacle, and edges are created from the canonical point to each arrangement vertex along the obstacle, subject to the parity conditions.



(b) A shortest-path between the two copies of an obstacle's canonical point. The obstacles whose canonical points lie in this path form a set of obstacles separating s and t.



(c) The set of obstacles induced by the shortest path in Figure 10b.

Figure 10 The construction of the auxiliary graph for the homology cover given in Figure 9.

with the edges of π has size $b_1 + b_2$ modulo 2 (i.e., if $b_1 = b_2$, the size must be even, and if $b_1 \neq b_2$, the size must be odd). Equivalently, the projected connected component of γ containing the canonical point copy (γ, b_2) also contains v^{b_1}.

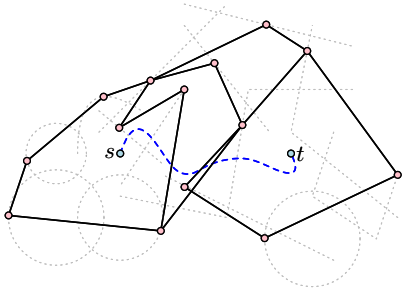
We visualize an example of the auxiliary graph in Figure 10.

The auxiliary graph is useful because the shortest path between any two corresponding vertex copies or (separately) any two corresponding canonical point copies both correspond exactly to the solution to the (s, t) point-separation problem. The high-level construction to prove this fact is as follows:

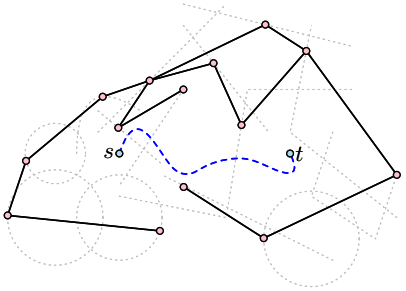
- Suppose there is a set of obstacles C. We'd like to determine when C separates s and t.
- Denote the set of copies of the obstacles in C in the homology cover as C', so that every element of C' is a closed curve representing one of the two connected components of an element of C projected into the homology cover.
- By Fact 1, we deduce that C separates s and t if and only if there is a path from v^+ to v^- along the union of obstacle copies in C', for some arrangement vertex v among pairs of elements in C only.

- Moreover, since each obstacle is connected, we may further assume that any such path visits each obstacle copy at most once, and moreover that it only visits each *obstacle* at most once.
- We may further assume that any such path visits the canonical point copy of each such obstacle copy, by inserting a path to and then from the canonical point copy while the obstacle copy is visited. Note that the resulting path may not be simple in the plane.
- Such paths also directly correspond to paths in the auxiliary graph, using only "obstacle vertices" corresponding to elements of C and the "arrangement vertices" incident to pairs of elements in C.
- With appropriate weights, the problem of finding the minimum-weight set C with this property then reduces to the problem of finding the pair v^+, v^- with the shortest distance in the auxiliary graph, so this is a shortest-path problem!
- A similar argument can show that an alternative algorithmic formulation is to find the pair $(\gamma, -), (\gamma, +)$ with the shortest-distance in the auxiliary graph.

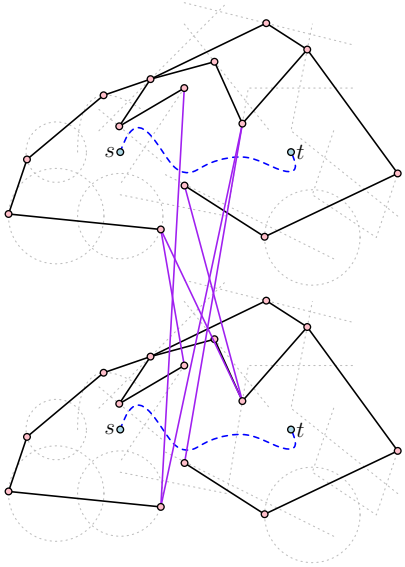
This essentially completes the set of tools necessary for the arrangement model. These arguments are given in more detail in the appendix of the full version.



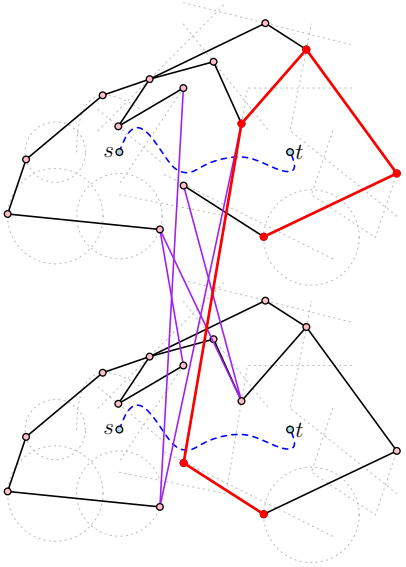
(a) The intersection graph with the path π.



(b) The intersection graph after deleting the edges whose underlying paths cross π an odd number of times.



(c) Two copies of the intersection graph are connected with "crossing" edges (in purple).



(d) A shortest-path in the homology cover between two copies of an obstacle.

Figure 11 How the intersection graph can be transformed into the intersection graph in the homology cover.

For the oracle model, the purpose of the oracles will be to construct the geometric intersection graph of the obstacle *copies* in the homology cover (which we will denote as $\overline{G}$). We will call $\overline{G}$ the **intersection graph in the homology cover**. See Figure 11 for an example. We will present two different constructions of this graph. They are equivalent, and each of them is quite simple, but they will serve two different purposes: The first will clarify which types of oracle queries are necessary, and provide an algorithmic construction. The second will instead build on the tools we have for the arrangement model, proving the correctness of another shortest-paths approach.

Let $\mathcal{C}$ denote the set of obstacles in the plane. We assume the canonical points for each obstacle are given (or implied) – they will be used by the oracle. We also assume some simple $s-t$ path π is fixed – this will also be used by the oracle. In most cases, π will be $\overline{st}$. The first construction of the graph is as follows:

- For each obstacle $\gamma \in \mathcal{C}$, create two vertices $(\gamma, -)$ and $(\gamma, +)$.
- For each pair of obstacles γ_1, γ_2 with canonical points x_1, x_2, and values $b_1, b_2 \in \{-, +\}$, we connect vertices (γ_1, b_1) and (γ_2, b_2) by an edge if there is a path from x_1 to x_2 in $\gamma_1 \cup \gamma_2$ crossing π exactly k times, for some k where $k \equiv b_1 + b_2 \pmod 2$.

Testing for this condition is actually the *only* type of oracle query needed (although we will also require support for the case when $\gamma_1 = \gamma_2$ later). Cabello and Giannopoulos [3] used a larger set of query types, but together they can be used to perform this type by carefully choosing the canonical points and fixing π to $\overline{st}$, so our variant of the oracle model is slightly more general (although practically equivalent).

We'd like a similar algorithmic property for the intersection graph in the homology cover $\overline{G}$ that we have for the auxiliary graph H. The second construction will be what gives us that property, by constructing $\overline{G}$ *from* H:

- For each arrangement vertex v^b in H, let its adjacent vertices be denoted $x_1, \ldots, x_k$. Delete v^b and create a clique over $\{x_1, \ldots, x_k\}$.
- After performing this for every arrangement vertex v^b, the result is *exactly* the intersection graph in the homology cover $\overline{G}$.
- Therefore, the set of vertices visited by the shortest path from some $(\gamma, -)$ to $(\gamma, +)$ (over all possible $\gamma \in \mathcal{C}$, breaking ties arbitrarily) corresponds to a minimum-weight set of obstacles separating s and t.

For specific obstacle types, we will usually work with $\overline{G}$, although we will do it implicitly. Hence, with the tools from this section, we can now discuss algorithms in all three model types as solutions to a certain form of shortest-path problem.

3 Algorithmic Results

In this section, we will devise algorithms for the (s, t) point-separation problem based on our homology cover structures. We will devise algorithms for all three model types: Some that work with the arrangements of curves, some that work with the "oracle model" (the intersection graph in the homology cover), and some that work with specific types of obstacles. Most of our algorithms are fairly simple reductions to various known shortest-path algorithms, greatly simplifying some of the previous approaches to the (s, t) point-separation problem. However, some of them are more involved.

In the previous section, we assumed no obstacle individually separated s and t. Before moving on, we quickly show this is enough:

▶ **Observation 2.** *Let $\mathcal{C}$ be a set of obstacles, and let s and t be points. Suppose an oracle exists that, for an obstacle $\gamma \in \mathcal{C}$, determines whether γ itself separates s and t, all in $O(1)$ time. Let $\mathcal{C}_0$ be the set of all obstacles that do. Then an instance of the weighted*

(unweighted) (s,t) point-separation problem over $\mathcal{C}$ can be reduced to an instance of the weighted (unweighted) (s,t) point-separation problem over $\mathcal{C} \setminus \mathcal{C}_0$ in $O(|\mathcal{C}|)$ time. In particular, such a reduction returns the best solution out of the solved subproblem (if any), and each of the individual obstacles forming solutions (if any).

3.1 General Weighted Obstacles in Oracle Model

With no significant further work, we already obtain some naïve algorithms by using a result in the appendix of the full version (or the constructions in Section 2):

▶ **Theorem 3.** *For points s and t, and a weighted set of obstacles $\mathcal{C}$ that have k pairwise intersections, the (s,t) point-separation problem can be solved in $O(|\mathcal{C}| \cdot k + |\mathcal{C}|^2 \log |\mathcal{C}|)$ time.*

Proof. Compute the intersection graph in the homology cover $\overline{G}$ of $\mathcal{C}$. For each $\gamma \in \mathcal{C}$, compute the shortest path from $(\gamma, -)$ to $(\gamma, +)$ in $\overline{G}$. The smallest such path induces the solution by a result in the appendix of the full version. Each shortest path can be computed in $O(k + |\mathcal{C}| \log |\mathcal{C}|)$ time by Dijkstra's algorithm with a Fibonacci heap [11], and there are $O(|\mathcal{C}|)$ total such paths. ◀

Note that Theorem 3 matches the result of [3], although the algorithm is much simpler. When $k \in \Theta(|\mathcal{C}|^2)$ (i.e., dense intersection graphs), this algorithm runs in $O(|\mathcal{C}|^3)$ time, which could also be obtained by running Floyd-Warshall for APSP instead of Dijkstra. It is not known if the general form of weighted APSP can be solved in "truly" subcubic time (see the appendix of the full version). However, since the weights are applied to vertices, not edges, there is a known faster algorithm (also discussed in the appendix of the full version):

▶ **Theorem 4.** *For points s and t, and a weighted set of obstacles $\mathcal{C}$, the (s,t) point-separation problem can be solved in $\widetilde{O}(|\mathcal{C}|^{(3+\omega)/2})$ time in the oracle model, where $2 \leq \omega < 2.371339$ is the matrix multiplication exponent.*

Proof. Apply the same reduction to APSP as before, but use the algorithm of Abboud, Fischer, Jin, Williams, and Xi [1] for APSP with real vertex weights. ◀

3.2 General Unweighted Obstacles in Oracle Model

We are also interested in the (s,t) point-separation problem for unit weights, which corresponds to shortest paths for unit weights. In this case, we can obtain faster algorithms:

▶ **Theorem 5.** *For points s and t, and an* unweighted *set of obstacles $\mathcal{C}$, the (s,t) point-separation problem can be solved in $O(|\mathcal{C}|^\omega \log |\mathcal{C}|)$ time, where $2 \leq \omega < 2.371339$ is the matrix multiplication exponent.*

Proof. Compute the intersection graph in some homology cover $\overline{G}$ of $\mathcal{C}$, and then run unweighted undirected APSP [16] on $\overline{G}$, for $O(|\mathcal{C}|^\omega \log |\mathcal{C}|)$ total time. ◀

3.3 General Weighted Obstacles in Arrangement Model

We can also obtain results using the arrangement instead of the intersection graph. The simplest of these is a slight modification of Theorem 3:

▶ **Theorem 6.** *For points s and t, a weighted set of obstacles $\mathcal{C}$, and an arrangement D, σ of $\mathcal{C}$, where σ is given as lists of vertices, let $m = \sum_{c \in \sigma} |c|$ be the total number of obstacle-vertex incidences. Then, the (s,t) point-separation problem can be solved in $O(\min(|\mathcal{C}|, |V(D)|) \cdot m + \min(|\mathcal{C}|, |V(D)|)^2 \log \min(|\mathcal{C}|, |V(D)|))$ time.*

Proof. Let $\overline{H}$ be the auxiliary graph in the homology cover, which has $O(m)$ vertices and edges. By a result in the appendix of the full version, it suffices to do one of the following:

- Compute the single-source shortest-path (SSSP) tree from each obstacle in $\overline{H}$.
- Compute the SSSP tree from each arrangement vertex in $\overline{H}$.

Let $k = \min(|\mathcal{C}|, |V(D)|)$. Each SSSP computation can be performed in $O(m + k \log k)$ time, and k such computations suffice to solve the problem. ◀

This theorem in particular is of note because as we will see later, it is essentially optimal assuming the APSP conjecture, at least in the case when $|\mathcal{C}| = \Theta(|V(D)|^2)$ and $m = \Theta(|\mathcal{C}|)$, as we will see in Section 4.

3.4 Restricted Obstacle Classes without Weights

As we will discuss in the appendix of the full version, Chan and Skrepetos [6, 7] studied APSP for several forms of unweighted/undirected geometric intersection graphs. Their intersection graphs are in the plane, but with some extra work it is possible to study intersection graphs in the homology cover, and consequently the unweighted (s, t) point-separation problem.

▶ **Theorem 7.** *For points s and t, and an* unweighted *set of obstacles $\mathcal{C}$. Let $\overline{\mathcal{C}}$ be the set of $2|\mathcal{C}|$ "mapped" obstacles in the homology cover. Let $SI(n, m)$ ("static intersection") be the time complexity for checking if each of n different objects in $\overline{\mathcal{C}}$ intersects any object in some subset $C \subset \overline{\mathcal{C}}$ of size $m = |C|$. Assume $SI(n, m)$ is super-additive, so that $SI(n_1, m_1) + SI(n_2, m_2) \leq SI(n_1 + n_2, m_1 + m_2)$. Then (s, t) point-separation over $\mathcal{C}$ can be solved in $O(nSI(n, n))$ time.*

Proof. Run APSP for the intersection graph in the homology cover via the algorithm of Chan [10, Theorem 2] (see further discussion in the appendix of the full version). ◀

To use this result, we need efficient algorithms for static intersection in the homology cover:

▶ **Lemma 8.** *For points s and t, the following values of $SI(n, n)$ hold for restricted obstacle types in the homology cover:*

Obstacle Class	$SI(n, n)$
General Disks	$O(n \log n)$
Axis-Aligned Line Segments	$O(n \log \log n)$
Arbitrary Line Segments	$O(n^{4/3} \log^{1/3} n)$

The proof of the lemma is left to the appendix of the full version. At a high-level, all cases are first reduced to the planar static intersection problem. In the case of line segments, this becomes the standard planar static intersection problem for line segments. For disks, this is not the case, and the algorithm is more involved.

The combination of these two results give an important corollary:

▶ **Corollary 9.** *For points s and t, the unweighted (s, t) point-separation problem can be solved in $O(n^2 \log \log n)$ time for axis-aligned line segments (or $O(1)$-length rectilinear polylines), $O(n^{7/3} \log^{1/3} n)$ time for line segments (or $O(1)$-length polylines), and $O(n^2 \log n)$ time for general disks or circles that do not contain both s and t.*

3.5 Restricted Obstacle Classes with Weights

We will now present a method for solving the weighted (s,t) point-separation problem using a tool called "biclique covers", which are essentially a tool for a type of graph sparsification.

For a graph $G = (V, E)$, a **biclique** in G is a complete bipartite subgraph ($A \times B \subset E$, where $A, B \subset V$ are disjoint). The **size** of a biclique is the number of vertices it contains ($|A| + |B|$). A **biclique cover** is a collection of bicliques in G covering the edges E, and its **size** is the the sum of all sizes in its bicliques. Biclique covers of geometric intersection graphs in two-dimensions are well-studied. We summarize known results in Table 2.

Table 2 Known results for biclique covers of 2D geometric intersection graphs. All of these results are stated and proven by Chan [8], although essentially all of the steps have appeared in a number of prior works. The notation $\widetilde{O}$ hides logarithmic factors. The notation $\widetilde{O}_k$ further assumes that k is constant. Note that axis-aligned line segment intersection graphs are K_3-free.

Graph Type	Cover Size	Construction Time
line segment intersection	$\widetilde{O}(n^{4/3})$	$\widetilde{O}(n^{4/3})$
Axis-aligned line seg. intersection	$\widetilde{O}(n)$	$\widetilde{O}(n)$
k-clique-free line seg. intersection	$\widetilde{O}_k(n)$	$\widetilde{O}_k(n)$

Biclique covers are useful in our case because they can be used for faster *vertex-weighted* shortest paths. In particular:

▶ **Lemma 10.** *Let $G = (V, E)$ be an n-vertex (undirected) graph with vertex-weights admitting a biclique cover of size $S(n)$ that can be constructed in $T(n)$ time. Then APSP over G can be solved in $O(T(n) + n \cdot S(n) \log(n))$ time.*

We leave the proof to the appendix of the full version.

These results aren't quite enough for the (s,t) point-separation problem, since what we need is actually a biclique cover *in the homology cover*. Fortunately, we can construct this:

▶ **Lemma 11.** *Let s and t be designated points in the plane, and let $\mathcal{C}$ be a set of line segments with $n = |\mathcal{C}|$. Let $\overline{G}$ be the intersection graph in the homology cover, and let G be the intersection graph in the plane. Then $\overline{G}$ has a homology cover of size $\widetilde{O}(n^{4/3})$ that can be found in $\widetilde{O}(n^{4/3})$ time. Moreover, if G contains no k-clique (including if $\mathcal{C}$ is a set of rectilinear segments, in which case G contains no 3-clique), then $\overline{G}$ has a homology cover of size $\widetilde{O}_k(n)$ that can be found in $\widetilde{O}_k(n)$ time.*

We leave the proof of this to the appendix of the full version as well. The proof is similar to that of Theorem 7.

The combination of these results gives the following theorem:

▶ **Theorem 12.** *For a weighted set of obstacles $\mathcal{C}$, and points s, t, the (s,t) point-separation can be solved in $\widetilde{O}(n^{7/3})$ time if $\mathcal{C}$ is a set of line segments, $\widetilde{O}(n^2)$ time if $\mathcal{C}$ is a set of axis-aligned line segments, and $\widetilde{O}_k(n^2)$ time if $\mathcal{C}$ is a set of line segments whose intersection graph has no k-clique. The same bounds hold if each obstacle is an $O(1)$-length polyline among lines of the same properties.*

Note that a biclique cover for $O(1)$-length polylines can be recovered from biclique covers over the individual line segments (adjusted slightly so that the line segments in the same polyline do not overlap).

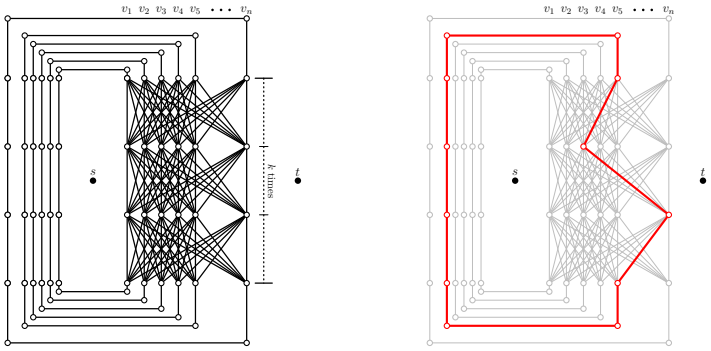


Figure 12 A demonstration of the sub-cubic reduction from min-weight k-walk to the (s,t) point-separation problem with line segments. Empty circles are used to annotate the ends segments.

4 Lower Bounds

In this section, we present several related fine-grained lower bounds for specific cases of the (s,t) point-separation problem. The main intermediate tool is the problem of finding a minimum-weight walk of length k in a directed graph, for a given k (or detecting if any walk of length k exists). All of our lower bounds are based on the following unified theorem:

▶ **Theorem 13.** *For a positively edge-weighted directed graph $G = (V, E)$ with n vertices and m edges with maximum edge-weight W, and an integer k, there exists three sets $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ of $2km + 6m$ obstacles, each with a weight equal to that of some edge in G, so that each of the following properties holds for one set:*

- *All obstacles are line segments.*
- *All obstacles are length-2 polylines, and the total number of unique intersection points of the obstacles is $k + 2(k+1)m + 4m = \Theta(km)$.*
- *All obstacles are length-3 rectilinear polylines.*

Moreover, there are points s and t in the plane so that G has a walk of length k with weight at most w if and only if there is some subset of $\mathcal{C}_i$ (for any $i \in \{1,2,3\}$) that separates s and t and has weight at most $w + (k+6)W$, so long as $w \leq kW$. Furthermore, each of $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ can be constructed in time proportional to their sizes.

Proof. For each property, we will construct a reduction from min-weight k-walk (for fixed k) in a directed edge-weighted graph to point separation, with the stated guarantees. The constructions for each property are essentially the same. We will focus on the line segment case, and explain the modifications afterwards.

The construction is visualized in Figure 12. For each vertex v_i ($i \in \{1, \ldots, n\}$), assign the column $x = i$. For each value $r \in [k+1]$, assign the row $y = r$. For each directed edge $(v_i, v_j) \in E$, with weight w_{v_i,v_j} create k line segments, each also with weight w_{v_i,v_j}. Each should go from the ith column to the jth column, and the rth row to the $(r+1)$th row (for each $r \in [k+1]$). We call these the **edge segments**. Pick s to be some point to the left of the columns, and t to be some point to the right. For the ith column, create a set of $k+6$ rectilinear line segments connecting the top point of the ith column to the bottom, so that the path formed by these line segments lies to the left of s at its x coordinate, ensuring that these line segments do not intersect the corresponding line segments generated for other columns, nor do they cross the minimal orthogonal rectangle containing the previous set of line segments. We call these the **vertex segments**. These two segment types form the entirety of the obstacles, so the stated time complexity follows. We need only prove correctness of the reduction.

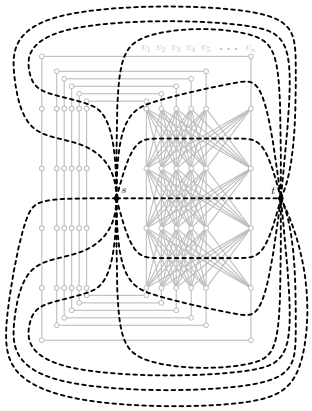


Figure 13 The finite set of (s, t) paths that need to be considered for the reduction of min-weight k-walk to the (s, t) point-separation problem with line segments.

A k-walk through G starting and ending at a vertex $v_i \in V$ corresponds exactly to a connected set of edge segments from the point $(i, 0)$ to (i, k), and both also have the same weight. The forward direction of the reduction follows: Given such a set of edge segments, a set of vertex segments of weight exactly $(k + 6)W$ exists (those for the ith column) so that the union of the two separates s and t, and has the specified weight. For the backwards direction of the reduction: For a set $C \subset \mathcal{C}$ with weight at most $w + (k + 6)W$, if $w \leq kW$, then it is only possible to have one maximal connected set of vertex segments in C. Each of the sets of (potential) obstacles crossed by paths in Figure 13 must have at least one obstacle chosen from them for the whole set to separate s and t, so at least one such maximal connected set of vertex segments must be in C. Hence, the total weight of the remaining obstacles in C is w, and some subset of these obstacles must themselves be a connected set of edge segments (one per row) corresponding exactly to a k-walk in G.

The two modified constructions with polylines are obtained by replacing the edge segments with polylines that have the desired properties. These cases are visualized in Figure 14. ◀

There are a number of fine-grained lower bounds for min-weight k-walk and k-walk detection. In particular, all such bounds hold for *fixed* k and graphs where k-walks and k-cycles coincide (which is itself always true for $k = 3$ in graphs with no self-loops, and is otherwise implied by a property called "k-circle-layered"). In the appendix of the full version, we discuss the following results that follow from known fine-grained bounds and Theorem 13:

If (s, t) point-separation can be solved in time...	for n obstacles that are...	then it would imply a new state-of-the-art algorithm for...
$O(n^{3/2-o(1)})$	weighted line segments	edge-weighted APSP
$O(n^{2-o(1)})$	weighted line segments	minimum-weight k-clique
$O(n^{2-o(1)})$	weighted length-3 rectilinear polylines	minimum-weight k-clique
$O(n^{3/2-o(1)})$	unweighted line segments	max-3-SAT
$O(n^{3/2-o(1)})$	unweighted length-3 rectilinear polylines	max-3-SAT
$O(n^{3/2-o(1)})$	weighted length-2 polylines with $O(\sqrt{n})$ unique intersection points and 3 intersection points per obstacle	edge-weighted APSP

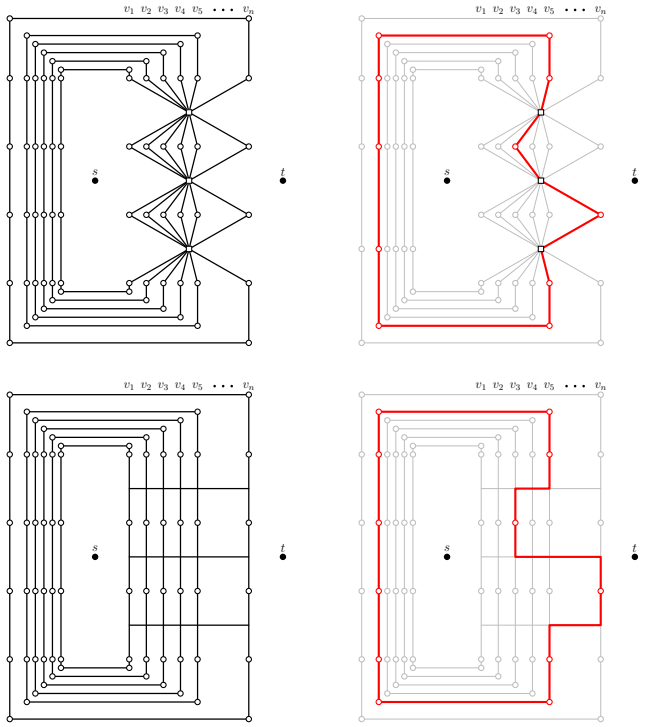


Figure 14 A demonstration of the reduction from min-weight k-walk to the (s,t) point-separation problem for length-2 polylines with few total intersection points, or length-3-rectilinear polylines.

5 Conclusion

We have discussed several upper and lower bounds for the (s,t) point-separation problem in various cases, and we conclude by briefly listing some of the most interesting open problems:

- Is there a near-quadratic algorithm for general weighted obstacles? Can a conditional lower bound (based on any popular hypotheses) excluding this possibility be formulated?
- Can a (truly) sub-quadratic algorithm be devised for disks, unit disks, or axis-aligned segments? Is there a non-trivial fine-grained lower bound in any of these cases?
- Are there faster algorithms or stronger lower bounds for the *unweighted* versions of the problem?

We note that it seems very likely that unit disks would admit a *slightly* sub-quadratic algorithm, since APSP is known to be solvable for unit disk graphs in (essentially) slightly sub-quadratic time [9], and the methods seem as though this method could plausibly be extended to the homology cover.

References

1 Amir Abboud, Nick Fischer, Ce Jin, Virginia Vassilevska Williams, and Zoe Xi. All-pairs shortest paths with few weights per node. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 1956–1964, 2025. doi:10.1145/3717823.3718240.

2 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.

3 Sergio Cabello and Panos Giannopoulos. The complexity of separating points in the plane. *Algorithmica*, 74(2):643–663, 2016. doi:10.1007/s00453-014-9965-6.

4 Sergio Cabello and Lazar Milinković. Two optimization problems for unit disks. *Comput. Geom.*, 70-71:1–12, 2018. doi:10.1016/j.comgeo.2017.12.001.

5 Erin W. Chambers, Jeff Erickson, Kyle Fox, and Amir Nayyeri. Minimum cuts in surface graphs. *SIAM J. Comput.*, 52(1):156–195, 2023. doi:10.1137/19M1291820.

6 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 590–598. ACM, 2007. doi:10.1145/1250790.1250877.

7 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. doi:10.1137/08071990X.

8 Timothy M Chan. Finding triangles and other small subgraphs in geometric intersection graphs. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1777–1805. SIAM, 2023. doi:10.1137/1.9781611977554.CH68.

9 Timothy M Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *27th International Symposium on Algorithms and Computation (ISAAC 2016)*, pages 24–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ISAAC.2016.24.

10 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in geometric intersection graphs. *J. Comput. Geom.*, 10(1):27–41, 2019. doi:10.20382/JOCG.V10I1A2.

11 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

12 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

13 Neeraj Kumar, Daniel Lokshtanov, Saket Saurabh, and Subhash Suri. A constant factor approximation for navigating through connected obstacles in the plane. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 822–839. SIAM, 2021. doi:10.1137/1.9781611976465.52.

14 Maarten Löffler, Tim Ophelders, Rodrigo I. Silveira, and Frank Staals. Shortest Paths in Portalgons. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry (SoCG 2023)*, volume 258 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2023.48.

15 Tim Ophelders, Maarten Löffler, Rodrigo I Silveira, and Frank Staals. Shortest paths in portalgons. *Journal of Computational Geometry*, 15(2):174–221, 2024.

16 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995. doi:10.1006/JCSS.1995.1078.

17 Jack Spalding-Jamieson and Anurag Murty Naredla. Separating two points with obstacles in the plane: Improved upper and lower bounds, 2025. doi:10.48550/arXiv.2504.17289.

Near-Optimal Differentially Private Graph Algorithms via the Multidimensional AboveThreshold Mechanism

Laxman Dhulipala
University of Maryland, College Park, MD, USA

Monika Henzinger
Institute of Science and Technology Austria, Klosterneuburg, Austria

George Z. Li
Carnegie Mellon University, Pittsburgh, PA, USA

Quanquan C. Liu
Yale University, New Haven, CT, USA

A. R. Sricharan
Faculty of Computer Science, UniVie Doctoral School Computer Science DoCS, University of Vienna, Austria

Leqi Zhu
University of Manitoba, Canada

Abstract

Many differentially private and classical non-private graph algorithms rely crucially on determining whether some property of each vertex meets a threshold. For example, for the k-core decomposition problem, the classic peeling algorithm iteratively removes a vertex if its induced degree falls below a threshold. The sparse vector technique (SVT) is generally used to transform non-private threshold queries into private ones with only a small additive loss in accuracy. However, a naive application of SVT in the graph setting leads to an amplification of the error by a factor of n due to composition, as SVT is applied to *every vertex*. In this paper, we resolve this problem by formulating a novel *generalized* sparse vector technique which we call the *Multidimensional AboveThreshold (MAT) Mechanism* which generalizes SVT (applied to vectors with one dimension) to vectors with *multiple* dimensions. When applied to vectors with n dimensions, we solve a number of important graph problems with better bounds than previous work.

Specifically, we apply our MAT mechanism to obtain a set of improved bounds for a variety of problems including k-core decomposition, densest subgraph, low out-degree ordering, and vertex coloring. We give a *tight* local edge differentially private (LEDP) algorithm for k-core decomposition that results in an approximation with $O(\varepsilon^{-1} \log n)$ additive error and *no* multiplicative error in $O(n)$ rounds. We also give a new $(2+\eta)$-factor multiplicative, $O(\varepsilon^{-1} \log n)$ additive error algorithm in $O(\log^2 n)$ rounds for any constant $\eta > 0$. Both of these results are asymptotically tight against our new lower bound of $\Omega(\log n)$ for any constant-factor approximation algorithm for k-core decomposition. Our new algorithms for k-core decomposition also directly lead to new algorithms for the related problems of densest subgraph and low out-degree ordering. Finally, we give novel LEDP differentially private defective coloring algorithms that use number of colors given in terms of the *arboricity* of the graph.

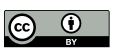
2012 ACM Subject Classification Security and privacy; Theory of computation

Keywords and phrases differential privacy, abovethreshold, densest subgraph

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.91

Related Version *Full Version*: `https://arxiv.org/abs/2508.02182`

Funding *Monika Henzinger and A. R. Sricharan*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (MoDynStruct, No. 101019564) and the Austrian Science Fund (FWF) grant DOI



© Laxman Dhulipala, Monika Henzinger, George Z. Li, Quanquan C. Liu, A. R. Sricharan, and Leqi Zhu;
licensed under Creative Commons License CC-BY 4.0

10.55776/Z422 and grant DOI 10.55776/I5982. *Laxman Dhulipala and George Z. Li*: supported by NSF award number CNS-2317194. *Quanquan C. Liu*: supported by a Google Academic Research Award and by an NSF award number CCF-2453323.

1 Introduction

Designing and analyzing algorithms that satisfy *differential privacy* [23], the de facto standard for protecting user data, is a tricky task, with few general techniques. Among them, the *sparse vector technique (SVT)* is a well-known approach to make certain *threshold-based* computations privacy-preserving. This typically involves showing that the computation can be reduced to a sequence of black-box invocations of private *AboveThreshold* mechanisms, followed by some post-processing. Each such mechanism allows one to iteratively (and adaptively) query whether some real-valued property of the data is above some threshold, until the first "yes" response, at which point no further queries are possible.

While SVT was originally presented as a technique for *sequential* computations, recent work on concurrent composition [54] show that it is possible to interleave queries to different (AboveThreshold) mechanisms with (worst-case) privacy loss proportional to the number of concurrently queried mechanisms. However, this can be prohibitively large for applications (e.g., in parallel and distributed settings) where there is a large amount of concurrency and, in this case, one must exploit structural properties of the instance to obtain better guarantees.

To analyze these situations, we introduce a new mechanism, called *Multidimensional AboveThreshold (MAT)*, which allows one to query, *in parallel*, whether a number of different real-valued properties of the data are above some specified thresholds. We identify a condition on the sensitivity of the submitted queries that allows one to use MAT with far less privacy loss versus concurrent composition. By analyzing various non-private algorithms through the lens of our mechanism, we obtain private algorithms with substantially improved accuracy guarantees for a variety of important, recently studied graph problems.

All of our algorithms apply in the stringent, *local* model of (edge) differential privacy *(LEDP)*, where each user (here, a vertex in the graph) interactively discloses information about its data (here, its incident edges) in multiple *rounds* of communication with an *untrusted*[1] curator (or server), which produces the output. The *transcript* of messages sent throughout the interaction must be differentially private.

We now describe our contributions and the main related work in more detail (see Table 1 for a summary of our results). In the following, all graphs are simple (no self-loops), undirected, and contain n vertices. Furthermore, all mentioned additive error upper bounds hold *with high probability*, i.e., $1 - 1/n^c$, for any constant $c > 0$.

k-Core Decomposition. At a high level, core numbers measure the relative importance of vertices in a graph. Formally, given a graph G, the *core number* (or, *coreness*) of a vertex v in G is the largest $k_G(v) \in \mathbb{N}$ (or $k(v)$ if G is clear from context) for which there exists a subgraph H of G *containing* v such that every vertex in H has induced degree $\geq k_G(v)$. In the *k-core decomposition problem*, the goal is to output core numbers of all vertices in G.[2]

[1] The curator may accidentally leak their messages, e.g., due to security breaches. However, it is not malicious. This is referred to as "honest but curious" in the literature.

[2] We note that the k-core decomposition problem in the literature is sometimes used to refer to the problem of computing a hierarchy of *k-cores*, which are subgraphs of G induced by vertices with core number at least k [49]; note that it is not possible to emit such a hierarchy privately, so we (and prior work) focus on privately emitting core numbers for every vertex.

This problem has recently received much attention from the database and systems communities due its applications in community detection [1, 28, 31], clustering and data mining [16, 51], and other machine learning and graph analytic applications [37, 18, 19]; for a more comprehensive survey, see [43]. Due to the rapid increase of applications using sensitive user data, designing private algorithms for this problem is becoming increasingly important.

An algorithm is a (ϕ, ζ)-*approximation* for the k-core decomposition problem if, for every vertex v in the given graph, the *estimate* of the core number of v, as returned by the algorithm, lies between $k(v) - \zeta$ and $\phi \cdot k(v) + \zeta$. Dhulipala et al. [20] gave the first differentially private $(2+\eta, O(\varepsilon^{-1}\log^3 n))$-approximation algorithm for k-core decomposition in the ε-LEDP model, which runs in $O(\log n)$ rounds. They left it as an open question whether their multiplicative and additive approximation factors can be improved.

Our contributions. We answer their question in the affirmative. In particular, we give an *exact* (i.e., 1-approximate) algorithm and a $(2+\eta)$-approximate algorithm, both with $O(\varepsilon^{-1}\log n)$ additive error, which run in $O(n)$ and $O(\log^2 n)$ rounds, respectively.

▶ **Theorem 1.** *For $\varepsilon > 0$, there is an exact, $O(n)$-round ε-LEDP algorithm for the k-core decomposition problem, which has $O(\varepsilon^{-1}\log n)$ additive error, with high probability.*

▶ **Theorem 2.** *For $\varepsilon > 0$ and* constant *$\eta > 0$, there is a $(2+\eta)$-approximate, $O(\log^2 n)$-round ε-LEDP algorithm for the k-core decomposition problem, which has $O(\varepsilon^{-1}\log n)$ additive error, with high probability.*

Furthermore, we show that the additive errors of our algorithms are tight, by proving a matching lower bound of $\Omega(\varepsilon^{-1}\log n)$ on the additive error of any constant-approximation algorithm in the *centralized* model (EDP), where the algorithm has full access to user data (here, the input graph). Since the centralized model is less restricted than the local model, the same bound holds in the LEDP model, regardless of round complexity. We also show that interactivity is needed if one wants an exact algorithm with polylogarithmic additive error, by proving a lower bound of $\Omega(\sqrt{n})$ on the additive error of any exact, 1-round algorithm, when ε is a constant.

▶ **Theorem 3.** *For $\varepsilon > 0$ and $\phi \geq 1$, any ϕ-approximate ε-EDP algorithm for the k-core decomposition problem in the centralized model has $\Omega((\varepsilon\phi)^{-1}\log n)$ additive error, with constant probability.*

▶ **Theorem 4.** *For* constant *$\varepsilon > 0$, any 1-round, exact ε-LEDP algorithm for the k-core decomposition problem has $\Omega(\sqrt{n})$ additive error, with constant probability.*

Densest Subgraph. Another measure of importance, defined on *subsets* of vertices in a graph, is density. Formally, the *density* of a set of vertices S in a graph G, denoted $\rho_G(S)$ (or simply $\rho(S)$), is the ratio of the number of edges in the subgraph induced by S and the number of vertices in S, i.e., $|E(S)|/|S|$. In the *densest subgraph* problem, the goal is to output a set of vertices with the maximum density in a given graph.

This problem has been studied in the (centralized) private [29, 48], LEDP [21, 20], dynamic [13, 50], streaming [3, 12, 27, 46], and distributed [4, 53] settings; for a recent survey on densest subgraph and its variants, see [39].

An algorithm is a (ϕ, ζ)-*approximation* to the densest subgraph problem if the set of vertices S returned by the algorithm has density $\rho(S) \geq \rho^*/\phi - \zeta$, where ρ^* is the maximum density. In the ε-LEDP setting, Dhulipala et al. [20] gave a $(4+\eta, O(\varepsilon^{-1}\log^3 n))$-approximation for the densest subgraph problem. More recently, and concurrently with

this work, Dinitz et al. [21] gave a $(2+\eta, O((\varepsilon\eta)^{-1}\log^2 n))$-approximation algorithm, which runs in $O(\log n)$ rounds, in the ε-LEDP setting, and as well as an $(1, O(\varepsilon^{-1}\sqrt{\log \delta^{-1}}\log n))$-approximation, which runs in $O(n^2 \log n)$ rounds in the (ε, δ)-LEDP setting. On the other hand, there is a lower bound of $\Omega(\phi^{-1}\sqrt{\varepsilon^{-1}\log n})$ on the additive error of any ϕ-approximate algorithm for densest subgraph that satisfies (ε, δ)-edge differential privacy in the centralized model [29, 48] (when $\delta \leq n^{-O(1)}$).

Our contributions. We give a condition on when a (private) (ϕ, ζ)-approximation algorithm for k-core decomposition can be transformed into a (private) $(2\phi, \zeta)$-approximation algorithm for densest subgraph. We show that our algorithms for k-core decomposition satisfy the condition, obtaining the following improved bounds for densest subgraph.

▶ **Theorem 5.** *For $\varepsilon > 0$, there is a 2-approximate, $O(n)$-round ε-LEDP algorithm for the densest subgraph problem, which has $O(\varepsilon^{-1}\log n)$ additive error, with high probability.*

▶ **Theorem 6.** *For $\varepsilon > 0$ and* constant *$\eta > 0$, there is a $(4+\eta)$-approximate, $O(\log^2 n)$-round ε-LEDP algorithm for the densest subgraph problem, which has $O(\varepsilon^{-1}\log n)$ additive error, with high probability.*

We also give a simple 1-round algorithm using the randomized response technique together with brute-force, exponential time search. We believe this algorithm serves as an interesting example of what a lower bound for 1-round algorithms must be able to reason about.

▶ **Theorem 7.** *For $\varepsilon > 0$, there is an exact, 1-round ε-LEDP algorithm for the densest subgraph problem, which has $O((1+\varepsilon^{-1})\sqrt{n})$ additive error, with high probability.*

Low Out-Degree Ordering. A *low out-degree orientation* of a (undirected) graph is an orientation of the edges of the graph such that the out-degree of any vertex is minimized. A *low out-degree ordering* is a permutation of the vertices, which implicitly encodes a low out-degree orientation, i.e., each edge is oriented from its earlier endpoint to its later endpoint (with respect to the permutation). In the *low out-degree ordering problem*, the goal is to output a low out-degree ordering of a given graph.

Differentially private low out-degree ordering was introduced by [20] as a way to release an implicit solution to the low out-degree orientation problem. The latter has attracted a lot of attention in the non-private setting (see, e.g., [9, 13, 38, 53] and references therein) and has been used to improve the running time of algorithms for matching [47, 33], coloring [34], and subgraph counting [7, 8, 42].

The *degeneracy* of a graph is the maximum core number of any vertex in the graph. It is known that degeneracy is equal to the minimum out-degree that is achievable through any ordering. Hence, we state the out-degree obtained through our algorithms in terms of α, the degeneracy of the input graph. An algorithm is a (ϕ, ζ)-*approximation* for the low out-degree ordering problem if the algorithm's output encodes an orientation where each vertex has out-degree at most $\leq \phi \cdot \alpha + \zeta$. In the ε-LEDP model, Dhulipala et al. [20] gave a $(2+\eta, O(\varepsilon^{-1}\log^3 n))$-approximation to the low out-degree ordering problem, which runs in $O(\log n)$ rounds.

Our contributions. We improve the multiplicative factor and the additive error by giving an exact algorithm and a $(2+\eta)$-approximate algorithm, both with additive error $O(\varepsilon^{-1}\log n)$, which run in $O(n)$ and $O(\log n)$ rounds, respectively.

▶ **Theorem 8.** *For $\varepsilon > 0$, there is an exact, $O(n)$-round ε-LEDP algorithm algorithm for the low out-degree ordering problem, which has $O(\varepsilon^{-1}\log n)$ additive error, with high probability.*

▶ **Theorem 9.** *For $\varepsilon > 0$ and* constant *$\eta > 0$, there is a $(2+\eta)$-approximate, $O(\log^2 n)$-round ε-LEDP algorithm for the low out-degree ordering problem, which has $O(\varepsilon^{-1} \log n)$ additive error, with high probability.*

Defective Coloring. A (vertex) *coloring* of a graph G is an assignment of labels (or, *colors*) from the set $\{1, \ldots, n\}$ to the vertices of G. A coloring is (c, d)-*defective* if it uses at most c different colors, and each vertex has at most d neighbors with the same color; d is the *defectiveness* of the coloring. In the *defective coloring* problem, the goal is to output a (c, d)-defective coloring of a given graph with small c and d.

This problem (especially the case $d = 0$) has been widely studied in a variety of settings including the (centralized) private [15], dynamic [11, 34, 35, 52], distributed [30, 32, 45], and streaming [2, 10, 6] settings. In the ε-edge differential privacy setting, [15] recently gave the first (c, d)-defective coloring algorithm with $c = O(\Delta/\log n + \varepsilon^{-1})$ and $d = O(\log n)$, where Δ is the maximum degree of the input graph. They also showed that $d = \Omega(\log n/(\log c + \log \Delta))$ defectiveness is necessary if $\leq c$ colors are used. Although not explicitly stated, we believe their coloring algorithm can be also adapted to the ε-LEDP setting with the same bounds.

Our contributions. Using the approximate low out-degree ordering results and MAT, we give new algorithms for defective coloring under LEDP. In particular, the number of colors used by our algorithms scales with the arboricity α instead of the maximum degree. In small-arboricity graphs, the number of colors used by our algorithm can be significantly smaller than that of [15].

▶ **Theorem 10.** *For $\varepsilon > 0$, there is an $O(n)$-round ε-LEDP algorithm for the defective coloring problem that uses $O(1 + \alpha\varepsilon/\log n)$ colors, where α is the arborocity of the input graph, and has $O(\varepsilon^{-1} \log n))$ defectiveness, with high probability.*

▶ **Theorem 11.** *For $\varepsilon > 0$, there is an $O(\log^2 n)$-round ε-LEDP algorithm for the defective coloring problem that uses $O(\alpha\varepsilon \log n + \log^2 n)$ colors, where α is the arborocity of the input graph, and has $O(\varepsilon^{-1} \log n))$ defectiveness, with high probability.*

Efficient centralized algorithms. All of the bounds in the above theorems are stated for ε-LEDP algorithms but also are algorithms in the centralized model with the same approximation and additive error guarantees. All of our algorithms based on MAT can be implemented in the centralized model in near-linear time if we lose an additional multiplicative factor of $1 + \eta$ for any given constant $\eta > 0$. This nearly matches the best running times for sequential non-private k-core decomposition algorithms, while maintaining a near-optimal utility for private algorithms.

Concurrent, Independent Work. The recent concurrent, independent work of Dinitz et al. [21] provides a large set of novel results on (ε, δ)-DP, ε-LEDP, and (ε, δ)-LEDP densest subgraph, improving on the approximation guarantees of previous results. Their work is based on a recent result of Chekuri, Quanrud, and Torres [14] and the parallel densest subgraph algorithm of Bahmani, Kumar, and Vassilvitskii [4]. They offer improvements in the bounds in the central DP setting via the private selection mechanism of Liu and Talwar [41]. Note that their techniques are different from those presented here; furthermore, any c-approximate algorithm for densest subgraph translates to a $2c$-approximate algorithm for finding the degeneracy of the input graph (which is the value of the maximum k-core). Thus, no densest subgraph algorithm can obtain a better than 2-approximation of the maximum k-core value.

Subsequent Work. Using the MAT framework of this work, Dinitz et al. [22] gave new algorithms for node-differentially private bipartite matching, improving upon the results of [36]. They also gave the first algorithms for maximum matchings in general graphs under local edge-differential privacy, also crucially using the MAT framework. Complementing their algorithms, they show a matching lower bound for edge-private maximum matchings. This follow up work highlights the wide applicability of the MAT framework, giving nearly tight algorithms for various combinatorial problems under differential privacy.

Table 1 Summary of error bounds in the local model. Each upper bound is for a mechanism satisfying (ε, δ)-edge differential privacy and holds with high probability. An asterisk in the citation indicates that an algorithm is (ε, δ)-LEDP instead of ε-LEDP. Lower bounds are for ε-differential privacy and hold with constant probability.

Problem	Apx. Factor	Additive Error	Rounds	Citation
Core Decomposition	1	$O(\varepsilon^{-1}\log n)$	$O(n)$	Theorem 1
	1	$\Omega(\sqrt{n})$ (constant ε)	1	Theorem 4
	$2+\eta$	$O(\varepsilon^{-1}\log n)$	$O(\log^2 n)$	Theorem 2
	$2+\eta$	$O(\varepsilon^{-1}\log^3 n)$	$O(\log n)$	[20]
	ϕ	$\Omega((\varepsilon\phi)^{-1}\log n)$	any	Theorem 3
Densest Subgraph	1	$O(\varepsilon^{-1}\sqrt{\log(\delta^{-1})}\log n)$	$O(n^2\log n)$	[21]*
	1	$O((1+\varepsilon^{-1})\sqrt{n})$	1	Theorem 7
	ϕ	$\Omega(\phi^{-1}\sqrt{\varepsilon^{-1}\log n})$	any	[29, 48]
	2	$O(\varepsilon^{-1}\log n)$	$O(n)$	Theorem 5
	$2+\eta$	$O(\varepsilon^{-1}\eta^{-1}\log^2 n)$	$O(\eta^{-1}\log n)$	[21]
	$4+\eta$	$O(\varepsilon^{-1}\log^3 n)$	$O(\log n)$	[20]
	$4+\eta$	$O(\varepsilon^{-1}\log n)$	$O(\log^2 n)$	Theorem 6
Low Out-degree Ordering	1	$O(\varepsilon^{-1}\log n)$	$O(n)$	Theorem 8
	$2+\eta$	$O(\varepsilon^{-1}\log n)$	$O(\log^2 n)$	Theorem 9
	$2+\eta$	$O(\varepsilon^{-1}\log^3 n)$	$O(\log n)$	[20]
	# Colors	**Defectiveness**		
Defective Coloring	$O(1+\varepsilon\alpha/\log n)$	$O(\varepsilon^{-1}\log n)$	$O(n)$	Theorem 10
	$O(\varepsilon\alpha\log n+\log^2 n)$	$O(\varepsilon^{-1}\log n)$	$O(\log^2 n)$	Theorem 11
	$O(\varepsilon^{-1}+\Delta/\log n)$	$O(\log n)$	2	[15]
	c	$\Omega(\frac{\log(n)}{\log c+\log\Delta})$	any	[15]

2 Technical Overview

MAT. Our new *Multidimensional AboveThreshold* (MAT) mechanism generalizes the AboveThreshold mechanism [24] to support a d-dimensional vector of queries. Conceptually, one may think of MAT as running d copies of AboveThreshold in parallel, one per dimension. More precisely, suppose that the input dataset for (single-dimensional) AboveThreshold was from some domain $\mathcal{X}$, then the input for our mechanism is $X = (x_1, \ldots, x_d) \in \mathcal{X}^d$, a d-tuple of elements from $\mathcal{X}$. Our mechanism takes as input an adaptively chosen sequence $\vec{f_1}, \vec{f_2}, \ldots$ of *query vectors* and a *threshold vector* $\vec{T} = (T_1, \ldots, T_d)$. Each query vector specifies d *queries*, one for each dimension of the dataset, i.e., $\vec{f_i} = (f_{i,1}, \ldots, f_{i,d})$ and query $f_{i,j} : \mathcal{X} \to \mathbb{R}$ is on element x_j. The goal is to (privately) return, after each query vector $\vec{f_i}$, a response vector $\vec{a_i} \in \{\bot, \top\}^d$ indicating whether each query $f_{i,j}$ in the vector has *crossed corresponding*

threshold T_j, i.e., $\vec{a}_i = (a_{i,1}, \ldots, a_{i,d})$ and $a_{i,j}$ indicates whether $f_{i,j}(x_j) \geq T_j$ ($\perp$ for "no", $\top$ for "yes"). After some query $f_{i,j}$ has crossed its threshold, then all future queries $f_{i',j}$, for $i' > i$, on element x_j are ignored (e.g., the response is always $\perp$). It is assumed that new query vectors are submitted until each element has had a query that has crossed its corresponding threshold. In a nutshell, MAT allows one to infer, for each element x_j, an estimate $\tilde{r}(j)$ of the index $r(j)$ of the *first* query $f_{r(j),j}$ on x_j that crosses the threshold T_j.

Sensitivity condition. We define a fairly technical condition on the *sensitivity* of the queries, generalizing the ℓ_1 sensitivity in the 1-dimensional case, such that the error of MAT can be bounded in terms of the sensitivity. For intuition, we first describe a simple case of our sensitivity condition.

Suppose that two datasets $X = (x_1, \ldots, x_d)$ and $X' = (x'_1, \ldots, x'_d)$ are neighboring only if they differ on at most $k \ll d$ coordinates, i.e., there exists a set $S \subseteq [d]$ of at most k indices such that $x_i = x'_i$ for all $i \notin S$ and $x_i \sim x'_i$ for all $i \in S$. Furthermore, suppose that the absolute difference of each query (on the appropriate element in X and X') is bounded by 1, i.e., for any query vector $\vec{f} = (f_1, \ldots, f_d)$, each query f_j satisfies $|f_j(x_j) - f_j(x'_j)| \leq 1$. Since MAT runs d copies of AboveThreshold in parallel and the queries are adaptive, applying concurrent composition [54] in this case would predict that the privacy loss is proportional to d. However, by coupling the outputs of the queries on the indices *not* in S (where X and X' are identical), it seems fairly intuitive that the actual privacy loss *should* be proportional to k. Hence, as the error of AboveThreshold is inversely proportional to the privacy loss parameter, to obtain ε-differential privacy, we *should* have at most $k \ll d$ times the original error of AboveThreshold in this setting, i.e., $O(k\varepsilon^{-1} \log n)$ with high probability.

More generally, for an arbitrary neighboring relation and arbitrary queries, roughly speaking, we define the sensitivity D as the worst-case (over pairs of neighboring datasets X and X') sum of the maximum absolute difference in the outputs of the queries (on X and X'); see Section 4 for a formal definition. In particular, for the simple case described above, $D = k$. Using this definition, we formalize the above intuition and prove that MAT guarantees an error of $O(D\varepsilon^{-1} \log n)$ in our applications, with high probability.

Applications of MAT. We apply MAT to get new private algorithms for the k-core decomposition, densest subgraph, low out-degree ordering, and defective coloring problems. In each case, given a graph $G = (V, E)$ with n vertices, we show that some non-private algorithm can be simulated via MAT, for suitable definitions of "dataset" and "queries".

As a simple example, for the core decomposition problem, a dataset $X = (x_u : u \in V)$ is an n-tuple containing the set of neighbors x_u of each vertex u in G. (Hence, the domain $\mathcal{X}$ is the set of all subsets of V.) A query $f_S : \mathcal{X} \to \mathbb{R}$ counts the number of neighbors of the vertex that are *not* among some fixed set of vertices $S \subseteq V$, shifted by some fixed amount i, and "mirrored" by n, i.e., $f_S(x) = n - (|x \setminus S| - i)$. (Notice that $f_S(x_u) \geq n$ means that vertex u has at most i neighbors that are not among S.) In this case, the absolute difference of each query on two neighboring datasets (i.e., graphs that differ by at most an edge) is at most 1 and there are at most two elements that differ for two neighboring datasets (namely, at the endpoints of the differing edge). Hence, the simple case mentioned above applies and MAT guarantees an error of $O(\varepsilon^{-1} \log n)$, with high probability.

For core decomposition, we apply MAT to analyze a variant of the classic peeling algorithm of Matula and Beck [44], which iteratively peels (removes) the lowest degree vertex in the graph. In our variant of the classical algorithm, we view the algorithm as running n iterations of a *peeling process*. In the ith iteration, the algorithm repeatedly peels all vertices with

(induced) degree less than i until none are left. It is known that, at the end of the ith iteration, a vertex is alive (has not been peeled) if and only if it is in the i-core. It can be shown that the peeling process can be simulated by MAT via the queries mentioned above. Hence, with some post-processing, we can obtain core numbers with no multiplicative factor and the optimal (up to a constant factor against our lower bound) additive error of $O(\varepsilon^{-1} \log n)$, with high probability. Similarly, we apply MAT to analyze the more round-efficient $(2+\eta)$-approximate core decomposition algorithm of [20], obtaining optimal error, with high probability.

Using our new core decomposition algorithms, we achieve improved additive error bounds for both the densest subgraph and low out-degree ordering problems. In particular, we show that the 2-approximate densest subgraph with additive error $O(\varepsilon^{-1} \log n)$ is contained in the subgraph consisting of all vertices with the maximum estimated core numbers (as we run the peeling algorithm, it can be shown that these vertices necessarily induce a subgraph with sufficiently high minimum degree). Furthermore, all of our core decomposition algorithms directly return a low out-degree ordering with the same multiplicative and additive errors as the original core decomposition algorithms by taking the order in which the nodes are peeled.

We also apply MAT to defective coloring by using MAT to greedily pick colors for each vertex from the available colors that have not yet been occupied by too many neighbors. In particular, each dimension represents a vertex and color pair consisting of all vertices in the graph and all available colors. For dimension j representing pair (v, c), the queries ask whether the number of neighbors of v colored with c exceeded our defectiveness threshold $O(\varepsilon^{-1} \log n)$. If it has, we will remove c from v's available set of colors. Our low round coloring algorithm follows a similar structure, and is inspired by the non-private algorithm of [34] and uses our low out-degree ordering algorithm to implement a separate set of available colors for each of $O(\log^2 n)$ groups of vertices of similar core numbers.

Local lower bound. Our lower bound for 1-round, exact core decomposition in the local model is obtained by reduction to a problem in the *centralized* model, for which there is a strong lower bound. Specifically, it is known that, to privately answer $\Theta(n)$ *random* inner product queries on a secret $\{0,1\}^n$ vector, most responses need to have $\Omega(\sqrt{n})$ additive error [17]. Recently, [26] gave an elegant lower bound on the additive error of 1-round triangle counting algorithms in the local model using similar techniques.

In our case, we define a class of "query graphs", one per inner product query, in which the core number of a *fixed* vertex x is (roughly) the answer to the query. In addition to x, there are some *secret* vertices (and other vertices). For each secret vertex v, the existence of the edge $\{v, x\}$ is private information (which depends on the secret vector). Crucially, all neighborhoods of x and the secret vertices in *any possible query graph* appear in two specific query graphs, namely, ones corresponding to the all-ones and all-zeros query vectors, respectively. The neighborhoods of the remaining vertices do not depend on the secret vector.

Our approach to solving the inner product problem is now as follows. The centralized algorithm first simulates the 1-round local core decomposition algorithm on the two fixed graphs to determine the messages that x and the secret vertices would send in *any* query graph and saves these messages. Subsequently, when answering a query, the centralized algorithm *reuses* the saved messages for x and the secret vertices (without further privacy loss) when it simulates the core decomposition algorithm on the corresponding query graph. This allows it to answer many queries correctly.

Originally, we had a more complex, direct argument. We briefly mention it here to give an idea of what is going on "under the hood" of the reduction. Specifically, we showed that, on a class of random graphs (similar to the query graphs instantiated in our reduction), most

transcripts of a 1-round core decomposition algorithm have the property that, conditioned on seeing the transcript, the coreness of some (fixed) vertex still has high variance, $\Omega(n)$. This implies that the standard deviation of the additive error is $\Omega(\sqrt{n})$. We prefer the reduction argument, as it is simpler and gives a stronger result.

One-round algorithm for densest subgraph. In our single-round exact densest subgraph algorithm, the vertices send the server a noisy version of the incidence matrix of the graph by using randomized response. In particular, each vertex sends its row of the incidence matrix and, for each bit, it sends the actual bit with probability $p = e^{\varepsilon}/1 + e^{\varepsilon}$; the flipped bit is sent with probability $1 - p$. Using this, the server can create an unbiased estimator $\tilde{E}(S)$ for the number of edges $E(S)$ inside any *fixed* subset of vertices S. The key observation is that the standard deviation σ of the estimated density, i.e., $\tilde{E}(S)/|S|$, is $O(1 + \varepsilon^{-1})$ and the estimated density is distributed binomially. Hence, by a Chernoff bound, the estimated density is $\Omega(\sqrt{\log t})$ standard deviations away from the actual density with probability at most $1/t$. By taking a union bound over all S, we can upper bound the total error over *all* (exponentially many) subsets by $\alpha = O(\sigma\sqrt{n})$ with high probability. Hence, a *brute-force*, exponential-time search, which computes the maximum $\tilde{E}(S)/|S|$ over all subsets S, finds a subset whose density is at most α from the maximum density, with high probability. The union bound appears to be necessary, but it is conceivable that there are more clever search strategies that do not require looking at exponentially many subsets S.

3 Differential Privacy Background

We give some basic definitions and background on differential privacy in this section, see [25] for more details. Since we will only be working with graphs, we will state these results in terms of edge differential privacy for graph algorithms; we will also use the terms differential privacy and edge differential privacy interchangeably. Edge-neighboring graphs is a well-studied model for differential privacy (see e.g. [40] for a survey of such results) and protects the privacy of edges with highly sensitive information like disease transmission graphs.

▶ **Definition 12.** *Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are* edge-neighboring*, denoted by $G_1 \sim G_2$, if $V_1 = V_2$ and $|E_1 \oplus E_2| = 1$ (i.e., they have the same vertex set and they differ by exactly one edge).*

▶ **Definition 13.** *We use $\mathcal{G}$ to denote the set of undirected graphs. An algorithm $\mathcal{M} : \mathcal{G} \to \mathcal{Y}$ is* (ε, δ)-edge differentially private *((ε, δ)-edge dp) if for all edge-neighboring graphs $G \sim G'$ and every $S \subseteq \mathcal{Y}$, we have*

$$\Pr[\mathcal{M}(G) \in S] \leq \exp(\varepsilon) \cdot \Pr[\mathcal{M}(G') \in S] + \delta.$$

If $\delta = 0$, we say $\mathcal{M}$ is ε-edge differentially private (ε-edge DP). When $\mathcal{M}$ is defined over $\mathbb{N}$ or $\mathbb{R}$ instead, it is called ε-differentially private (ε-DP).

Next, we define two important properties of DP: composition and post-processing.

▶ **Lemma 14.** *Let $\mathcal{M}_1 : \mathcal{G} \to \mathcal{Y}_1$ and $\mathcal{M}_2 : \mathcal{G} \to \mathcal{Y}_2$ be ε_1- and ε_2-edge differentially private mechanisms, respectively. Then $\mathcal{M} : \mathcal{G} \to \mathcal{Y}_1 \times \mathcal{Y}_2$ defined by $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2)$ is $(\varepsilon_1 + \varepsilon_2)$-edge differentially private.*

▶ **Lemma 15.** *Let $\mathcal{M} : \mathcal{G} \to \mathcal{Y}$ be an ε-edge differentially private mechanism. and $f : \mathcal{Y} \to \mathcal{Z}$ be an arbitrary randomized mapping. Then $f \circ \mathcal{M} : \mathcal{G} \to \mathcal{Z}$ is ε-edge differentially private.*

The ℓ_1-*sensitivity* of a function $f : \mathcal{G} \to \mathbb{R}^d$, denoted $\Delta_1(f)$, is the supremum of the quantity $\|f(x) - f(x')\|_1 = \sum_{i=1}^{d} |f(x)_i - f(x')_i|$, over all neighboring $G, G' \in \mathcal{G}$. The *Laplace* distribution (centered at 0) with *scale* $b > 0$ has probability density function $f(x) = \frac{1}{2b} \exp(-|x|/b)$. We write $X \sim \mathsf{Lap}(b)$ or just $\mathsf{Lap}(b)$ to denote a random variable X that is distributed according to the Laplace distribution with scale b.

▶ **Fact 3.1.** *If random variable $X \sim \mathsf{Lap}(b)$, then, for any $\beta > 0$, $\Pr[|X| > b \log(1/\beta)] \leq \beta$.*

▶ **Fact 3.2** (Theorem 3.6 in [25])**.** *Let $f : \mathcal{G} \to \mathbb{R}^d$ be any function, let $\epsilon > 0$, and let, for each $i \in [d]$, $X_i \sim \mathsf{Lap}(\Delta_1(f)/\epsilon)$. Then the* Laplace mechanism *$\mathcal{A}(x) = f(x) + (X_1, \ldots, X_d)$ is ϵ-differentially private.*

3.1 Local Edge Differential Privacy (LEDP)

The local model of differential privacy studies the setting where all private information is kept private by the individual parties, i.e. there exists no trusted curator. For graphs, the local model is called local edge differential privacy [20]. We now define this formally.

▶ **Definition 16.** *Let G be a graph. An ε-**local randomizer** $R : \mathbf{a} \to \mathcal{Y}$ for vertex v is an ε-edge differentially private algorithm that takes as input the set of its neighbors $N(v)$ in G, represented by an adjacency list $\mathbf{a} = (b_1, \ldots, b_{|N(v)|})$.*

Note that the information released via local randomizers is public to all nodes and the curator. The curator performs some computation on the released information and makes the result public via sending a message to all vertices. The overall computation is formalized via the notion of the transcript.

▶ **Definition 17.** *Let G be a graph. A* transcript *π is a vector consisting of 4-tuples $(S_U^t, S_R^t, S_\varepsilon^t, S_Y^t)$ – encoding the set of vertices chosen, the set of local randomizers assigned, the set of privacy parameters of the randomizers, and the set of randomized outputs produced – for each round t. Let S_π be the collection of all transcripts and S_R be the collection of all randomizers. Let STOP denote a special character indicating that the local randomizer's computation halts. A* protocol *is an algorithm $\mathcal{A} : S_\pi \to (2^{[n]} \times 2^{S_R} \times 2^{\mathbb{R}^{\geq 0}} \times 2^{\mathbb{R}^{\geq 0}}) \cup \{STOP\}$ mapping transcripts to sets of vertices, randomizers, and randomizer privacy parameters. The length of the transcript, as indexed by t, is its round complexity. Given $\varepsilon > 0$, a randomized protocol $\mathcal{A}$ on graph G is* ε-local edge differentially private *if the algorithm that outputs the entire transcript generated by $\mathcal{A}$ is ε-edge differentially private on graph G.*

The definition is difficult to parse, but it naturally corresponds to the intuition of vertices with private information communicating with an untrusted curator (represented by the protocol) so that the curator can compute some output that depends on the private data of the vertices (see also the discussion in [21]). At the beginning, the curator only knows the vertex set V and each vertex knows its incident edges, i.e., its neighborhood. In each round, a chosen subset of the vertices performs some computation using local randomizers on (a) their local edge information, (b) the outputs from previous rounds, and (c) the public information, so basically the whole transcript so far. The vertices then output a message which can be seen by all other vertices and the curator. The released information of each vertex is ε-dp since it computed based on the output of its local randomizer. In each round the curator can choose whom to query, what local randomizer they use, and what privacy parameters the randomizer uses. Since the curator's choice only depends on the transcripts from the previous rounds, this is exactly the definition of a protocol in the above definition.

Algorithm 1 Multidimensional AboveThreshold (MAT).

Input: Private graph G, adaptive queries $\{\vec{f}_1, \ldots, \vec{f}_n\}$, threshold vector $\vec{T}$, privacy ε, sensitivity D.
Output: A sequence of responses $\{\vec{a}_1, \ldots, \vec{a}_n\}$ where $a_{i,j}$ indicates if $f_{i,j}(G) \geq \vec{T}_j$

for $j = 1, \ldots, d$ **do**
  $\hat{T}_j \leftarrow \vec{T}_j + \mathrm{Lap}(2D/\varepsilon)$
end for

for each query $\vec{f}_i \in \{\vec{f}_1, \ldots, \vec{f}_n\}$ **do**
  for $j = 1, \ldots, d$ **do**
    Let $\nu_{i,j} \leftarrow \mathrm{Lap}(4D/\varepsilon)$
    if $f_{i,j}(G) + \nu_{i,j} \geq \hat{T}_j$ **then**
      Output $a_{i,j} = \top$
      Stop answering queries for coordinate j
    else
      Output $a_{i,j} = \bot$
    end if
  end for
end for

We end the discussion with a short remark on stateless/stateful vertices. As defined above, a local randomizer is only allowed to look at the past (public) transcript and its (private) neighborhood list to compute its output for the current round. This formal definition disallows the use of private states that persist across rounds in the local memory of a vertex. Private stateful behaviour is nonetheless crucial for implementing MAT locally, since we require each vertex to store its noisy threshold privately. This is not a problem, since one can implement a protocol as in [5, Lemma 8] to simulate private states using local randomizers.

4 Generalized Sparse Vector Technique

We introduce a simple, novel multidimensional generalization of the AboveThreshold mechanism, which is the basis of the algorithms in this paper. In the standard AboveThreshold mechanism from Section 3.6 in [25], we are given as input a threshold T and an adaptive stream of sensitivity 1 queries $f_1, f_2, \ldots$, and the goal is to output the first query which exceeds the threshold. In the multidimensional version, we have a d-dimensional vector of thresholds $\vec{T} = (T_1, \ldots, T_d)$ and an adaptive stream of ℓ_1-sensitivity D vector-valued queries $\vec{f}_1, \vec{f}_2, \ldots$. The goal is to output, for each coordinate $j \in [d]$, the first query i_j for which the j^{th} coordinate exceeds the threshold T_j. The pseudo-code is given in Algorithm 1.

Here D is defined as the sensitivity for *all* dimensions, i.e., for edge-neighboring $G \sim G'$,

$$D := \max_{G \sim G'} \left(\sum_{j \in [d]} \left(\max_{i \in [I_j]} \left(|f_{i,j}(G) - f_{i,j}(G')| \right) \right) \right), \tag{1}$$

where I_j is the number of queries in dimension j.

▶ **Theorem 18.** *Algorithm 1 is ε-differentially private.*

Proof. Fix $G \sim G'$ to be arbitrary edge-neighboring graphs. Let $\mathcal{A}(G)$ and $\mathcal{A}(G')$ denote the random variable representing the output of Algorithm 1 when run on G and G', respectively. For each coordinate $j \in [d]$, there is some $r(j) \in \{1, \ldots, n+1\}$ such that the output satisfies $a_{\cdot j} \in \{\top, \bot\}^{r(j)}$ and has the form that for all $i < r(j)$, $a_{i,j} = \bot$. We can assume without loss of generality that $a_{r(j),j} = \top$; if $r(j) < n+1$, this must be the case and if $r(j) = n+1$, we can simply ask an additional query which evaluates to ∞, independent of the dataset. Our goal is to show that for any output a, we have

$$\Pr[\mathcal{A}(G) = a] \leq \exp(\varepsilon) \cdot \Pr[\mathcal{A}(G') = a].$$

Observe that there are two sources of randomness in the algorithm: the noisy thresholds $\hat{T}_j$ for each $j \in [d]$ and the perturbations of the queries $\nu_{i,j}$ for each $j \in [d]$, $i \in [r(j)]$. We fix the random variables $\nu_{i,j}$ for each $j \in [d]$ and $i < r(j)$ by simply conditioning on their randomness; for neighboring graphs, we have a coupling between corresponding variables. For simplicity, we omit this from notation. The remaining random variables are $\hat{T}_j$ and $\nu_{r(j),j}$ for each $j \in [d]$; we will be taking probabilities over their randomness. The main observation needed is that the output of $\mathcal{A}$ is uniquely determined by the values of $r(j)$ for each $j \in [d]$. Thus, we can define the (deterministic, due to conditioning on the probabilities) quantity

$$g_j(G) \coloneqq \max\{f_{i,j}(G) + \nu_{i,j} : i < r(j)\}$$

for each $j \in [d]$ so that the event that $\mathcal{A}(G) = a$ is equivalent to the event where

$$\hat{T}_j > g_j(G) \text{ and } f_{r(j),j}(G) + \nu_{r(j),j} \geq \hat{T}_j \text{ for each } j \in [d]. \tag{2}$$

Let $p(\cdot)$ and $q(\cdot)$ denote the density functions of the random vectors consisting of $\hat{T}(j)$ and $\nu_{r(j),j}$ for each $j \in [d]$. Thus,

$$\Pr[\mathcal{A}(G) = a] = \Pr[\hat{T}_j \in (f_{r(j),j}(G) + \nu_{r(j),j}, g_j(G)) \ \forall j \in [d]]$$

Using v'_j to denote the value of $\nu_{r(j),j}$ and t'_j to denote the value of $\hat{T}_j$,

$$\Pr[\mathcal{A}(G) = a] = \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} p(\vec{t'}) q(\vec{v'}) \cdot \mathbb{1}\{t'_j \in (f_{r(j),j}(G) + v'_j, g_j(G)) \ \forall j \in [d]\} \, d\vec{v'} \, d\vec{t'} \tag{3}$$

Now, we make a change of variables from $\vec{v'}$ to $\vec{v}$ and $\vec{t'}$ to $\vec{t}$, with the change given by

$$v'_j = v_j + g_j(G) - g_j(G') + f_{r(j),j}(G') - f_{r(j),j}(G)$$
$$t'_j = t_j + g_j(G) - g_j(G')$$

for each $j \in [d]$. We have that $d\vec{t'}_j = d\vec{t}_j$, and $d\vec{v'}_j = d\vec{v}_j$ since the new variables differ from the old ones by a constant that is independent of $\vec{v}$ and $\vec{t}$. The term inside the indicator function in the integral now becomes

$$t_j + g_j(G) - g_j(G') \in (f_{r(j),j}(G) + v_j + g_j(G) - g_j(G') + f_{r(j),j}(G') - f_{r(j),j}(G), g_j(G)) \forall j \in [d]$$

which on subtracting $g_j(G) - g_j(G')$ gives

$$t_j \in (f_{r(j),j}(G') + v_j, g_j(G')) \forall j \in [d]$$

Thus, we can apply the change of variables to rewrite (3) as

$$\int_{\mathbb{R}^d} \int_{\mathbb{R}^d} p(\vec{t'}) q(\vec{v'}) \mathbb{1}\{t_j \in (f_{r(j),j}(G') + v_j, g_j(G')) \ \forall j \in [d]\} \, d\vec{v} \, d\vec{t}.$$

Since we have conditioned on the randomness of $\nu_{i,j}$ for each $j \in [d]$ and $i < r(j)$, we have $\|\vec{v'} - \vec{v}\|_1 \leq 2D$ and $\|\vec{t'} - \vec{t}\|_1 \leq D$ since the vectors $f_{i\cdot}(G)$ have sensitivity D, implying that the vectors $g_{\cdot}(G)$ also have sensitivity D. This implies that $p(\vec{t'}) \leq \exp(\varepsilon/2) \cdot p(\vec{t})$ and $q(\vec{v'}) \leq \exp(\varepsilon/2) \cdot q(\vec{v})$ by the properties of the Laplace distribution. We can thus upper bound the above integral by

$$\exp(\varepsilon) \cdot \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} p(\vec{t}) q(\vec{v}) \mathbb{1}\{t_j \in (f_{r(j),j}(G') + v_j, g_j(G'))\ \forall j \in [d]\}\, d\vec{v}\, d\vec{t}$$

Rewriting the integral as the probability of an event, we get

$$\exp(\varepsilon) \cdot \Pr[\hat{T}_j \in (f_{r(j),j}(G') + \nu_{r(j),j}, g_j(G'))\ \forall j \in [d]].$$

Finally, by our observation in (2), this is equal to

$$\exp(\varepsilon) \cdot \Pr[\mathcal{A}(G') = a].$$

Putting the inequalities together, we get

$$\Pr[\mathcal{A}(G) = a] \leq \exp(\varepsilon) \cdot \Pr[\mathcal{A}(G') = a].$$

Since G, G' were arbitrary edge-neighboring graphs, this completes the proof. ◀

4.1 LEDP Multidimensional AboveThreshold (MAT) Mechanism

In our algorithms, we apply the multidimensional AboveThreshold mechanism in the special case where the queries $\vec{f_i}$ have $d = n$ coordinates, and each coordinate of the query corresponds to a node u in the graph G and only depends on the edges $e = (u, v)$ for $v \in V - \{u\}$. In other words, we have an instance of the AboveThreshold on each node $u \in V$, where the data used for the AboveThreshold instance is only the local (edge) data of u. We will show that in this setting, the multidimensional AboveThreshold mechanism can be implemented locally to satisfy local edge-differential privacy.

For clarity of notation, we index the coordinates of the queries and threshold vector by nodes $v \in V$ instead of indices $j \in [n]$. That is, each query consists of $\vec{f_{i,v}}$ for $v \in V$ and the threshold vector consists of $\vec{T_v}$ for $v \in V$. We will now present the changes needed for the local implementation. In Lines 1–3, let each node $u \in V$ compute and store its noisy threshold $\hat{T}_u$ using the public threshold $\vec{T}_u$. Then for each query $\vec{f_i}$ in lines 5–15, each node $u \in V$ can sample its own noise $\nu_{i,u}$ and check the condition $f_{i,u}(G) + \nu_{i,u} \geq \hat{T}_u$. The pseudocode is given in Algorithm 2, where actions performed by node v in the algorithm are performed their corresponding local randomizers, and we now show that this is locally edge-differentially private.

▶ **Theorem 19.** *Algorithm 2 is ε-locally edge differentially private.*

Proof. First, we need to show that the local randomizers are in fact edge-differentially private. This can be seen because each randomizers' output is a subset of the output of Algorithm 1 and it is only computed based on the incident edges of each vertex. In other words, the output is a post-processing of Algorithm 1, so the privacy guarantees follow from Theorem 18 and Lemma 15. Next, we argue that the transcript is also ε-edge differentially private. Recall that the transcript consists of the set of vertices chosen, the set of local randomizers assigned, the set of privacy parameters assigned, and the set of outputs. In the algorithm, the set of vertices chosen, the set of local randomizers assigned, and the set of privacy parameters assigned are all functions of the outputs from the previous rounds and the public information. Thus, it suffices to prove that the outputs are differentially private by post-processing (Lemma 15). But this was proven in Theorem 18, so we are done. ◀

Algorithm 2 Local Multidimensional AboveThreshold.

Input: Private graph G, adaptive queries $\{\vec{f}_i\}$, threshold vector $\vec{T}$, privacy ε, sensitivity D.
Output: A sequence of responses $\{\vec{a}_i\}$ where $a_{i,j}$ indicates if $f_{i,j}(G) \geq \vec{T}_j$

for $v \in V$ **do**
  Node v computes $\hat{T}_v \leftarrow \vec{T}_v + \text{Lap}(2D/\varepsilon)$ and stores it
end for

for each query $\vec{f}_i$ **do**
  for $v \in V$ **do**
    Node v samples $\nu_{i,v} \leftarrow \text{Lap}(4D/\varepsilon)$
    if $f_{i,v}(G) + \nu_{i,v} \geq \hat{T}_v$ **then**
      Node v outputs $a_{i,j} = \top$
      Node v outputs STOP, and stops answering queries
    else
      Node v outputs $a_{i,v} = \perp$
    end if
  end for
end for

5 LEDP k-Core Decomposition

In this section, we present one application of MAT by giving an improved algorithm for differentially private k-core decomposition. We first present a variant of the classical (non-private) algorithm for k-core decomposition in Section 5.1, and then show how to make it private in Section 5.2. We defer our other proofs to the full version.

5.1 A Variation of the Classical Algorithm

The classical peeling algorithm begins with the full vertex set V, which is the 0-core of the graph. Given the $(k-1)$-core, the algorithm computes the k-core via an iterative peeling process: the algorithm repeatedly removes all nodes v for which the induced degree is less than k, and labels the nodes which remain as being part of the k-core. Running this for k from 1 up to n gives the full algorithm, the pseudocode of which is given in Algorithm 3.

▶ **Theorem 20.** *For all $v \in V$, the output $\hat{k}(v)$ given by Algorithm 3 is the core number of v.*

Proof. We will inductively show that the algorithm recovers the k-core of the graph. The base case of $k = 0$ is easy. Now, assume that the algorithm finds the true $(k-1)$-core. Let $V(k)$ denote the subset of nodes which aren't removed in the iterative process for $k-1$ in Line 2. We have that each node $v \in V(k)$ has induced degree at least k in $V(k)$, or else it would have been removed. Thus, we know that the core numbers $k(v)$ is at least k for each $v \in V(k)$, so we have that $V(k)$ is a subset of the true k-core. Now, let K denote the true k-core. Since the k-core is always a subset of the $(k-1)$-core by definition, each node $v \in K$ is in V_t at the beginning of the iterations. Furthermore, we know that $v \in K$ is never removed from V_t since the induced degree is always at least k since $K \subseteq V_t$. Thus, we have that $V(k)$ is a superset of the true k-core as well. Thus, $V(k)$ is the true k-core for each k, so all nodes are labelled correctly. ◀

Algorithm 3 Threshold-Based k-core Decomposition Algorithm.

1 **Input:** Graph $G = (V, E)$.
2 **Output:** k-core value of each node $v \in V$

1: Initialize $V_0 \leftarrow V$, $t \leftarrow 0$, $\hat{k}(v) \leftarrow 0$ for all $v \in V$
2: **for** $k = 1, \ldots, n$ **do**
3: **repeat**
4: $t \leftarrow t + 1$, $V_t \leftarrow V_{t-1}$
5: **for** $v \in V_{t-1}$ **do**
6: **if** $d_{V_{t-1}}(v) < k$ **then**
7: $V_t \leftarrow V_t - \{v\}$
8: **end if**
9: **end for**
10: **until** $V_{t-1} - V_t = \emptyset$
11:
12: Update the core numbers $\hat{k}(v) \leftarrow k$ for all nodes $v \in V_t$
13: **end for**

5.2 Private Implementation of the Algorithm

It is difficult to turn the classical algorithm into a differentially private one because it has $\Omega(n)$ iterations, which causes us to incur $\tilde{\Omega}(n)$ additive error when using basic composition (Lemma 14). In fact, this was cited in [20] as the motivation for basing their algorithms on parallel/distributed algorithms for k-core decomposition, since those algorithms often have $\operatorname{poly}\log(n)$ round-complexity. Our main observation is that the private version of Algorithm 3 can be analyzed as a special case of the Multidimensional AboveThreshold mechanism, so it doesn't need to incur the $\Omega(n)$ additive error due to composition.

▶ **Theorem 21.** *Algorithm 4 is ε-edge differentially private.*

Proof. We will show that Algorithm 4 is an instance of the multidimensional AboveThreshold mechanism, implying that it is ε-edge differentially private. Specifically, we will show that its output can be obtained by post-processing the output of an instance of Algorithm 1. Indeed, consider the instance of the multidimensional AboveThreshold mechanism where the input graph is G, the privacy parameter is ε, and the threshold vector $\vec{T} = \vec{0}$ is the zero vector. We will now inductively (and adaptively) define the queries.

For each iteration t, the t^{th} query consists of the vector of $k - d_{V_{t-1}}(v)$ for each $v \in V$, where $V_0 = V$ as in the algorithm; note that this matches with the queries on Line 2 of the algorithm. First, observe that the sensitivity of the vector queries of $k - d_{V_{t-1}}(v)$ for each $v \in V$ is $D = 2$ since one edge change will only affect two coordinates of the query (each by 1), as needed in the privacy analysis. Next, observe that we can construct V_t from V_{t-1} using only the outputs $a_t(v)$ of the queries at the current iteration: given $v \in V_{t-1}$, we include v in V_t as well if and only if $a_t(v) = \perp$, which is equivalent to the condition in Line 2 of $d_{V_{t-1}}(v) + \nu_{t,v} \leq k + \tilde{\ell}(v)$[3]. Thus, this is a feasible sequence of adaptive queries for the AboveThreshold mechanism.

[3] Here, we implicitly assume that the randomness used by the algorithm and the AboveThreshold mechanism are the same. This is justified by a coupling of the random variables in the two algorithms. Specifically, we couple the noise added in Line 4 of the algorithm with the noise added in Line 2 of the AboveThreshold mechanism and couple the noise $\nu_{t,v}$ with $\nu_{i,j}$ in the AboveThreshold mechanism. It

Algorithm 4 ε-Differentially Private k-Core Decomposition.

Input: Graph $G = (V, E)$, privacy parameter $\varepsilon > 0$.
Output: An $(1, 120\log(n)/\varepsilon)$-approximate k-core value of each node $v \in V$

$V_0 \leftarrow V$, $t \leftarrow 0$, $k = 60\log n/\varepsilon$.
Initialize $\hat{k}(v) \leftarrow 0$ for all $v \in V$
for $v \in V$ **do**
  $\tilde{\ell}(v) \leftarrow \text{Lap}(4/\varepsilon)$
end for

while $k \leq n$ **do**
  repeat
    $t \leftarrow t + 1$, $V_t \leftarrow V_{t-1}$
    for $v \in V_{t-1}$ **do**
      $\nu_{t,v} \leftarrow \text{Lap}(8/\varepsilon)$
      if $d_{V_{t-1}}(v) + \nu_{t,v} \leq k + \tilde{\ell}(v)$ **then**
        $V_t \leftarrow V_t - \{v\}$
      end if
    end for
  until $V_{t-1} - V_t = \emptyset$

  Update the core numbers $\hat{k}(v) \leftarrow k$ for all nodes $v \in V_t$
  $k \leftarrow k + 60\log n/\varepsilon$
end while

Since the sequence of subsets $\{V_t\}$ are obtained by post-processing the outputs $a_t(v)$ of the AboveThreshold mechanism, they are ε-differentially private by Theorem 18 and Lemma 15. By applying post-processing again, the sequence of pairs (V_t, k_t) for each iteration t is also ε-differentially private since the sequence k_t is public. Given the pairs (V_t, k_t) for each iteration t, we can now recover the approximate core numbers which the algorithm outputs by setting the core numbers as $\hat{k}(v) = k_t$ for each node in $V_t - V_{t-1}$. It is easily verified that this gives the exact same output as Algorithm 4, so we have ε-differential privacy for the algorithm by applying post-processing (Lemma 15) once again. ◀

▶ **Theorem 22.** *Algorithm 4 outputs* $(1, \frac{120\log n}{\varepsilon})$*-approximate core numbers* $\hat{k}(v)$ *with probability* $1 - O(\frac{1}{n^2})$.

Proof. By the density function of the Laplace distribution, we know that we have $|\nu_{t,u}| \leq \frac{40\log n}{\varepsilon}$ and $|\tilde{\ell}(u)| \leq \frac{20\log n}{\varepsilon}$ each with probability $\geq 1 - \frac{1}{n^5}$. Since there are at most $O(n^3)$ such random variables, taking a union bound over all nodes $u \in V$ and all iterations of the loops gives the above guarantee with probability $\geq 1 - O\left(\frac{1}{n^2}\right)$. We condition on the event that the above inequalities hold true for all $t \in [T]$ and $u \in V$ for the remainder of the proof.

Fix an arbitrary iteration k of the while loop starting on Line 7. Let H be the set of nodes remaining in V_t at the end of the while loop in Lines 8–16. We claim that (i) all nodes $u \in H$ have core number at least $k - \frac{60\log n}{\varepsilon}$ and (ii) all nodes $u \notin H$ have core number at most $k + \frac{60\log n}{\varepsilon}$. To see (i), consider the subgraph H; we claim that each node $u \in H$ has

is easy to see that for $\Delta = 2$, the random variables are exactly the same so the privacy guarantees of multidimensional AboveThreshold translates to privacy guarantees for our algorithm.

induced degree at least $k - \frac{60 \log n}{\varepsilon}$ in H. Indeed, since each node in H was not removed in the final iteration of the while loop in Lines 8–16, we have that $d_{H_t}(u) + \nu_{t,u} \geq k + \tilde{\ell}(u)$ for each $u \in H$. But since we have assumed $|\nu_{t,u}| \leq \frac{40 \log n}{\varepsilon}$ and $|\tilde{\ell}(u)| \leq \frac{20 \log n}{\varepsilon}$, our desired bounds follow directly by the triangle inequality. To see (ii), let's suppose for contradiction that the k-core value of u is $k(u) > \ell + \frac{60 \log n}{\varepsilon'}$. Then there exists a subgraph $u \in K \subseteq V$ where the induced degree of each node $v \in K$ is $d_K(v) \geq k(u)$. But for such a subgraph K, the condition in Line 12 will always be true (again, because $|\nu_{t,u}| \leq \frac{40 \log n}{\varepsilon}$ and $|\tilde{\ell}(u)| \leq \frac{20 \log n}{\varepsilon}$) so $K \subseteq H$. But since $u \in K$, this contradicts the fact that $u \notin H$.

Using the above bounds, we can now prove our desired results. First, consider an arbitrary node $u \in V$ labeled $\hat{k}(u)$ within the while loop in Lines 7–20 and not relabeled later. Since u was labeled $\hat{k}(u)$ in the current iteration, we have that $k(u) \geq \hat{k}(u) - \frac{60 \log n}{\varepsilon'}$. Since u was not relabelled in the next iteration where the threshold was $k = \hat{k}(u) + \frac{60 \log n}{\varepsilon}$, the node u was removed from V_t at that iteration. Consequently, we have $k(u) \leq k + \frac{60 \log n}{\varepsilon} = \hat{k}(u) + \frac{120 \log n}{\varepsilon}$ by what we just proved above. Since the output of our algorithm is $\hat{k}(u)$, the desired bounds follow directly. Now, let's consider an arbitrary node $u \in V$ labeled 0 at Line 2 at the beginning of the algorithm and not relabeled later. Since the node u was not relabelled at the first iteration where $k = \frac{60 \log n}{\varepsilon}$, it was removed from V_t at that iteration, implying that $k(u) \leq \frac{120 \log n}{\varepsilon}$. Hence, we have the desired approximation guarantees for all nodes. ◀

References

1 J. Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, 2005.

2 Sepehr Assadi, Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Coloring in graph streams via deterministic and adversarially robust algorithms. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '23, pages 141–153, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3584372.3588681.

3 Bahman Bahmani, Ashish Goel, and Kamesh Munagala. Efficient primal-dual graph algorithms for MapReduce. In *International Workshop on Algorithms and Models for the Web Graph (WAW)*, volume 8882, pages 59–78, 2014. doi:10.1007/978-3-319-13123-8_6.

4 Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proc. VLDB Endow.*, 5(5):454–465, 2012. doi:10.14778/2140436.2140442.

5 Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In *Annual International Cryptology Conference*, pages 451–468, 2008. doi:10.1007/978-3-540-85174-5_25.

6 Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 11:1–11:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.11.

7 Suman K. Bera, Lior Gishboliner, Yevgeny Levanzov, C. Seshadhri, and Asaf Shapira. Counting subgraphs in degenerate graphs. *J. ACM*, 69(3), June 2022. doi:10.1145/3520240.

8 Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. In *Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151, pages 38:1–38:20, 2020. doi:10.4230/LIPICS.ITCS.2020.38.

9 Edvin Berglin and Gerth Stølting Brodal. A simple greedy algorithm for dynamic graph orientation. *Algorithmica*, 82(2):245–259, 2020. doi:10.1007/S00453-018-0528-0.

10 Anup Bhattacharya, Arijit Bishnu, Gopinath Mishra, and Anannya Upasana. Even the easiest(?) graph coloring problem is not easy in streaming! In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 15:1–15:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ITCS.2021.15.

11 Sayan Bhattacharya, Fabrizio Grandoni, Janardhan Kulkarni, Quanquan C. Liu, and Shay Solomon. Fully dynamic $(\Delta+1)$-coloring in $O(1)$ update time. *ACM Trans. Algorithms*, 18(2):10:1–10:25, 2022. doi:10.1145/3494539.

12 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *ACM Symposium on Theory of Computing (STOC)*, pages 173–182, 2015. doi:10.1145/2746539.2746592.

13 Chandra Chekuri, Aleksander Bjørn Christiansen, Jacob Holm, Ivor van der Hoog, Kent Quanrud, Eva Rotenberg, and Chris Schwiegelshohn. Adaptive out-orientations with applications. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3062–3088. SIAM, 2024.

14 Chandra Chekuri, Kent Quanrud, and Manuel R. Torres. Densest subgraph: Supermodularity, iterative peeling, and flow. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1531–1555, 2022. doi:10.1137/1.9781611977073.64.

15 Aleksander BG Christiansen, Eva Rotenberg, Teresa Anna Steiner, and Juliette Vlieghe. Private graph colouring with limited defectiveness. *arXiv preprint arXiv:2404.18692*, 2024.

16 Luciano da Fontoura Costa, Osvaldo N Oliveira Jr, Gonzalo Travieso, Francisco Aparecido Rodrigues, Paulino Ribeiro Villas Boas, Lucas Antiqueira, Matheus Palhares Viana, and Luis Enrique Correa Rocha. Analyzing and modeling real-world phenomena with complex networks: a survey of applications. *Advances in Physics*, 60(3):329–412, 2011.

17 Anindya De. Lower bounds in differential privacy. In *Theory of Cryptography Conference, TCC*, pages 321–338. Springer, 2012. doi:10.1007/978-3-642-28914-9_18.

18 Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. Julienne: A framework for parallel graph algorithms using work-efficient bucketing. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 293–304, 2017. doi:10.1145/3087556.3087580.

19 Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2018.

20 Laxman Dhulipala, Quanquan C. Liu, Sofya Raskhodnikova, Jessica Shi, Julian Shun, and Shangdi Yu. Differential privacy from locally adjustable graph algorithms: k-core decomposition, low out-degree ordering, and densest subgraphs. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 754–765. IEEE, 2022. doi:10.1109/FOCS54457.2022.00077.

21 Michael Dinitz, Satyen Kale, Silvio Lattanzi, and Sergei Vassilvitskii. Almost tight bounds for differentially private densest subgraph. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2908–2950. SIAM, 2025. doi:10.1137/1.9781611978322.94.

22 Michael Dinitz, George Z. Li, Quanquan C. Liu, and Felix Zhou. Differentially private matchings, 2025. doi:10.48550/arXiv.2501.00926.

23 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, pages 265–284, 2006. doi:10.1007/11681878_14.

24 Cynthia Dwork, Moni Naor, Omer Reingold, Guy N Rothblum, and Salil Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *ACM Symposium on Theory of Computing (STOC)*, pages 381–390, 2009. doi:10.1145/1536414.1536467.

25 Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014. doi:10.1561/0400000042.

26 Talya Eden, Quanquan C. Liu, Sofya Raskhodnikova, and Adam D. Smith. Triangle counting with local edge differential privacy. In *International Colloquium on Automata, Languages, and Programming, ICALP*, pages 52:1–52:21, 2023. doi:10.4230/LIPICS.ICALP.2023.52.

27 Hossein Esfandiari, MohammadTaghi Hajiaghayi, and David P Woodruff. Brief announcement: Applications of uniform sampling: Densest subgraph and beyond. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 397–399, 2016. doi:10.1145/2935764.2935813.

28 Hossein Esfandiari, Silvio Lattanzi, and Vahab Mirrokni. Parallel and streaming algorithms for k-core decomposition. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1397–1406, 2018.

29 Alireza Farhadi, Mohammad Taghi Hajiaghai, and Elaine Shi. Differentially private densest subgraph. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 151, pages 11581–11597, 2022. URL: https://proceedings.mlr.press/v151/farhadi22a.html.

30 Mohsen Ghaffari and Fabian Kuhn. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1009–1020. IEEE, 2022.

31 Mohsen Ghaffari, Silvio Lattanzi, and Slobodan Mitrović. Improved parallel algorithms for density-based network clustering. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2201–2210, 2019. URL: http://proceedings.mlr.press/v97/ghaffari19a.html.

32 Mohsen Ghaffari and Christiana Lymouri. Simple and near-optimal distributed coloring for sparse graphs. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.DISC.2017.20.

33 Meng He, Ganggui Tang, and Norbert Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *International Symposium on Algorithms and Computation*, pages 128–140. Springer, 2014. doi:10.1007/978-3-319-13075-0_11.

34 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity. *CoRR*, abs/2002.10142, 2020. arXiv:2002.10142.

35 Monika Henzinger and Pan Peng. Constant-time dynamic $(\Delta+1)$-coloring. *ACM Trans. Algorithms*, 18(2):16:1–16:21, 2022. doi:10.1145/3501403.

36 Justin Hsu, Zhiyi Huang, Aaron Roth, Tim Roughgarden, and Zhiwei Steven Wu. Private matchings and allocations. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 21–30, 2014. doi:10.1145/2591796.2591826.

37 H. Kabir and K. Madduri. Parallel k-core decomposition on multicore platforms. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1482–1491, 2017.

38 Łukasz Kowalik. Approximation scheme for lowest outdegree orientation and graph density measures. In *International Symposium on Algorithms and Computation*, pages 557–566. Springer, 2006.

39 Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzone, and Francesco Bonchi. A survey on the densest subgraph problem and its variants. *arXiv preprint arXiv:2303.14467*, 2023. doi:10.48550/arXiv.2303.14467.

40 Yang Li, Michael Purcell, Thierry Rakotoarivelo, David Smith, Thilina Ranbaduge, and Kee Siong Ng. Private graph data release: A survey. *ACM Computing Surveys*, 55(11):1–39, 2023. doi:10.1145/3569085.

41 Jingcheng Liu and Kunal Talwar. Private selection from private candidates. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 298–309, 2019. doi:10.1145/3313276.3316377.

42 Quanquan C. Liu, Jessica Shi, Shangdi Yu, Laxman Dhulipala, and Julian Shun. Parallel batch-dynamic algorithms for k-core decomposition and related graph problems. In *34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 191–204, 2022. doi:10.1145/3490148.3538569.

43 Fragkiskos D Malliaros, Christos Giatsidis, Apostolos N Papadopoulos, and Michalis Vazirgiannis. The core decomposition of networks: Theory, algorithms and applications. *The VLDB Journal*, 29:61–92, 2020. doi:10.1007/S00778-019-00587-4.

44 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, July 1983. doi:10.1145/2402.322385.

45 Yannic Maus. Distributed graph coloring made easy. *ACM Trans. Parallel Comput.*, 10(4), December 2023. doi:10.1145/3605896.

46 Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T Vu. Densest subgraph in dynamic graph streams. In *International Symposium on Mathematical Foundations of Computer Science*, pages 472–482. Springer, 2015. doi:10.1007/978-3-662-48054-0_39.

47 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms (TALG)*, 12(1):1–15, 2015. doi:10.1145/2700206.

48 Dung Nguyen and Anil Vullikanti. Differentially private densest subgraph detection. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8140–8151, 2021. URL: http://proceedings.mlr.press/v139/nguyen21i.html.

49 Ahmet Erdem Sariyüce and Ali Pinar. Fast hierarchy construction for dense subgraphs. *Proceedings of the VLDB Endowment*, 10(3):97–108, 2016. doi:10.14778/3021924.3021927.

50 Saurabh Sawlani and Junxing Wang. Near-optimal fully dynamic densest subgraph. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 181–193, 2020. doi:10.1145/3357713.3384327.

51 Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Corescope: Graph mining using k-core analysis—patterns, anomalies and algorithms. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 469–478. IEEE, 2016. doi:10.1109/ICDM.2016.0058.

52 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. *ACM Trans. Algorithms*, 16(3), June 2020. doi:10.1145/3392724.

53 Hsin-Hao Su and Hoa T. Vu. Distributed Dense Subgraph Detection and Low Outdegree Orientation. In *34th International Symposium on Distributed Computing*, pages 15:1–15:18, 2020. doi:10.4230/LIPICS.DISC.2020.15.

54 Salil P. Vadhan and Tianhao Wang. Concurrent composition of differential privacy. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 582–604. Springer, 2021. doi:10.1007/978-3-030-90453-1_20.

Incremental Maximization for a Broad Class of Objectives

Yann Disser
TU Darmstadt, Germany

David Weckbecker
TU Darmstadt, Germany

Abstract

We consider incremental maximization problems, where the solution has to be built up gradually by adding elements one after the other. In every step, the incremental solution must be competitive, compared against the optimum solution of the current cardinality. We prove that a competitive solution always exists when the objective function is monotone and β-accountable, by providing a scaling algorithm that guarantees a constant competitive ratio. This generalizes known results and, importantly, yields the first competitive algorithm for the natural class of monotone and subadditive objective functions.

2012 ACM Subject Classification Theory of computation → Online algorithms; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Combinatorial optimization

Keywords and phrases incremental maximization, competitive analysis, subadditive functions

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.92

Funding Supported by DFG grant DI 2041/2.

1 Introduction

Large infrastructure projects are typically implemented sequentially over longer periods of time. In such settings, it makes sense to prioritize completion of parts of the project that already yield some benefit on their own. This raises the general question of how to compute incremental solutions with a good partial benefit throughout the course of the project. We model such settings mathematically by using the abstract framework of the *incremental maximization* (IncMax) problem.

In the IncMax problem, we are given a countable set of elements U, along with an objective function $f\colon 2^U \to \mathbb{R}_{\geq 0}$. We generally require f to be monotone, i.e., that $f(A) \leq f(B)$ for all $A \subseteq B \subseteq U$. An *incremental solution* π is a permutation $(e_1, e_2, \dots)$ of the elements in U that specifies the order in which the elements are to be included in the solution. The solution after step $C \in \mathbb{N}$ is then given by $\pi(C) := \{e_1, \dots, e_C\}$, and we denote the optimum solution value of cardinality C by

$$\textsc{Opt}(C) := \sup\{f(A) \mid A \subseteq U, |A| = C\}.$$

We say that the incremental solution π is *ρ-competitive* if $\textsc{Opt}(C) \leq \rho \cdot \pi(C)$ for all $C \in \mathbb{N}$. The *competitive ratio* of π is $\rho(\pi) := \inf\{\rho \geq 1 \mid \pi \text{ is } \rho\text{-competitive}\}$, the competitive ratio of a problem instance given by U and f is $\rho(U, f) := \inf_\pi \rho(\pi)$, and the competitive ratio of a set of instances $\mathcal{C}$ is $\rho(\mathcal{C}) := \sup\{\rho(U, f) \mid (U, f) \in \mathcal{C}\}$.

Clearly, we cannot hope for good incremental solutions in full generality: For example, if $f(E)$ is given by the value of a maximum flow using an edge set E in some underlying graph that consists of a short path of low capacity and a long path of high capacity, then we have to decide whether to invest in the short path to obtain some value early, or in the long path to achieve a high value as quickly as possible, but cannot ensure both simultaneously. More


© Yann Disser and David Weckbecker;
licensed under Creative Commons License CC-BY 4.0

generally, we must demand that the objective function has some form of a diminishing-returns property that requires the value of solutions to be explainable by the value of their parts, i.e., value cannot suddenly emerge upon completion of certain structures (such as paths in the flow example).

Diminishing returns are typically captured by submodular objective functions, i.e., functions $f\colon 2^U \to \mathbb{R}_{\geq 0}$ with $f(A)+f(B) \geq f(A\cup B)+f(A\cap B)$ for all $A, B \subseteq U$. In a classical result, Nemhauser et al. [25] showed that the obvious greedy algorithm, that adds to the current solution S the element e of largest marginal value $f(S\cup\{e\})-f(S)$ in each step, produces a $\frac{\mathrm{e}}{\mathrm{e}-1}$-competitive solution for submodular objectives. The most natural relaxation of submodularity that still intuitively captures diminishing returns is subadditivity, i.e., $f(A)+f(B)\geq f(A\cup B)$ for all $A, B \subseteq U$. While subadditivity is a classical and natural property that intuitively seems promising for incremental maximization, no bounds on the compatitive ratio were known to date.

Unable to bound the competitive ratio for subadditive objectives, Bernstein et al. [1] considered *accountable* functions f instead (see Figure 1 along with the following). A function $f\colon 2^U \to \mathbb{R}_{\geq 0}$ is called *accountable* if, for every finite and non-empty set $S \subseteq U$, there exists $e \in S$ with $f(S\setminus\{e\}) \geq \big(1-\frac{1}{|S|}\big)f(S)$. Intuitively, this property directly requires that the value of a set cannot suddenly emerge when adding its last element. Bernstein et al. [1] gave a $(\varphi+1)$-competitive scaling algorithm for accountable objectives, where $\varphi := (\sqrt{5}+1)/2$ denotes the golden ratio.

Note that submodular functions are accountable, and many combinatorial maximization problems, such as weighted (multidimensional) matching, set packing, coverage, knapsack, and many more, give rise to accountable objective functions (see [1]). On the other hand, not all subadditive functions are accountable. For a simple example, assume that the elements of U represent identical unit squares, and $f(S)$ is the diameter of the largest square that can be tiled with the elements in $S \subseteq U$. Then, $f(S) = \lfloor\sqrt{|S|}\rfloor$ grows strictly sublinearly in $|S|$ and is thus subadditive, but it is not accountable, as can be seen by considering $|S| = 4$, which yields $f(S\setminus\{e\}) = 1 < \frac{3}{2} = \big(1-\frac{1}{4}\big)f(S)$ for all $e \in S$. More generally, functions of the form $f(A) = \lfloor g(|A|)\rfloor$ with $g \in o(|A|)$ are subadditive, but not accountable.

Disser and Weckbecker [11] considered γ-α-augmentability as an alternative relaxation of submodularity that additionally maintains competitiveness of the greedy algorithm. A function is called γ-α-augmentable if, for all finite $A, B \subseteq U$, there exists $b \in B$ with $f(A\cup\{b\})-f(A) \geq \frac{\gamma f(A\cup B)-\alpha f(A)}{|B|}$. In this setting, the greedy algorithm has competitive ratio exactly $\frac{\alpha}{\gamma}\cdot\frac{\mathrm{e}^\alpha}{\mathrm{e}^\alpha-1}$.

In this paper, we provide a scaling algorithm that is competitive for an abstract class of functions that relaxes all above classes and includes, for the first time, all subadditive functions.

Our results

We introduce a class of functions based on an equivalent characterization of accountability given by Disser et al. [10]. They showed that a function $f\colon 2^U \to \mathbb{R}_{\geq 0}$ is accountable if and only if, for every finite $S \subseteq U$, there exists an ordering $(e_1,\dots,e_{|S|})$ of S with $f(\{e_1,\dots,e_i\}) \geq \frac{i}{|S|}f(S)$ for all $i \in \{1,\dots,|S|\}$. We relax this characterization as follows.

▶ **Definition 1.** *For $\beta \in (0,1]$, a function $f\colon 2^U \to \mathbb{R}_{\geq 0}$ is called* β-accountable *if, for every $S \subseteq U$, there exists an ordering $(e_1,\dots,e_{|S|})$ of the elements in S such that, for all $i \in \{1,\dots,|S|\}$,*

$$f(\{e_1,\dots,e_i\}) \geq \beta\frac{i}{|S|}f(S).$$

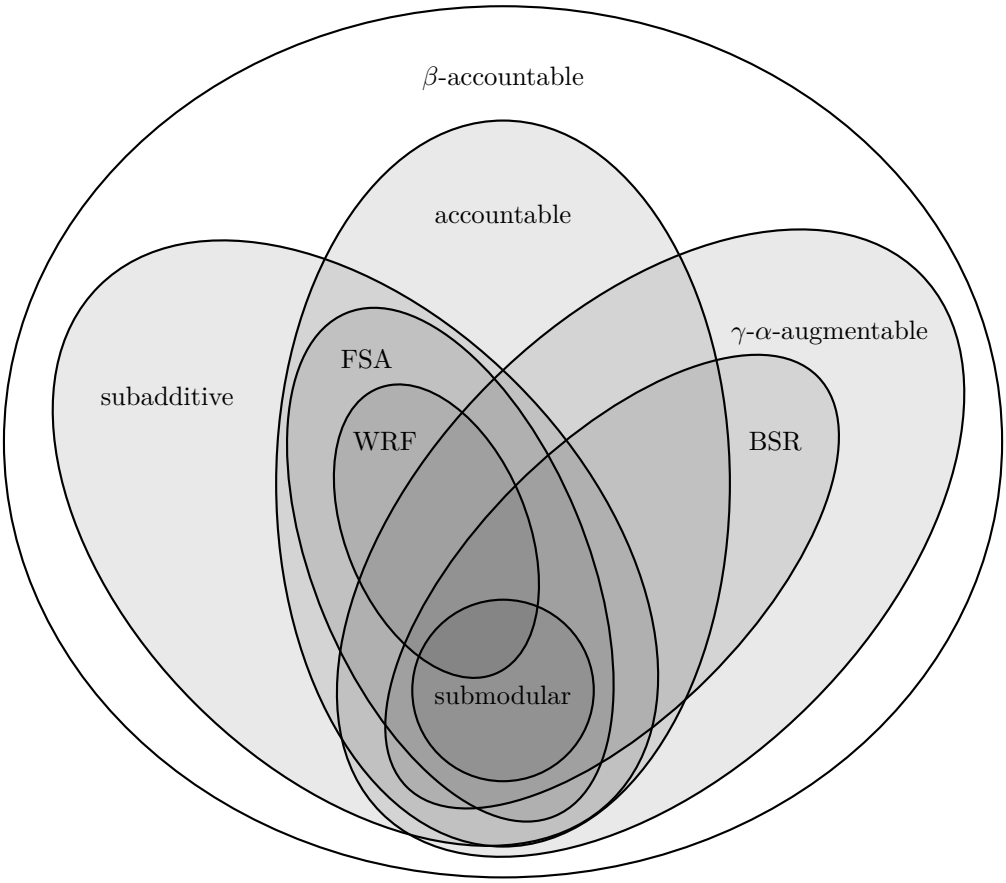


Figure 1 Relation of different function properties, including fractionally subadditivity (FSA), weighted rank functions (WRF), and functions of bounded submodularity ratio (BSR). For the depicted inclusions to hold simultaneously, we can set $\beta = \min\{\frac{1}{2}, \frac{\gamma}{\alpha}\}$.

This definition generalizes multiple other function properties (see Figure 1). In particular, accountable functions [10] correspond exactly to 1-accountable functions (by definition and [10]), every (monotone) γ-α-augmentable function [11] is $\frac{\gamma}{\alpha}$-accountable (Proposition 12), and every (monotone) subadditive function is $\frac{1}{2}$-accountable (Proposition 8).

We establish a general upper bound on the competitive ratio of incremental maximization with β-accountable objective via a scaling algorithm parameterized in β, and complement this result by a lower bound that is tight in the limit $\beta \to 0$ (see Figure 2).

▶ **Theorem 2.** *IncMax with a β-accountable objective has competitive ratio*

$$\rho \in \left[\frac{1}{\beta} \cdot \left(1 + \frac{1}{\lceil \frac{1}{\beta} \rceil + 1}\right), \frac{1}{2\beta} + 1 + \sqrt{\frac{1}{4\beta^2} + 1}\right].$$

Note that, for $\beta = 1$, the upper bound recovers the bound of $\varphi + 1 \approx 2.618$ of [1].

Importantly, Theorem 2 yields the first constant upper bound for the natural class of subadditive objective functions ($\beta = \frac{1}{2}$).

▶ **Theorem 3.** *IncMax with a subadditive objective has competitive ratio at most $2 + \sqrt{2}$.*

Furthermore, Theorem 2 also yields the following bound, which improves on the best known upper bound for γ-α-augmentable functions of [11] when $\gamma < \alpha \frac{e^\alpha}{e^\alpha - 1}$.

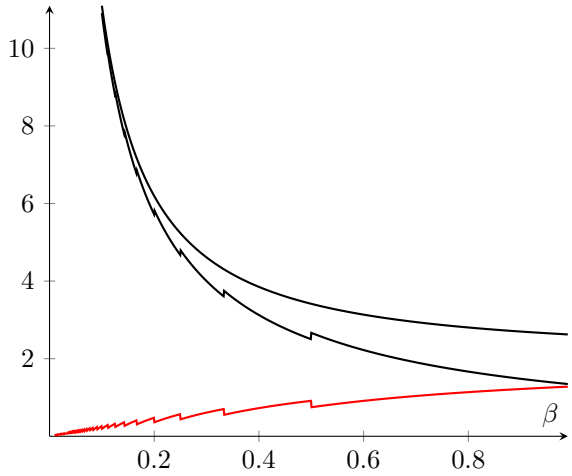


Figure 2 Plot of the bounds of Theorem 2, and their difference (red).

▶ **Theorem 4.** *IncMax with a γ-α-augmentable objective has competitive ratio at most*

$$\frac{\alpha}{2\gamma} + 1 + \sqrt{\frac{\alpha^2}{4\gamma^2} + 1}.$$

Related work

The general framework of incremental maximization was first proposed by Bernstein et al. [1]. They assumed that the objective is accountable and showed that in this case the competitive ratio of this problem is at most $\varphi + 1 \approx 2.618$. Utilizing a continuization technique, Disser et al. [10] showed a lower bound of 2.246 in this setting and provided evidence that the upper bound of $\varphi + 1$ is tight. Before the general framework was introduced, Zhu et al. [30] considered a special case of the problem where edges have to be added over time in order to maximize the number of internal nodes. They proved an upper bound of 12/7.

The performance of a simple greedy algorithm for this problem was investigated separately. Rado [27] showed that the greedy algorithm performs optimally if the objective function is modular. For submodular objectives, Nemhauser et al. [25] showed that the greedy algorithm has a competitive ratio of at most $\frac{\mathrm{e}}{\mathrm{e}-1}$. This is known to be tight due to Feige [12]. More recently, other classes of objectives were considered. Das and Kempe [6] introduced the submodularity ratio as a relaxation of submodularity and showed that the greedy algorithm has a competitive ratio of at most $\frac{\mathrm{e}^\gamma}{\mathrm{e}^\gamma-1}$ for objectives with submodularity ratio $\gamma \in [0, 1]$. This was later shown to be tight by Bian et al. [2] who also parametrized the bound by the curvature. A different relaxation of submodularity is α-augmentability. Here, the competitive ratio of the greedy algorithm is known to be at most $\frac{\mathrm{e}^\alpha}{\mathrm{e}^\alpha-1}$ due to Bernstein et al. [1]. Disser and Weckbecker [11] showed that this upper bound is tight and generalized this result by showing that the greedy algorithm has a competitive ratio of exactly $\frac{\alpha}{\gamma} \cdot \frac{\mathrm{e}^\alpha}{\mathrm{e}^\alpha-1}$ if the objective is γ-α-augmentable.

A generalization of the incremental maximization is the problem where, instead of an increasing cardinality constraint, one is faced with an increasing knapsack constraint. Under the assumption that the objective is modular, Megow and Mestre [22] provide an instance-sensitive near-optimal solution. Disser et al. [9] additionally assume that items that do not fit into the knapsack can be discarded and show that the competitive ratio in this setting is exactly 2. This result was later generalized to submodular objectives with a bounded curvature by Klimm and Knaack [20]. Kawase et al. [19] gave an upper bound of $\frac{2\mathrm{e}}{\mathrm{e}-1}$ on the

randomized competitive ratio of the incremental maximization problem under a knapsack constraint with submodular objective under the assumption that items that do not fit can be discarded. Instead of discarding items that do not fit, Navarra and Pinotti [24] assumed that every item fits into the knapsack and provided a 2-competitive algorithm for modular objectives. Disser et al. [8] assumed that the ratio between the largest and smallest value of a single element is bounded by some constant M and showed that the competitive ratio of the incremental maximization problem under a knapsack constraint with a fractionally subadditive objective lies between $\max\{2.618, M\}$ and $\max\{3.293\sqrt{M}, 2M\}$.

Disser et al. [7] considered an incremental budget constraint for the prize-collecting Steiner tree problem, where the objective does not obey a diminishing returns property and the authors had to resort to bicriterial approximation guarantees. Other variants of the incremental maximization problem include a variation where the goal is to maximize a sum-objective [18, 14, 29], or incremental solutions in a changing environment [16, 17, 28]. Incremental minimization considered, among others, for the k-center problem [15], the minimum latency problem [3, 13, 4], the k-median problem [23, 5, 21], and the facility location problem [26, 21].

2 Competitive Bounds

The best known algorithm to solve the IncMax problem with an accountable objective was presented in [1]. It follows the idea of calculating cardinalities $c_0, c_1, \dots$ and adding the optimum solution sets for these cardinalities one after the other, where the order of the elements within each set follows Definition 1. The cardinalities $c_0, c_1, \dots$ are calculated by setting $c_0 = 1$ and iteratively scaling via $c_{i+1} = \lceil \delta c_i \rceil$ for some $\delta > 1$.

Since β-accountability is closely related to accountability, we introduce a modified version of this algorithm to find an incremental solution for the IncMax problem with a β-accountable objective.

The algorithm Scaling$_\beta$ uses the scaling parameter

$$\delta = \frac{1}{2\beta} + 1 + \sqrt{\frac{1}{4\beta^2} + 1}$$

and chooses

$$c_1 \in \arg\max_{C \in \mathbb{N}} \frac{\textsc{Opt}(C)}{C}$$

in an arbitrary, but fixed way. Then, for $i \in \mathbb{N}$, it chooses

$$c_{i+1} \in \arg\max_{C \in \mathbb{N}, C \geq \delta c_i} \frac{\textsc{Opt}(C)}{C},$$

also in an arbitrary, but fixed way. Scaling$_\beta$ operates in phases, and in phase $i \in \mathbb{N}$, it adds the optimum solution of cardinality c_i in the order given by Definition 1.

We state the following observation.

▶ **Observation 5.** *For all $i \in \mathbb{N}$, we have*

(i) $c_{i+1} \geq \delta c_i$,
(ii) $\sum_{j=1}^{i} c_j \leq \frac{\delta}{\delta - 1} c_i$,
(iii) $\frac{\textsc{Opt}(c_i)}{c_i} \geq \frac{\textsc{Opt}(c)}{c}$ *for all* $c \in \mathbb{N}_{\geq \delta c_i}$.

Proof. (i) and (iii) follow immediately from the definition of c_i for all $i \in \mathbb{N}$.

Furthermore, we have

$$\sum_{j=1}^{i} c_j \overset{(i)}{\leq} \sum_{j=1}^{i} \left(\frac{1}{\delta}\right)^{i-j} c_i \leq \sum_{j=1}^{\infty} \left(\frac{1}{\delta}\right)^{i-j} c_i = \frac{\delta}{\delta - 1} c_i.$$

◀

We are now ready to prove an upper bound on the competitive ratio of $\textsc{Scaling}_\beta$.

▶ **Theorem 6.** *The algorithm $\textsc{Scaling}_\beta$ is δ-competitive for IncMax with a β-accountable objective function.*

Proof. Let π denote the incremental solution of $\textsc{Scaling}_\beta$. Let $C \in \mathbb{N}$ and $i \in \mathbb{N}$ such that $C \in \left(\sum_{j=1}^{i-1} c_j, \sum_{j=1}^{i} c_j\right]$. If $i = 1$, we let $x = 0$, otherwise, let

$$x = \frac{1}{\beta} \cdot \frac{\textsc{Opt}(c_{i-1})}{\textsc{Opt}(c_i)} c_i + \sum_{j=1}^{i-1} c_j, \tag{1}$$

i.e., we have

$$\textsc{Opt}(c_{i-1}) = \beta \frac{x - \sum_{j=1}^{i-1} c_j}{c_i} \textsc{Opt}(c_i).$$

The value x is chosen such that, if $C \geq x$, then the value of the partially added optimum solution of cardinality c_i in the solution $\pi(C)$ is at least as large as the value of the completely contained optimum solution of cardinality c_{i-1}.

Case 1: $C < \lceil \delta c_{i-1} \rceil$.
As $\pi(C)$ contains the optimum solution of cardinality c_{i-1}, by monotonicity, we have

$$\frac{\textsc{Opt}(C)}{f(\pi(C))} \leq \frac{\textsc{Opt}(C)}{\textsc{Opt}(c_{i-1})} \leq \frac{\textsc{Opt}(\lceil \delta c_{i-1} \rceil - 1)}{\textsc{Opt}(c_{i-1})} \overset{\text{Obs. 5 } (iii)}{\leq} \frac{\lceil \delta c_{i-1} \rceil - 1}{c_{i-1}} \leq \delta.$$

Case 2: $\lceil \delta c_{i-1} \rceil \leq C < x$.
Note that $C < x$ implies that $x > 0$, i.e., (1) holds. The solution $\pi(C)$ contains the optimum solution of cardinality c_{i-1}. Thus,

$$\begin{aligned}
\frac{\textsc{Opt}(C)}{f(\pi(C))} &\leq \frac{\textsc{Opt}(C)}{\textsc{Opt}(c_{i-1})} \\
&\overset{\text{Obs. 5 } (iii)}{\leq} \frac{\textsc{Opt}(c_i)}{\textsc{Opt}(c_{i-1})} \cdot \frac{C}{c_i} \\
&\leq \frac{\textsc{Opt}(c_i)}{\textsc{Opt}(c_{i-1})} \cdot \frac{x}{c_i} \\
&\overset{(1)}{=} \frac{1}{\beta} + \frac{\textsc{Opt}(c_i)}{\textsc{Opt}(c_{i-1})} \cdot \frac{1}{c_i} \sum_{j=1}^{i-1} c_j \\
&\overset{\text{Obs. 5 } (iii)}{\leq} \frac{1}{\beta} + \frac{c_i}{c_{i-1}} \cdot \frac{1}{c_i} \sum_{j=1}^{i-1} c_j \\
&\overset{\text{Obs. 5 } (ii)}{\leq} \frac{1}{\beta} + \frac{\delta}{\delta - 1} = \delta,
\end{aligned}$$

where the last equality follows from the definition of δ.

Case 3: $C \geq \lceil \delta c_{i-1} \rceil$, $C \geq x$.
If $i = 1$, by definition of c_1, we have

$$\frac{\textsc{Opt}(C)}{f(\pi(C))} \leq \frac{\textsc{Opt}(C)}{\beta \frac{C}{c_1} \textsc{Opt}(c_1)} \leq \frac{1}{\beta} < \delta.$$

Now, assume that $i \geq 2$, i.e., (1) holds. The solution $\pi(C)$ contains $C - \sum_{j=1}^{i-1} c_j$ elements from the optimum solution of cardinality c_i. Thus,

$$\begin{aligned}
\frac{\textsc{Opt}(C)}{f(\pi(C))} &\leq \frac{\textsc{Opt}(C)}{\beta\frac{C-\sum_{j=1}^{i-1} c_j}{c_i}\textsc{Opt}(c_i)} \\
&\overset{\text{Obs. 5 } (iii)}{\leq} \frac{1}{\beta}\cdot\frac{C}{C-\sum_{j=1}^{i-1} c_j} \\
&\overset{C\geq x}{\leq} \frac{1}{\beta}\cdot\frac{x}{x-\sum_{j=1}^{i-1} c_j} \\
&\overset{(1)}{=} \frac{1}{\beta}\left(1+\frac{\sum_{j=1}^{i-1} c_j}{\frac{1}{\beta}\cdot\frac{\textsc{Opt}(c_{i-1})}{\textsc{Opt}(c_i)}c_i}\right) \\
&= \frac{1}{\beta}+\frac{\textsc{Opt}(c_i)}{\textsc{Opt}(c_{i-1})}\cdot\frac{1}{c_i}\sum_{j=1}^{i-1} c_j \\
&\overset{\text{Obs. 5 } (iii)}{\leq} \frac{1}{\beta}+\frac{c_i}{c_{i-1}}\cdot\frac{1}{c_i}\sum_{j=1}^{i-1} c_j \\
&\overset{\text{Obs. 5 } (ii)}{\leq} \frac{1}{\beta}+\frac{\delta}{\delta-1} \\
&= \delta.
\end{aligned}$$

◀

We complement this upper bound with a lower bound, that, in particular, shows that for $\beta \to 0$, we cannot be better than $\frac{1}{\beta}$-competitive.

▶ **Theorem 7.** *For all $\beta \in (0,1]$, the competitive ratio of* IncMax *with a β-accountable objective function is at least*

$$\frac{1}{\beta}\cdot\left(1+\frac{1}{\lceil\frac{1}{\beta}\rceil+1}\right).$$

Proof. Let $k := \lceil\frac{1}{\beta}\rceil+2$ and $d := \frac{k-1}{k}\beta$. We define an instance where no incremental solution can have a competitive ratio better than $\frac{1}{d}$. Let $U = \{e_1,\dots,e_{k+1}\}$ be the groundset and $f\colon 2^U \to \mathbb{R}_{\geq 0}$ be the objective such that, for $S \subseteq U$,

$$f(S) = \begin{cases} \frac{kd}{\beta} = k-1, & \text{if } \{e_2,\dots,e_{k+1}\} \subseteq S, \\ \max\big\{|\{e_1\}\cap S|, |\{e_2,\dots,e_{k+1}\}\cap S|\cdot d\big\}, & \text{else.} \end{cases}$$

This objective is monotone because of the maximum in the definition and because $\frac{kd}{\beta} \geq kd$. We show that f is β-accountable. For this, let $S \subseteq U$. We have to show that there is an ordering $(e_{i_1},\dots,e_{i_{|S|}})$ of the elements in S such that $f(\{e_{i_1},\dots,e_{i_j}\}) \geq \beta\frac{j}{|S|}f(S)$ for all $j \in \{1,\dots,|S|\}$. If we have $e_1 \in S$ and $f(S) = 1$, we can simply choose e_1 to be the first element in the ordering and obtain

$$f(\{e_{i_1},\dots,e_{i_j}\}) = 1 = f(S) \geq \beta\frac{j}{|S|}f(S)$$

for all $j \in \{1,\dots,|S|\}$. Otherwise, if $e_1 \notin S$ or $f(S) > 1$, then, with $S' := S \cap \{e_2,\dots,e_{k+1}\}$, either $f(S) = \frac{kd}{\beta} = \frac{|S'|d}{\beta}$, or $f(S) = |S'|d \leq \frac{|S'|d}{\beta}$. In our ordering, we can put the elements from S' in the beginning and, for $j \in \{1,\dots,|S'|\}$, obtain

$$f(\{e_{i_1},\dots,e_{i_j}\}) \geq j\cdot d = \beta\frac{j}{|S'|}\frac{|S'|d}{\beta} \geq \beta\frac{j}{|S'|}f(S) \geq \beta\frac{j}{|S|}f(S).$$

If $S' = S$, we are done. Otherwise, $|S| = |S'| + 1$ holds and, for $j = |S|$, we have $f(\{e_{i_1}, \dots, e_{i_j}\}) = f(S) \geq \beta \frac{j}{|S|} f(S)$.

Let π be an incremental solution for this instance. We consider two cases. First, assume that e_1 is not the last element in the ordering π. We have

$$(k-1)d = \frac{\left(\left\lceil \frac{1}{\beta} \right\rceil + 1\right)^2}{\left\lceil \frac{1}{\beta} \right\rceil + 2} \beta > \frac{\left\lceil \frac{1}{\beta} \right\rceil^2 + 2\left\lceil \frac{1}{\beta} \right\rceil}{\left\lceil \frac{1}{\beta} \right\rceil + 2} \beta = \left\lceil \frac{1}{\beta} \right\rceil \beta \geq 1.$$

The solution $\pi(k)$ contains e_1 and $k-1$ elements from $\{e_2, \dots, e_{k+1}\}$. Thus,

$$f(\pi(k)) = \max\{1, (k-1)d\} = (k-1)d.$$

The optimum solution of cardinality k is the set $\{e_2, \dots, e_{k+1}\}$ with a value of $k-1$. Thus, in this case, the competitive ratio of π is at least $\frac{1}{d}$.

Now, consider the other case, i.e., e_1 is the last element in the ordering π. Then the solution $\pi(1)$ contains exactly one element from the set $\{e_2, \dots, e_{k+1}\}$ and has therefore value $f(\pi(1)) = d$. The optimum solution of cardinality 1 is $\{e_1\}$ with a value of 1. Thus, also in this case, the competitive ratio of π is at least $\frac{1}{d}$. ◀

In Figure 2, we plot the upper bound from Theorem 6 and the lower bound from Theorem 7 in black, as well as a plot of their difference in red. On the one hand, one can see that both bounds are unbounded for $\beta \to 0$. This seems plausible because for $\beta \to 0$, we have (almost) no guarantee that the value of large sets is bounded by the value of smaller sets. On the other hand, one can see in Figure 2 that the difference between upper and lower bound is almost 0 for $\beta \to 0$. Thus, in the limit $\beta \to 0$, ScalingBeta performs optimally.

3 Incremental Maximization for Subclasses

We employ the previous result to obtain new and improved bounds on the competitive ratio of IncMax with subadditive and γ-α-augmentable objectives.

3.1 Subadditivity

In order to derive results for IncMax with subadditive objectives, we compare β-accountability and subadditivity.

▶ **Proposition 8.** *Every monotone, subadditive function is $\frac{1}{2}$-accountable.*

Proof. Let $f\colon 2^U \to \mathbb{R}_{\geq 0}$ be monotone and subadditive, $S \subseteq U$ be finite, and $k := |S|$. We define $\ell := \lceil \log_2 k \rceil$, i.e., we have $2^{\ell-1} < k \leq 2^\ell$. Furthermore, we define $S_\ell := S$ and iteratively, for $j \in \{\ell - 1, \dots, 0\}$, $S_j \subseteq S_{j+1}$ with $|S_j| = \left\lceil \frac{k}{2^{\ell-j}} \right\rceil$ and $f(S_j) \geq \frac{1}{2} f(S_{j+1})$. This is possible as we will see now. We have $2\left\lceil \frac{k}{2^{\ell-j}} \right\rceil = \left\lceil 2\left\lceil \frac{k}{2^{\ell-j}} \right\rceil \right\rceil \geq \left\lceil \frac{k}{2^{\ell-j-1}} \right\rceil$, i.e., we can choose $A, B \subseteq S_j$ with $|A| = |B| = \left\lceil \frac{k}{2^{\ell-j}} \right\rceil$ and $A \cup B = S_{j+1}$. By subadditivity, we have $f(A) \geq \frac{1}{2} f(S_{j+1})$ or $f(B) \geq \frac{1}{2} f(S_{j+1})$. Thus, we can choose $S_j \in \{A, B\}$ with the desired properties.

Now, consider any order $(e_1, \dots, e_k)$ of S with $\{e_1, \dots, e_{|S_j|}\} = S_j$ for all $j \in \{0, \dots, \ell\}$. Let $i \in \{1, \dots, k\}$ and $j \in \{0, \dots, \ell\}$ such that $\left\lceil \frac{k}{2^{\ell-j}} \right\rceil \leq i < \left\lceil \frac{k}{2^{\ell-j-1}} \right\rceil$. This implies that $S_j \subseteq \{e_1, \dots, e_i\}$ and, because $i \in \mathbb{N}$, that $i < \frac{k}{2^{\ell-j-1}}$. Together with monotonicity of f, we obtain

$$f(\{e_1, \dots, e_i\}) \geq f(S_j) \geq \left(\frac{1}{2}\right)^{\ell-j} f(S_\ell) = \frac{1}{2^{\ell-j}} f(S) \geq \frac{1}{2} \frac{i}{k} f(S),$$

which yields $\frac{1}{2}$-accountability. ◀

We combine Theorem 6 and Proposition 8 to obtain an upper bound on the competitive ratio of IncMax with a subadditive objective. This immediately proves Theorem 3.

▶ **Theorem 3.** *IncMax with a subadditive objective has competitive ratio at most* $2+\sqrt{2}$.

We complement this upper bound on the competitive ratio of IncMax with subadditive objective functions by a lower bound. To this end, we employ separability of problem instances of incremental maximization, a property introduced in [10].

▶ **Definition 9.** *An instance of IncMax with objective* $f\colon 2^U \to \mathbb{R}_{\geq 0}$ *is called* separable *if there exist a partition* $U = U_1 \cup U_2 \cup \dots$ *of* U *and* densities $d_i > 0$ *such that, for all* $S \subseteq U$,

$$f(S) = \max_{i\in\mathbb{N}}\{|S\cap U_i|\cdot d_i\}.$$

▶ **Proposition 10.** *The objective function of every separable problem instance is subadditive.*

Proof. Let a separable instance with objective function $f\colon 2^U \to \mathbb{R}_{\geq 0}$ be given. Furthermore, let $U = U_1 \cup U_2 \cup \dots$ be a partition of U, and let $d_1, d_2, \dots > 0$ be the densities such that, for all $S \subseteq U$,

$$f(S) = \max_{i\in\mathbb{N}} |S\cap U_i|\cdot d_i.$$

In order to show subadditivity, we fix two sets $A, B \subseteq U$. Let $i^* \in \mathbb{N}$ be the index such that $f(A\cup B) = |(A\cup B)\cap U_{i^*}|\cdot d_{i^*}$. Then

$$\begin{aligned} f(A\cup B) &= |(A\cup B)\cap U_{i^*}|\cdot d_{i^*} \\ &= |A\cap U_{i^*}|\cdot d_{i^*} + |(B\setminus A)\cap U_{i^*}|\cdot d_{i^*} \\ &\leq |A\cap U_{i^*}|\cdot d_{i^*} + |B\cap U_{i^*}|\cdot d_{i^*} \\ &\leq \big(\max_{i\in\mathbb{N}} |A\cap U_i|\cdot d_i\big) + \big(\max_{i\in\mathbb{N}} |B\cap U_i|\cdot d_i\big) \\ &= f(A) + f(B). \end{aligned}$$

◀

This result yields that the (non-strict) competitive ratio of IncMax with subadditive objectives is at least that for separable instances. In [10], the authors show a lower bound of 2.246 on the competitive ratio in the separable setting. Thus, we obtain the following.

▶ **Corollary 11.** *The competitive ratio of IncMax with a subadditive objective function is at least* 2.246.

This bound is weaker than the lower bound of $\frac{8}{3}$ from Theorem 7 for IncMax with a $\frac{1}{2}$-accountable objective. Yet, note that the lower bound construction in the proof of Theorem 7 is not subadditive.

3.2 γ-α-Augmentability

We now turn to comparing β-accountability and γ-α-augmentability.

▶ **Proposition 12.** *For all* $\gamma \in (0,1]$ *and* $\alpha \geq 1$, *every monotone,* γ-α*-augmentable function is* $\frac{\gamma}{\alpha}$*-accountable.*

Proof. For $\gamma \in (0,1]$ and $\alpha \geq 1$, let $f\colon 2^U \to \mathbb{R}_{\geq 0}$ be monotone and γ-α-augmentable. Let $S \subseteq U$ be finite and $k := |S|$. By γ-α-augmentability, there exists an ordering $(e_1, \dots, e_k)$ of the elements in S such that, for all $i \in \{0, \dots, k-1\}$,

$$f(\{e_1,\dots,e_{i+1}\}) - f(\{e_1,\dots,e_i\}) \geq \frac{\gamma f(S) - \alpha f(\{e_1,\dots,e_i\})}{k-i}.$$

For $i \in \{0, \dots, k\}$, let $S_i := \{e_1, \dots, e_i\}$. Then, for $i \in \{0, \dots, k-1\}$, this yields

$$f(S_{i+1}) - f(S_i) \geq \alpha \frac{\frac{\gamma}{\alpha} f(S) - f(S_i)}{k-i} \overset{\alpha \geq 1}{\geq} \frac{\frac{\gamma}{\alpha} f(S) - f(S_i)}{k-i}. \tag{2}$$

To show $\frac{\gamma}{\alpha}$-accountability of f, we prove by induction that, for $i \in \{1, \dots, k\}$, we have

$$f(S_i) \geq \frac{\gamma}{\alpha} \frac{i}{k} f(S). \tag{3}$$

For $i = 0$, (2) yields

$$f(S_1) \geq \frac{1}{k}\left(\frac{\gamma}{\alpha} f(S) - f(\emptyset)\right) + f(\emptyset) = \frac{\gamma}{\alpha}\frac{1}{k} f(S) + \left(1 - \frac{1}{k}\right) f(\emptyset) \geq \frac{\gamma}{\alpha}\frac{1}{k} f(S),$$

where the last inequality follows from non-negativity of f.

Now, suppose that (3) holds for some $i \in \{0, \dots, k-1\}$. Then

$$\begin{aligned}
f(S_{i+1}) &\overset{(2)}{\geq} f(S_i) + \frac{\frac{\gamma}{\alpha} f(S) - f(S_i)}{k-i} \\
&= \frac{\gamma}{\alpha}\frac{1}{k-i} f(S) + \left(1 - \frac{1}{k-i}\right) f(S_i) \\
&\overset{(3)}{\geq} \frac{\gamma}{\alpha}\frac{1}{k-i} f(S) + \left(\frac{k-i-1}{k-i}\right)\frac{\gamma}{\alpha}\frac{i}{k} f(S) \\
&= \frac{\gamma}{\alpha}\frac{ki + k - i^2 - i}{(k-i)k} f(S) \\
&= \frac{\gamma}{\alpha}\frac{(k-i)(i+1)}{(k-i)k} f(S) \\
&= \frac{\gamma}{\alpha}\frac{i+1}{k} f(S),
\end{aligned}$$

which concludes the induction. ◀

We combine the results from Proposition 12 and Theorem 6 to obtain an upper bound on the competitive ratio of IncMax with a monotone and γ-α-augmentable objective. This immediately proves Theorem 4.

▶ **Theorem 13.** *For $\gamma \in (0, 1]$ and $\alpha \geq 1$, $\textsc{Scaling}_{\gamma/\alpha}$ has a competitive ratio of at most*

$$\frac{\alpha}{2\gamma} + 1 + \sqrt{\frac{\alpha^2}{4\gamma^2} + 1}$$

for IncMax with a γ-α-augmentable objective.

We compare this upper bound on the competitive ratio of IncMax with a γ-α-augmentable objective to

$$\frac{\alpha}{\gamma} \cdot \frac{\mathrm{e}^\alpha}{\mathrm{e}^\alpha - 1},$$

the competitive ratio of the greedy algorithm for IncMax with γ-α-augmentable objectives [11]. Let the ratio between the two upper bounds be denoted by

$$r(\gamma, \alpha) := \frac{\frac{\alpha}{2\gamma} + 1 + \sqrt{\frac{\alpha^2}{4\gamma^2} + 1}}{\frac{\alpha}{\gamma} \cdot \frac{\mathrm{e}^\alpha}{\mathrm{e}^\alpha - 1}} = \frac{\mathrm{e}^\alpha - 1}{\mathrm{e}^\alpha} \cdot \left(\frac{1}{2} + \frac{\gamma}{\alpha} + \sqrt{\frac{1}{4} + \frac{\gamma^2}{\alpha^2}}\right).$$

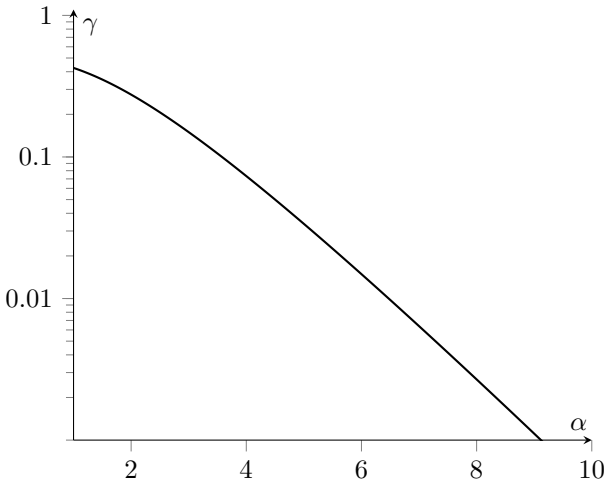


Figure 3 Plot of the value of γ depending on α for which $\textsc{Scaling}_{\gamma/\alpha}$ and the greedy algorithm achieve the same competitive ratio. For values of γ below this line, $\textsc{Scaling}_{\gamma/\alpha}$ performs better, and vice versa.

We have $\lim_{\gamma\to 0} r(\gamma,\alpha) = \frac{e^\alpha-1}{e^\alpha} < 1$, i.e., for small values of γ, $\textsc{Scaling}_{\gamma/\alpha}$ performs better than the greedy algorithm. Since $\gamma \in (0,1]$ and $\alpha \geq 1$, we have $r(\gamma,\alpha) = 1$ if and only if $\gamma = \alpha\frac{e^\alpha}{e^{2\alpha}-1}$. This value lies in the interval $\left(0, \frac{e}{e^2-1}\right] \subseteq (0, 0.426)$, and for $\alpha \to \infty$, approaches 0 (cf. Figure 3). Thus, for large $\alpha \geq 1$, the greedy algorithm performs better than $\textsc{Scaling}_{\gamma/\alpha}$ for almost all values of $\gamma \in (0,1]$. This is probably due to the fact that γ-α-augmentability is a property that relaxes an inequality that is a core estimate in the analysis of the the greedy algorithm for monotone and submodular functions.

References

1 Aaron Bernstein, Yann Disser, Martin Groß, and Sandra Himburg. General bounds for incremental maximization. *Mathematical Programming*, 191(2):953–979, 2022. doi:10.1007/s10107-020-01576-0.

2 Andrew An Bian, Joachim M. Buhmann, Andreas Krause, and Sebastian Tschiatschek. Guarantees for greedy maximization of non-submodular functions with applications. In *Proceedings of the 34th Interantional Conference on Machine Learning (ICML)*, pages 498–507, 2017. URL: https://proceedings.mlr.press/v70/bian17a.html.

3 Avrim Blum, Prasad Chalasani, Don Coppersmith, William R. Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. The minimum latency problem. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 163–171, 1994. doi:10.1145/195058.195125.

4 Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 36–45, 2003. doi:10.1109/SFCS.2003.1238179.

5 Marek Chrobak, Claire Kenyon, John Noga, and Neal E. Young. Incremental medians via online bidding. *Algorithmica*, 50(4):455–478, 2008. doi:10.1007/s00453-007-9005-x.

6 Abhimanyu Das and David Kempe. Approximate submodularity and its applications: Subset selection, sparse approximation and dictionary selection. *Journal of Machine Learning Research*, 19:1–34, 2018. URL: https://jmlr.org/papers/v19/16-534.html.

7 Yann Disser, Svenja M. Griesbach, Max Klimm, and Annette Lutz. Bicriterial approximation for the incremental prize-collecting steiner-tree problem. In *ESA 2024*, pages 47:1–47:16, 2024. doi:10.4230/LIPIcs.ESA.2024.47.

8 Yann Disser, Max Klimm, Annette Lutz, and David Weckbecker. Fractionally subadditive maximization under an incremental knapsack constraint. *SIAM Journal on Discrete Mathematics*, 38(1):764–789, 2024.

9 Yann Disser, Max Klimm, Nicole Megow, and Sebastian Stiller. Packing a knapsack of unknown capacity. *SIAM Journal on Discrete Mathematics*, 31(3):1477–1497, 2017. doi:10.1137/16M1070049.

10 Yann Disser, Max Klimm, Kevin Schewior, and David Weckbecker. Incremental maximization via continuization. In *50th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261, pages 47:1–47:17, 2023. doi:10.4230/LIPIcs.ICALP.2023.47.

11 Yann Disser and David Weckbecker. Unified greedy approximability beyond submodular maximization. *SIAM Journal on Discrete Mathematics*, 38(1):248–379, 2024. doi:10.1137/23M1569265.

12 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

13 Michel X. Goemans and Jon M. Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82:111–124, 1998. doi:10.1007/BF01585867.

14 Michel X. Goemans and Francisco Unda. Approximating incremental combinatorial optimization problems. In *Proceedings of the 20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 6:1–6:14, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.6.

15 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

16 Jeff Hartline and Alexa Sharp. An incremental model for combinatorial maximization problems. In *Proceedings of the 5th International Workshop on Experimental Algorithms (WEA)*, pages 36–48, 2006. doi:10.1007/11764298.

17 Jeff Hartline and Alexa Sharp. Incremental flow. *Networks*, 50(1):77–85, 2007. doi:10.1002/net.20168.

18 Thomas Kalinowski, Dmytro Matsypura, and Martin W.P. Savelsbergh. Incremental network design with maximum flows. *European Journal of Operational Research*, 242(1):51–62, 2015. doi:10.1016/j.ejor.2014.10.003.

19 Yasushi Kawase, Hanna Sumita, and Takuro Fukunaga. Submodular maximization with uncertain knapsack capacity. *SIAM Journal on Discrete Mathematics*, 33(3):1121–1145, 2019. doi:10.1137/18M1174428.

20 Max Klimm and Martin Knaack. Maximizing a submodular function with bounded curvature under an unknown knapsack constraint. In *Proceedings of the 25th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 49:1–49:19, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.49.

21 Guolong Lin, Chandrashekhar Nagarajan, Rajmohan Rajaraman, and David P. Williamson. A general approach for incremental approximation and hierarchical clustering. *SIAM Journal on Computing*, 39(8):3633–3669, 2010. doi:10.1137/070698257.

22 Nicole Megow and Julián Mestre. Instance-sensitive robustness guarantees for sequencing with unknown packing and covering constraints. In *Proceedings of the 4th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 495–504, 2013. doi:10.1145/2422436.2422490.

23 Ramgopal R. Mettu and C. Greg Plaxton. The online median problem. *SIAM Journal on Computing*, 32(3):816–832, 2003. doi:10.1137/S0097539701383443.

24 Alfredo Navarra and Cristina M. Pinotti. Online knapsack of unknown capacity: How to optimize energy consumption in smartphones. *Theoretical Computer Science*, 697:98–109, 2017. doi:10.1016/j.tcs.2017.07.029.

25 George Nemhauser, Laurence Wolsey, and Michael Fisher. An analysis of approximations for maximizing submodular set functions - I. *Mathematical Programming*, 14:265–294, 1978. doi:10.1007/BF01588971.

26 C. Greg Plaxton. Approximation algorithms for hierarchical location problems. *Journal of Computer and System Sciences*, 72(3):425–443, 2006. doi:10.1016/j.jcss.2005.09.004.

27 Richard Rado. A theorem on independence relations. *The Quarterly Journal of Mathematics*, 13:83–89, 1942. doi:10.1093/qmath/os-13.1.83.

28 Clemens Thielen, Morten Tiedemann, and Stephan Westphal. The online knapsack problem with incremental capacity. *Mathematical Methods of Operations Research*, 83(2):207–242, 2016. doi:10.1007/s00186-015-0526-9.

29 Francisco T. Unda Surawski. *Combinatorial Incremental Problems*. PhD thesis, Massachusetts Institute of Technology, 2018. URL: https://dspace.mit.edu/bitstream/handle/1721.1/122890/1126788574-MIT.pdf?sequence=1&isAllowed=y.

30 Xianbin Zhu, Wenjun Li, Yongjie Yang, and Jianxin Wang. Incremental algorithms for the maximum internal spanning tree problem. *Science China Information Sciences*, 64, May 2021. doi:10.1007/s11432-019-2630-2.

Recognizing and Realizing Temporal Reachability Graphs

Thomas Erlebach
Department of Computer Science, Durham University, UK

Othon Michail
Department of Computer Science, University of Liverpool, UK

Nils Morawietz
LaBRI, Université de Bordeaux, Talence, France
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Abstract

A temporal graph $\mathcal{G} = (G, \lambda)$ can be represented by an underlying graph $G = (V, E)$ together with a function λ that assigns to each edge $e \in E$ the set of time steps during which e is present. The reachability graph of $\mathcal{G}$ is the directed graph $D = (V, A)$ with $(u, v) \in A$ if and only if there is a temporal path from u to v. We study the Reachability Graph Realizability (RGR) problem that asks whether a given directed graph $D = (V, A)$ is the reachability graph of some temporal graph. The question can be asked for undirected or directed temporal graphs, for reachability defined via strict or non-strict temporal paths, and with or without restrictions on λ (simple, proper, or both). Answering an open question posed by Casteigts et al. (TCS 2024), we show that all variants of the problem are NP-complete, except for two variants that become trivial in the directed case. For undirected temporal graphs, we consider the complexity of the problem with respect to the solid graph, that is, the graph containing all edges that could potentially receive a label in any realization. We show that the RGR problem is fixed-parameter tractable for the feedback edge set number of the solid graph. As we show, the latter parameter can presumably not be replaced by smaller parameters like feedback vertex set number or treedepth, since the problem is W[2]-hard for them.

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

Keywords and phrases parameterized complexity, temporal graphs, FPT algorithm, feedback edge set, directed graph recognition

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.93

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2503.15771

Funding *Nils Morawietz*: Supported by the French ANR, project ANR-22-CE48-0001 (TEMPOGRAL).

1 Introduction

Temporal graphs are graphs that change over time. The vertex set is often assumed to be fixed, and the edge set can differ from one time step to the next. The study of temporal graphs has attracted significant attention in recent years [5, 22, 23]. One way to represent a temporal graph $\mathcal{G}$ with vertex set V is as a sequence $(G_i)_{i \in [L]}$, where $G_i = (V, E_i)$ is the graph containing the edges that are present in time step i. If an edge e is present in time step i, we refer to (e, i) as a *time edge*, and call i a *time label* of edge e. A *strict temporal path* from vertex u to vertex v in $\mathcal{G}$ is a sequence of time edges forming a u-v-path whose time steps are strictly increasing. A *non-strict temporal path* is defined analogously, except that the time steps only need to be non-decreasing.


© Thomas Erlebach, Othon Michail, and Nils Morawietz;
licensed under Creative Commons License CC-BY 4.0



In this work, we initiate the study of designing temporal graphs whose dynamics *guarantee certain reachability requirements while preventing others*, a problem naturally arising in areas like epidemic control, transportation, and logistics. For example, in epidemic control, a central goal is to maintain a minimal degree of societal connectivity for critical services while preventing broader or specific temporal connections that could facilitate disease spread.

Given a temporal graph $\mathcal{G}$, its temporal reachability relation can be represented as a directed graph $D = (V, A)$, called the *reachability graph* of $\mathcal{G}$, with $(u, v) \in A$ for $u \neq v$ if and only if there exists a (strict or non-strict, respectively) temporal path from u to v in $\mathcal{G}$. We can now state our problem as that of *realizing* a given reachability graph D by finding a temporal graph $\mathcal{G}$ whose temporal connectivity is specified by D: each arc or non-arc of D represents a temporal reachability that *must* or *must not* be realized in $\mathcal{G}$, respectively. In particular, we want to know which directed graphs can arise as reachability graphs of temporal graphs, and how difficult it is to determine for a given directed graph D whether there exists a temporal graph $\mathcal{G}$ with reachability graph D. Interestingly, this formulation turns out to have been posed twice recently as an open question, by Casteigts et al. [3, Open question 5] and Döring [11]. In contrast to our work, those papers aim to understand the expressivity of various temporal graph models, using reachability graphs as a tool.

Part of our contribution is to resolve this open question, by showing that the associated decision problem is NP-complete. Actually, there are a number of variations of the question, as we may ask for an undirected or a directed temporal graph $\mathcal{G}$, for a temporal graph with a restricted kind of labeling (simple, proper, happy)[1], and may consider reachability with respect to strict or non-strict temporal paths. We show that all these variations are NP-hard if we ask for an undirected temporal graph, and also if we ask for a directed temporal graph except for two variations (strict temporal paths with arbitrary or simple labelings) that are known to become trivial in the directed case [11]. See Table 1 for an overview of these results.

On the positive side, we present the following algorithmic results. For a given digraph $D = (V, A)$, we refer to $\{u, v\}$ for $u, v \in V$ as a *solid* edge if both (u, v) and (v, u) are in A. Let $G = (V, E)$ be the undirected graph on V whose edge set is the set of solid edges. We refer to G as the *solid graph* of D. The structural properties of solid graphs turn out to be crucial for our problem. We show that all undirected problem variants can be solved in polynomial time if the solid graph is a tree. Furthermore, we give an FPT algorithm for the feedback edge set number of the solid graph. This parameter can presumably not be replaced by smaller parameters like feedback vertex set, treedepth, or pathwidth, since two undirected versions of our problem turn out to be W[2]-hard for these parameters.

We believe that reachability graphs of temporal graphs form an interesting class of directed graphs whose study has only recently begun. Our results on their structure and on recognizing such graphs could be of independent interest in the context of algorithmic problems in directed graphs.

Related work. As mentioned, our work falls into the area of temporal graph realization problems. In these problems, one is given some data about the behavior of a temporal graph, and the goal is to detect whether there actually is a temporal graph with this behavior (and to compute such a temporal graph if one exists). Klobas et al. [17] introduced the problem of deciding for a given matrix of fastest travel durations and a period Δ whether there exists a simple temporal graph $\mathcal{G}$ with period Δ with the property that the duration of the fastest

[1] simple $\hat{=}$ one label per edge, proper $\hat{=}$ no two adjacent edges share a label, happy $\hat{=}$ both proper and simple

Table 1 The complexity results for all variants of RGR. Red cells indicate NP-hard cases and green cells indicate cases that are trivial (and thus polynomial-time solvable).

undirected	Strict	Non-strict	directed	Strict	Non-strict
Any	Theorem 14	Theorem 14		Lemma 21, [11]	Theorem 22
Simple	Theorem 14	Theorem 14		Lemma 21, [11]	Theorem 22
Proper	Theorem 14			Theorem 22	
Happy	Theorem 14			Theorem 22	

temporal path between any pair of nodes in $\mathcal{G}$ is equal to the value specified in the input matrix. Erlebach et al. [13] extended the problem to a multi-label version, where each edge is allowed up to ℓ time labels. Motivated by the design of transportation networks, a related problem with respect to upper-bounded travel durations was considered [19, 20, 21].

In an early example of a temporal network design problem studied by Kempe et al. [15], the goal was to *reconstruct* a temporal labeling restricted to a single label per edge, so that a designated root reaches via temporal paths all vertices in a set P while avoiding those in a set N. For multi-labeled temporal graphs, Mertzios et al. [18] studied the problem of designing a temporal graph that preserves the reachability relation or all paths of an underlying static graph while minimizing either the *temporality* (maximum number of labels per edge) or the *temporal cost* (total number of labels used). Göbel et al. [14] showed that it is NP-complete to decide whether the edges of a given undirected graph can be labeled with a single label per edge in such a way that each vertex can reach every other vertex via a strict temporal path. Notably, this problem becomes solvable in linear time when only the degree sequence of the underlying graph is given [4]. Other studies have focused on variants of minimizing edge deletions [12], vertex deletions [25], or edge delays [9] to restrict reachability, motivated, for instance, by epidemic containment strategies that limit interactions. Temporal network design is an active area of research, with further related questions explored for example in [1, 16, 7].

Casteigts et al. [3] studied the relationships between the classes of reachability graphs that arise from undirected temporal graphs if different restrictions are placed on the graph (simple, proper, happy [3]) and depending on whether strict or non-strict temporal paths are considered. They showed that reachability with respect to strict temporal paths in arbitrary temporal graphs yields the widest class of reachability graphs while reachability in happy temporal graphs yields the narrowest class. The class of reachability graphs that arise from proper temporal graphs is the same as for non-strict paths in arbitrary temporal graphs, and this class is larger than the class of reachability graphs arising from non-strict reachability in simple temporal graphs. Strict reachability in simple temporal graphs was also shown to lie between the happy case and the general strict case. Döring [11] completed the picture of a two-stranded hierarchy for undirected temporal graphs and also extended the study to directed temporal graphs. As noted earlier, Casteigts et al. [3] also posed the open question of whether there is a characterization of directed graphs that arise as reachability graphs of temporal graphs (or of some restricted subclass of temporal graphs), and how hard it is to decide whether a given directed graph is the reachability graph of some temporal graph. These questions were posed again by Döring [11]. This is in particular of interest because Casteigts et al. [6] showed that several temporal graph problems can be solved in FPT time with respect to temporal parameters defined over the reachability graph. In this paper, we resolve these open questions regarding the complexity of all directed and undirected variants.

Organization of the paper. In Section 2, we provide a formal problem definition and define the notions used in this work. In Section 3, we show upper and lower bounds for the required number of labels per edge in any realization and provide a single exponential algorithm for all problem variants. Afterwards, in Section 4, we analyze properties and define splitting operations based on bridge edges in the solid graph of the undirected versions of our problem. These structural insights will be mainly used in our FPT algorithm in Section 6, but also immediately let us describe a polynomial-time algorithm for instances where the solid graph is a tree in Section 4.2. In Section 5, we then provide NP-hardness results for all undirected problem versions as well as parameterized intractability results for two of them with respect to feedback vertex set number and treedepth. Afterwards, in Section 6, we provide our main algorithmic result: an FPT algorithm with respect to the feedback edge set number fes of the solid graph. This algorithm is achieved in three steps: Firstly, we apply our splitting operations of Section 4 and provide a polynomial-time reduction rule to simplify the instance at hand, such that all we need to deal with is a subset X^* of vertices of size $\mathcal{O}(\text{fes})$ for which the remainder of the graph decomposes into edge-disjoint trees that only interact with X^* via two leaves each. We call such trees *connector trees*. Secondly, we show that we can efficiently extend the set X^* to a set W^* of size $\mathcal{O}(\text{fes})$ such that each resulting connector tree for the set W^* is more or less independent from the remainder of the graph with respect to the interactions of temporal paths in any realization. All these preprocessing steps run in polynomial time. Afterwards, our algorithm enumerates all reasonable labelings on the edges incident with vertices of W^* and tries to extend each such labeling to a realization for D. As we show, there are only FPT many such reasonable labelings and for each such labeling, there are only FPT many possible extensions that need to be checked. Finally, in Section 7, we briefly discuss the directed version of the problem and provide NP-hardness results for all but the two trivial cases. Proofs of statements with ($\star$) are deferred to the full version.

2 Preliminaries

For definitions on parameterized complexity, the Exponential Time Hypothesis (ETH), or parameters like treedepth, we refer to the textbooks [10, 8].

For natural numbers i, j with $i \leq j$ we write $[i]$ for the set $\{1, \dots, i\}$ and $[i, j]$ for the set $\{i, \dots, j\}$. For an undirected graph $G = (V, E)$, we denote an edge between vertices u and v as $\{u, v\}$ or uv. By $N(v) = \{u \in V \mid vu \in E\}$ we denote the set of neighbors of u. The *degree* of v is the size of $N(v)$. An edge $e \in E$ is a *bridge* or *bridge edge* in a graph $G = (V, E)$ if deleting e increases the number of connected components. A vertex of degree 1 is a *pendant* vertex, and an edge incident with a pendant vertex is called a *pendant* edge.

We assume directed graphs have no parallel arcs and no self-loops, and we denote an arc from u to v by (u, v). A directed graph D is a *directed acyclic graph* (DAG) if it does not contain a directed cycle. The *degree* of a vertex v is the number of arcs containing v. For an undirected graph G, the *feedback edge set number* (*feedback vertex set number*) denotes the minimum number of edges (vertices) to remove from G to obtain an acyclic graph.

A temporal graph $\mathcal{G}$ with vertex set V and lifetime L is given by a sequence $(G_t)_{t\in[L]}$ of L static graphs $G_t = (V, E_t)$ referred to as *snapshots* or *layers*. The graph $\mathcal{G}_\downarrow = (V, E_\downarrow)$ with $E_\downarrow = \bigcup_{i\in[L]} E_i$ is called the *underlying graph* of $\mathcal{G}$. Alternatively, $\mathcal{G}$ can be represented by an undirected graph $G = (V, E)$ with $E \supseteq \bigcup_{t\in[1,L]} E_t$ and a labeling function $\lambda : E \to 2^{[1,L]}$ that assigns to each edge e the (possibly empty) set of time steps during which e is present, that is, $\lambda(e) = \{t \in [1, L] \mid e \in E_t\}$. We write $\mathcal{G} = (G, \lambda)$ in this case. In this representation we allow G to contain extra edges in addition to the edges of the underlying graph; such

edges e satisfy $\lambda(e) = \emptyset$. This is useful because in the problems we consider in this paper, there is a natural choice of a graph $G = (V, E)$, the *solid graph* defined below, that contains all edges of the underlying graph of every realization but may contain additional edges.

We assume that each layer of a temporal graph is an undirected graph unless we explicitly refer to directed temporal graphs. If $t \in \lambda(e)$, we refer to (e, t) as a *time edge*. A *temporal path* from u to v in $\mathcal{G}$ is a sequence of time edges $((e_j, t_j))_{j\in[\ell]}$ for some ℓ such that $(e_1, e_2, \ldots, e_\ell)$ is a u-v path in G and $t_1 \leq t_2 \leq \cdots \leq t_\ell$ holds in the non-strict case and $t_1 < t_2 < \cdots < t_\ell$ holds in the strict case.

A temporal graph is *proper* if no two adjacent edges share a label, *simple* if every edge has a single label, and *happy* if it is both proper and simple [3]. The reachability graph $\mathcal{R}(\mathcal{G})$ of a temporal graph $\mathcal{G}$ with vertex set V is the directed graph (V, A) with the same vertex set V and $(u, v) \in A$ if and only if $u \neq v$ and $\mathcal{G}$ contains a temporal path from u to v. Note that $\mathcal{R}(\mathcal{G})$ depends on whether we consider strict or non-strict temporal paths and can be computed in polynomial time in both cases [2]. We are interested in the following problem:

Reachability Graph Realizability (RGR)
Input: A simple directed graph $D = (V, A)$.
Question: Does there exist a temporal graph $\mathcal{G}$ with $\mathcal{R}(\mathcal{G}) = D$?

For yes-instances of RGR, we are also interested in computing a temporal graph $\mathcal{G}$ with $\mathcal{R}(\mathcal{G}) = D$. We refer to such a temporal graph as a *solution* or a *realization* for D, and we typically represent it by a labeling function. With the adjacency matrix representation of D in mind, we also write $D_{uv} = 1$ for $(u, v) \in A$ and $D_{uv} = 0$ for $(u, v) \notin A$.

We can consider RGR with respect to reachability via strict temporal paths or with respect to non-strict temporal paths. Furthermore, we can require the realization of D to be simple, proper, or happy. For proper and happy temporal graphs, strict and non-strict reachability coincide. Therefore, the distinct problem variants that we can consider are Any Strict RGR, Any Non-strict RGR, Simple Strict RGR, Simple Non-strict RGR, Proper RGR, and Happy RGR. Finally, we write DRGR instead of RGR if we are asking for a directed temporal graph that realizes D. We sometimes write URGR if we want to make it explicit that we are asking for an undirected realization.

The solid graph. If $D_{uv} = 1$ and $D_{vu} = 1$ for some $u \neq v$, we say that there is a *solid edge* between u and v. If $D_{uv} = 1$ and $D_{vu} = 0$, we say that there is a *dashed arc* from u to v. We use $G = (V, E)$ to denote the graph on V whose edge set is the set of solid edges, and we refer to this graph as the *solid graph* (of D). For URGR it is clear that only solid edges can receive labels in a realization of D, as the two endpoints of an edge that is present in at least one time step can reach each other. Bridges of the solid graph must receive labels, but a solid edge e that is not a bridge may not receive labels, as the endpoints of e could reach each other via temporal paths of length greater than one.

▶ **Observation 1.** *Let D be an instance of Any Strict URGR or Simple Strict URGR. Let λ be a realization of D and let $\{u, v\}$ and $\{v, w\}$ be solid edges that both receive at least one label under λ. If D contains neither the arc (u, w) nor the arc (w, u), then there is a label α such that $\lambda(\{u, v\}) = \lambda(\{v, w\}) = \{\alpha\}$.*

Note that for Proper URGR, Happy URGR, and all versions of Non-strict URGR, no realization for D can assign labels to both of two adjacent edges $\{u, v\}$ and $\{v, w\}$ if D contains neither (u, w) nor (w, u).

A path $P = (u_0 = u, \dots, u_\ell = v)$ from u to v in G is a *dense* u-v-path if there exist arcs $(u_i, u_j) \in A$ for all $0 \leq i < j \leq \ell$. In each undirected realization of D, each temporal path is a dense path in G.

We say that a realization λ is *frugal* if there is no edge e such that the set of labels assigned to e can be replaced by a smaller set while maintaining the property that λ is a realization. We say that a realization λ is *minimal* on edge e if it is impossible to obtain another realization by replacing $\lambda(e)$ with a proper subset. A realization is minimal if it is minimal on every edge e. Note that every frugal realization is also minimal.

3 Basic Observations and an Exponential Algorithm

We first provide upper and lower bounds for the number of labels per edge in realizations.

▶ **Lemma 2** (⋆). *Let D be an instance of URGR, and let e be a solid edge of G that is not part of a triangle.*

- *For* Any Strict *URGR, in each minimal realization of D, e receives at most two labels.*
- *For all other versions of URGR, in each minimal realization of D, e receives at most one label.*

For general edges, we show a linear upper bound with respect to the number of vertices.

▶ **Lemma 3** (⋆). *If a graph $D = (V, A)$ is realizable, then each minimal realization for D assigns at most $n = |V|$ labels per edge.*

Lemma 3 implies that all versions of URGR and DRGR under consideration are in NP. Next, we show that the bound of Lemma 3 is essentially tight for Any Strict URGR.

▶ **Theorem 4** (⋆). *For* Any Strict *URGR, there is an infinite family of directed graphs $\mathcal{B}$, such that for every $D = (V, A) \in \mathcal{B}$, where $G = (V, E)$ is the solid graph of D, (i) D is realizable and some edge $e \in E$ receives $\Omega(n)$ labels in every realization of D, where $n = |V|$, and (ii) G has a feedback vertex set of size 2 and a feedback edge set of size $\Theta(n)$.*

The proof of Theorem 4 is based on the construction of an instance with a solid graph that consists of a number of subgraphs, called pages, that all share a common edge, called the spine. The instance is such that all edges of each page (except the spine) must receive the same single label, and the labels increase from one page to the next. Furthermore, for every i the reachability matrix specifies that the top of the i-th page can reach the bottom of the $(i + 2)$-th page and of all later pages. This requirement can then only be realized by adding a label whose value lies between the labels of page i and $i + 2$ to the spine, thus forcing the spine to have a linear number of labels.

Based on the upper bounds on labels per edge/arc from Lemma 3, one can solve all problems under consideration via dynamic programming over subsets of already labeled edges/arcs. Roughly speaking, we design a dynamic program that stores entries for each subset A' of A and each time step i, on whether there is a temporal graph with lifetime i that has reachability graph $D' := (V, A')$. If Simple RGR or Happy RGR is considered, the table also needs to store the already labeled edges/arcs in the first i time steps, to ensure that none of these edges/arcs receive another label.

▶ **Theorem 5** (⋆). *Each version of URGR and DRGR under consideration can be solved in $2^{\mathcal{O}(|A|)} \cdot n^{\mathcal{O}(1)}$ time, where A denotes the arc set of the input graph.*

4 Solid Bridge Edges: Properties and Splitting Operations

In this section, we show several structural results as well as three splitting operations for graphs with bridge edges in the solid graph. These insights will be important for our FPT algorithm and will also simplify instances of URGR where the solid graph is a tree.

Consider a bridge edge $e = \{u, v\}$ whose deletion splits the solid graph G into connected components G_u and G_v, where G_u contains u and G_v contains v. A dashed arc (a, b) *spans* e if $a \in V(G_u)$ and $b \in V(G_v)$ or vice versa. Two arcs (a, b) and (c, d) span e in the same direction if they span e and either $a, c \in V(G_u)$ or $a, c \in V(G_v)$. If e has a single label in a realization, then it must be the case that the dashed arcs spanning e are "transitive" in the following sense: If dashed arcs (a, b) and (c, d) both span e in the same direction, then (a, d) and (c, b) must also be dashed arcs. This is because for any two temporal paths that pass through e in the same direction, the part of one path up to edge e can be combined with the part of the other path after e.

▶ **Definition 6.** *Consider a bridge edge $e = \{u, v\}$ whose deletion splits the solid graph G into connected components G_u and G_v. The edge e is a* special *bridge edge if there exist vertices a in G_u and b in G_v such that $D_{av} = 1$, $D_{ub} = 1$ and $D_{ab} = 0$ (or if the same condition holds with u and v exchanged). A bridge edge that is not special is called* non-special.

Intuitively, a special bridge edge is a bridge edge e such that there are dashed arcs spanning e in the same direction for which transitivity (as outlined above) is violated.

▶ **Lemma 7** (⋆). *In each frugal realization of D, every special bridge edge is assigned two labels and every non-special bridge edge is assigned a single label.*

▶ **Corollary 8.** *Let D be an instance of any version of URGR under consideration besides* Any Strict URGR. *If the solid graph of D contains a special bridge edge, then D is a no-instance.*

4.1 Splitting Into Subinstances

We now define splitting operations on bridges. First, consider non-special bridges.

▶ **Lemma 9** (Splitting at a non-special bridge). *Consider any variant of URGR. Let $e = \{u, v\}$ be a non-special bridge edge, and let G_u and G_v be the connected components of G resulting from the deletion of e. Then the instance D is realizable if and only if the subinstances induced by $V(G_u) \cup \{v\}$ and $V(G_v) \cup \{u\}$ are realizable.*

Proof. Let D_u and D_v denote the subinstances induced by $V(G_u) \cup \{v\}$ and $V(G_v) \cup \{u\}$, respectively. If D is realizable, then the realization λ induces realizations of the two subinstances D_u and D_v. If D_u and D_v are realizable, their realizations assign a single label to e, and these realizations can be chosen so that e receives the same label in both realizations. Hence, the union of the two realizations is a realization of D. ◀

By applying Lemma 9 repeatedly to a non-pendant non-special bridge edge until no such edge exists, we obtain subinstances in which all non-special bridge edges are pendant.

Now consider special bridges. Special bridge edges cannot occur in yes-instances of any variant of URGR except Any Strict URGR (see Corollary 8), so we only consider Any Strict URGR in the following. Let edge $e = \{u, v\}$ be a special bridge with G_u and G_v defined as above. If the instance is realizable, for every vertex in $V(G_u)$ there are at most three possibilities for which vertices in $V(G_v)$ it can reach, depending on whether it cannot reach v, can reach v at time α, or can reach v only at time β, where α and β with $\alpha < \beta$ are

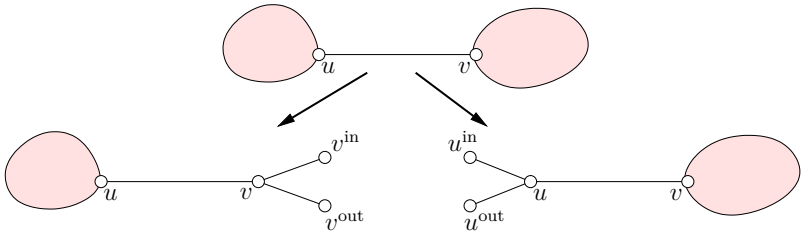


Figure 1 Illustration of the splitting operation at a special bridge $\{u, v\}$ (Lemma 11).

the labels assigned to e in the realization. The same holds with G_u and G_v exchanged. If this condition is violated, then the instance cannot be realized. The following definition captures this condition. For each vertex $w \in V(G_u)$, let $R_{u\to v}(w) = \{x \in V(G_v) \mid (w, x) \in A\}$, and define $R_{v\to u}(x)$ for $x \in V(G_v)$ analogously.

▶ **Definition 10.** *Let $e = \{u, v\}$ be a special bridge with G_u and G_v defined as above. We say that e is a special bridge edge with* plausible reachability *if the following conditions hold:*

- *If there exist vertices $a \in V(G_u)$ and $b \in V(G_v)$ such that $D_{av} = D_{ub} = 1$ and $D_{ab} = 0$, then every vertex $w \in V(G_u)$ satisfies $R_{u\to v}(w) \in \{\emptyset, R_{u\to v}(a), R_{u\to v}(u)\}$ with $\emptyset \subset R_{u\to v}(a) \subset R_{u\to v}(u)$. Otherwise, every vertex satisfies $R_{u\to v}(w) \in \{\emptyset, R_{u\to v}(u)\}$.*
- *The same condition holds with the roles of G_u and G_v exchanged.*

If there is a special bridge edge $\{u, v\}$ with plausible reachability, we can split the instance into two subinstances that can be solved separately, as illustrated in Figure 1. Each subinstance consists of the edge $\{u, v\}$ together with either G_u or G_v. For the subinstance containing $\{u, v\}$ and G_u, up to two new leaf vertices are attached to v, named v^{in} and v^{out} in the figure. Here, v^{in} represents the vertices in G_v that can reach u only by traversing $\{u, v\}$ using the larger of its two time labels, and v^{out} represents the vertices in G_v that can be reached from G_u only by traversing $\{u, v\}$ using the larger of its two time labels. The other subinstance is handled analogously.

▶ **Lemma 11** ((⋆), Splitting at a special bridge)**.** *Consider* Any Strict URGR*. Let $e = \{u, v\}$ be a special bridge edge with plausible reachability, and let G_u and G_v be the connected components of G resulting from the deletion of e. Then the instance D is realizable if and only if the two subinstances constructed as follows are realizable:*

- *To construct subinstance D_u, take the subgraph D' of D induced by $V(G_u) \cup \{v\}$ and attach one or two leaves to v as follows:*
  - *If there are vertices $a \in V(G_u)$ and $b \in V(G_v)$ with $D_{av} = D_{ub} = 1$ and $D_{ab} = 0$, then attach a leaf (called* out-leaf*) z to v with $D'_{zv} = 1$ and $D'_{az} = 1$ for all $a \in \{v\} \cup \{w \in V(G_u) \mid R_{u\to v}(w) = R_{u\to v}(u)\}$, and all other entries of D' involving z equal to 0.*
  - *If there are vertices $c \in V(G_v)$ and $d \in V(G_u)$ with $D_{cu} = D_{vd} = 1$ and $D_{cd} = 0$, then attach a leaf (called* in-leaf*) z' to v with $D'_{vz'} = 1$ and $D'_{z'a} = 1$ for all $a \in \{v\} \cup R_{v\to u}(c)$ and $D'_{z'z} = 1$ if an out-leaf z has been added, and all other entries of D' involving z' equal to 0.*

  Note that at least one of the two conditions above must be satisfied because e is a special bridge edge. The resulting instance D' is the desired subinstance D_u.
- *Subinstance D_v is constructed analogously, with the roles of u and v exchanged.*

Note that we cannot exhaustively apply the operation behind Lemma 11, since each application on a special bridge $\{u, v\}$ results into two new instances in which edge $\{u, v\}$ is again a special bridge. However, by recursively applying this operation on special bridges on

which the operation has not yet been applied, we eventually obtain instances, where each special bridge has at least one endpoint that has at most two other neighbors and both of them have a degree of 1.

Finally, we consider the case of two pendant edges that have a common neighbor and must receive the same single label in every realization. We argue that we can remove one of them from the instance provided their reachability requirements from/to the rest of the graph are the same. This is because a realization of the instance with one of the two pendant edges removed gives a realization of the original instance simply by labeling the pendant edge that was removed with the same label as the other pendant edge.

▶ **Lemma 12** ((⋆), Removal of redundant pendant vertices)**.** *Let D be an instance of URGR and let v and w be two degree-1 vertices with the same common neighbor u in G that satisfy $D_{vw} = D_{wv} = 0$. Then,*

- *for all variants of Non-strict URGR (including Proper URGR and Happy URGR), D is not realizable, and*
- *for Any Strict URGR and Simple Strict URGR, D is realizable if and only if (i) the subinstance D' resulting from D by deleting v is realizable and (ii) for each $x \in V \setminus \{v, w\}$, $D_{vx} = D_{wx}$ and $D_{xv} = D_{xw}$.*

4.2 Algorithms for Instances with a Tree as Solid Graph

Since all edges of trees are bridges, we now describe, based on our insights about labels in frugal realizations, an algorithm for instances of URGR where the solid graph is a tree.

▶ **Theorem 13.** *There is a polynomial-time algorithm for solving instances of Any Strict URGR for which the solid edges form a tree.*

Proof Sketch. Let $T = (V, E)$ denote the tree of solid edges. We first check that D satisfies several conditions that must hold if D is realizable. For example, if (u, v) is an arc, then (u, w) and (w, v) must also be arcs for every internal node w of the unique u-v-path in T. If one of these conditions is violated, D is not realizable. Otherwise, we construct a linear program (LP) that has two non-negative variables ℓ_e and h_e for each $e \in E$ that represent the lower and higher label of e, respectively, in a realization. Constraints $\ell_e = h_e$ for non-special edges and $\ell_e + 1 \leq h_e$ for special edges ensure that each edge receives the correct number of labels. For edges uv and vw such that $(u, w) \in A$, the constraints $\ell_{uv} + 1 \leq h_{vw}$ and $h_{uv} \leq \ell_{vw}$ ensure that there is a temporal u-w-path but no temporal w-u-path. Call a non-arc (u, v) *minimal* if it is the only arc missing in the direction from u to v on the path from u to v. For every minimal non-arc (u, v), we add constraints $h_{e_i} = \ell_{e_{i+1}}$ for every pair of consecutive edges e_i and e_{i+1} on the u-v path (which we show to consist of special edges except for the first and last edge, which are both non-special). For arcs (u, v) such that the first and last edge of the u-v-path are non-special and all other edges are special, we add the constraint $\sum_i (\ell_{e_{i+1}} - h_{e_i}) \geq 1$, where the sum is over all pairs of consecutive edges e_i, e_{i+1} of the u-v-path. We can show that the LP is feasible if and only if D is realizable. Furthermore, any feasible solution of the LP corresponds to a frugal realization with fractional labels, which can be made integral in a straightforward post-processing step. ◀

By testing additional conditions (such as the absence of special edges) before constructing the linear program, we can also use the approach to solve all other variants of URGR.

5 Hardness for Undirected Reachability Graph Realizability

In this section, we show that all considered versions of Undirected Reachability Graph Realizability are NP-hard, even on instances with a constant maximum degree. We obtain these hardness results by reductions from SAT, where we have a source s_i and a terminal vertex t_i for each clause c_i, such that D contains the dashed arc (s_i, t_i). The only way to realize such an arc is to follow a dense path that crosses through some variable gadget. We design a variable gadget for each variable x, for which the only way to realize it is to ensure that (i) only clauses that contain x positively can reach their terminal via this variable gadget, or (ii) only clauses that contain x negatively can reach their terminal over this variable gadget.

▶ **Theorem 14** (⋆). *Each version of URGR under consideration is NP-hard on directed graphs of constant maximum degree. Moreover, no version of URGR under consideration can be solved in $2^{o(|V|+|A|)} \cdot n^{\mathcal{O}(1)}$ time, unless the ETH fails.*

Hence, the running time of Theorem 5 can presumably not be improved significantly.

We now strengthen our hardness result for Any Strict URGR and Simple Strict URGR, which will highly motivate the analysis of parameterized algorithms for the parameter "feedback edge set number" of the solid graph, which we consider in Section 6.

▶ **Theorem 15** (⋆). *Any Strict URGR and Simple Strict URGR are W[2]-hard when parameterized by the feedback vertex set number and treedepth of the solid graph.*

Proof. We reduce from Set Cover which is W[2]-hard when parameterized by k [10].

Let $I := (U, \mathcal{F}, k)$ be an instance of Set Cover with $\mathcal{F} = \{F_1, \dots, F_r\}$. Assume without loss of generality that each element of U is contained in at least one hyperedge and that $\mathcal{F}$ has size at least k, as otherwise I could be solved trivially. We obtain a directed graph $D = (V, A)$ with solid graph $G = (V, E)$ as follows: The graphs contain the vertices $\{\top, a, a', b, b', \bot\}$, each element $u \in U$ as a vertex, and for each $i \in [1, k]$ the vertex c_i. Additionally, for each $F \in \mathcal{F}$ and each $u \in F$, G contain the vertices w_u^F, v_u^F, and q_u^F.

Next, we describe the solid edges of G, that is, the bidirectional arcs of D. See Figure 2 for the solid edges and dashed arcs of the main connection gadget of the instance.

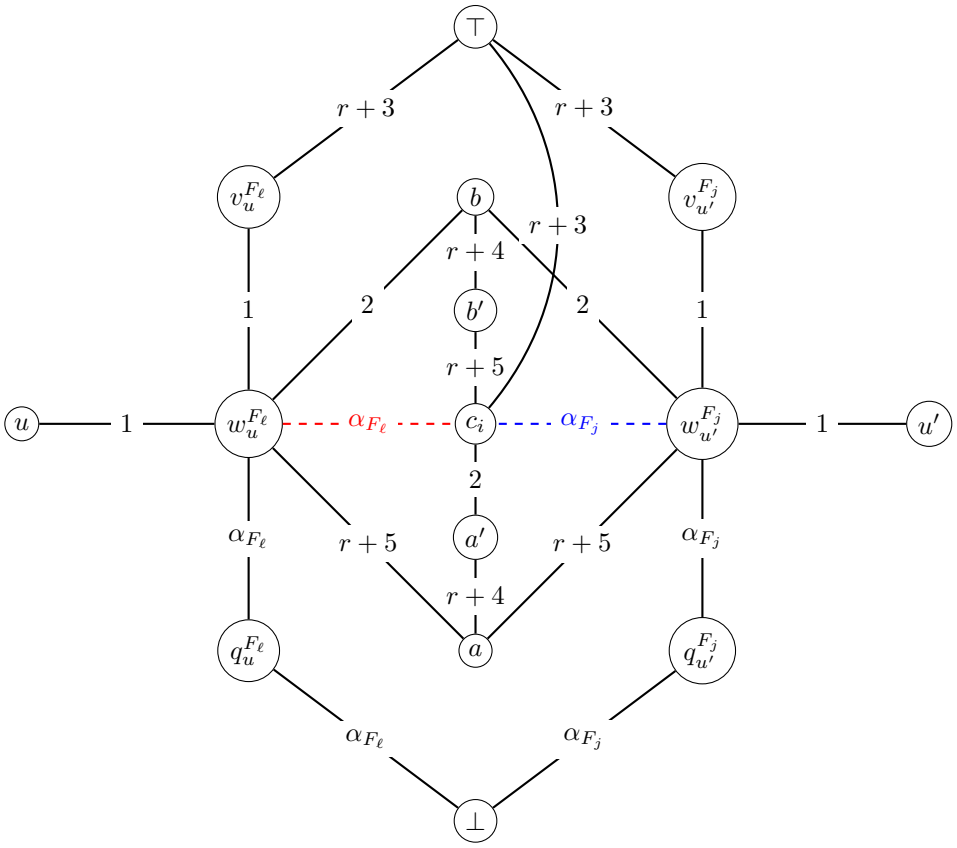
The graph G contains the edges $\{a, a'\}$, $\{b, b'\}$, and for each $i \in [1, k]$, the edges $\{a', c_i\}$, $\{c_i, b'\}$, and $\{c_i, \top\}$. Moreover, for each $F \in \mathcal{F}$ and each $u \in F$, G contains the edges $\{u, w_u^F\}$, $\{w_u^F, a\}$, $\{w_u^F, b\}$, $\{\{w_u^F, c_i\} \mid i \in [1, k]\}$, $\{w_u^F, v_u^F\}$, $\{w_u^F, q_u^F\}$, $\{v_u^F, \bot\}$, and $\{q_u^F, \top\}$.

Finally, we describe the dashed arcs: D contains the dashed arcs (a', b), (b, a), $(a', \top)$, $(b, \top)$, and $(\top, b')$. For each $i \in [1, k]$, D also contains the dashed arcs (b, c_i) and (c_i, a). Let $u \in U$. Then, D contains the arc $(u, \top)$. Moreover, D contains the arcs (u, a), (u, b), (u, b'), and (u, c_i) for each $i \in [1, k]$. Additionally, for each hyperedge $F \in \mathcal{F}$ with $u \in F$, D contains the arcs (u, v_u^F), (v_u^F, a), (b, v_u^F), $(w_u^F, \top)$, (w_u^F, b'), (a', w_u^F), (q_u^F, v_u^F), (q_u^F, a), (q_u^F, b), (q_u^F, b'), and (q_u^F, c_i) for each $i \in [1, k]$. The only other dashed arcs in D are between v_u^F-vertices. Let $F_x, F_y \in \mathcal{F}$ with $x < y$ and let $u_x \in F_x$ and $u_y \in F_y$. Then, D contains the arc $(v_{u_x}^{F_x}, v_{u_y}^{F_y})$.

This completes the construction of D. First, we show the parameter bounds.

▷ Claim 16 (⋆). G has a feedback vertex set of size $k + 6$ and treedepth $\mathcal{O}(k)$.

We show that I is a yes-instance of Set Cover if and only if D is realizable via strict temporal paths. More precisely, we show that if I is a yes-instance of Set Cover, then there is a simple undirected temporal graph with strict reachability graph D. This then implies the correctness of the reduction for both Any Strict URGR and Simple Strict URGR.



	u	$\top$	v_u^F	w_u^F	q_u^F	a	a'	b	b'	$c_i, i \in [1,k]$	$\bot$
u		1	1	1	0	1	0	1	1	1	0
$\top$	0		0	0	1	0	0	0	1	1	0
v_u^F	0	0		1	0	1	0	0	0	0	1
w_u^F	1	1	1		1	1	0	1	1	1	0
q_u^F	0	1	1	1		1	0	1	1	1	0
a	0	0	0	1	0		1	0	0	0	0
a'	0	1	0	1	0	1		0	1	1	0
b	0	1	1	1	0	1	0		1	1	0
b'	0	0	0	0	0	0	0	1		1	0
$c_i, i \in [1,k]$	0	1	0	1	0	1	1	0	1		0
$\bot$	0	0	1	0	0	0	0	0	0	0	

Figure 2 Illustration of a part of the adjacency matrix of the W[2]-hardness reduction. The highlighted cells of the matrix indicate the dashed arcs. As we show, the labels α_{F_ℓ} and α_{F_j} need to be distinct if the hyperedges F_ℓ and F_j are distinct. Thus, for each $i \in [1,k]$, labeling both edges $\{w_u^{F_\ell}, c_i\}$ and $\{w_{u'}^{F_j}, c_i\}$ is not possible, as otherwise, there would be a temporal path from $w_u^{F_\ell}$ to $w_{u'}^{F_j}$ (or vice versa), which is not allowed.

We defer this proof to the full version and only provide an informal idea for the correctness. Intuitively, in each realization for D, for each $i \in [1,k]$, there can be at most one hyperedge $F \in \mathcal{F}$ for which edges between c_i and vertices of $W_F := \{w_u^F \mid u \in F\}$ can receive labels. This can be seen as follows: For each $w_{u'}^{F'} \in V$, (i) there is no dashed arc between $w_{u'}^{F'}$ and $\bot$ and (ii) there is no dashed arc between $v_{u'}^{F'}$ and c_i. Hence, Observation 1 implies that, if $\{w_{u'}^{F'}, c_i\}$ receives at least one label, then there is some $\alpha_{F'} \in \mathbb{N}$, such that the edges $\{w_{u'}^{F'}, c_i\}$ and $\{v_{u'}^{F'}, \bot\}$ receive the label set $\{\alpha_{F'}\}$ under λ. Based on the dashed arcs between the $v_{u'}^{F'}$-vertices, the label $\alpha_{F'}$ and the label $\alpha_{F''}$ are distinct for distinct hyperedges F' and F''. This then implies that there can be at most one hyperedge $F \in \mathcal{F}$ for which edges between c_i and vertices of W_F can receive labels, as otherwise, a strict temporal path between distinct $w_{u'}^{F'}$-vertices would be realized.

The edges with at least one label between c_i and w_u^F-vertices thus resemble a selection of at most one hyperedge of $\mathcal{F}$ for each $i \in [1,k]$. Since the only dense paths from a vertex $u \in U$ to $\top$ are of the form $(u, w_u^F, c_i, \top)$ for $i \in [1,k]$ and $F \in \mathcal{F}$ with $u \in F$, this selection of k hyperedges encodes a set cover, as the arcs $(u, \top)$ are realized over such dense paths. ◀

6 Parameterizing by the Feedback Edge Set Number

In this section we generalize our algorithm on tree instances of URGR to tree-like graphs. Recall that a feedback edge set of a graph G is a set F of edges of G, such that $G - F$ is acyclic. We denote by fes the feedback edge set number of the solid graph G of D, that is, the size of the smallest feedback edge set of G.

▶ **Theorem 17.** *Each version of URGR can be solved in* $\text{fes}^{\mathcal{O}(\text{fes}^2)} \cdot n^{\mathcal{O}(1)}$ *time.*

Recall that the parameter can presumably not be replaced by a smaller parameter like feedback vertex set number or treedepth (see Theorem 15). In the remainder of this section, we present the main idea for the proof of Theorem 17. We will only describe the algorithm for the most general version Any Strict URGR, but for all more restrictive versions, all arguments work analogously.

Definitions and Notations. Let $G = (V, E)$ be the solid graph and let F be a minimum size feedback edge set of G. We assume without loss of generality that G is connected, as otherwise, we can solve each connected component independently, or detect in polynomial time that D is not realizable if there are dashed arcs between different connected components of the solid graph. Moreover, let X denote the endpoints of the edges of F. Note that $|X| \leq 2 \cdot |F|$. We define the set V^* as the (unique) largest subset S of vertices of V that each have at least two neighbors in $G[S]$, that is, V^* is the 2-core [24] of G. This set can be computed in polynomial time. Moreover, we define $V' := V \setminus V^*$. Note that $X \subseteq V^*$, since F is a minimum-size feedback edge set. Since G is connected, for each vertex $v' \in V'$, there is a unique vertex $v \in V^*$ which has closest distance to v' among all vertices of V^*. For a vertex $v \in V^*$, denote by V_v the set of all vertices of V' for which v is the closest neighbor in V^*. Note that V_v might be empty. Since v' is in no cycle, removing v from G would result in v' ending up in a component that has no vertex of V^*. Hence, $T_v := G[\{v\} \cup V_v]$ is a tree for which we call v the *root*. Moreover, we call T_v the *pendant tree* of v.

Recall that each vertex of V^* has degree at least 2 in $G[V^*]$. Furthermore, $G - F$ is acyclic and thus $G[V^*] - F$ is a tree. Let Y_3 denote the set of all vertices of degree at least 3 in $G[V^*] - F$, and let Y_2 be the neighbors of $X \cup Y_3$ in $V^* \setminus (X \cup Y_3)$. Observe that each leaf of $G[V^*] - F$ is a vertex of X. Moreover, in each tree with ℓ leaves, there

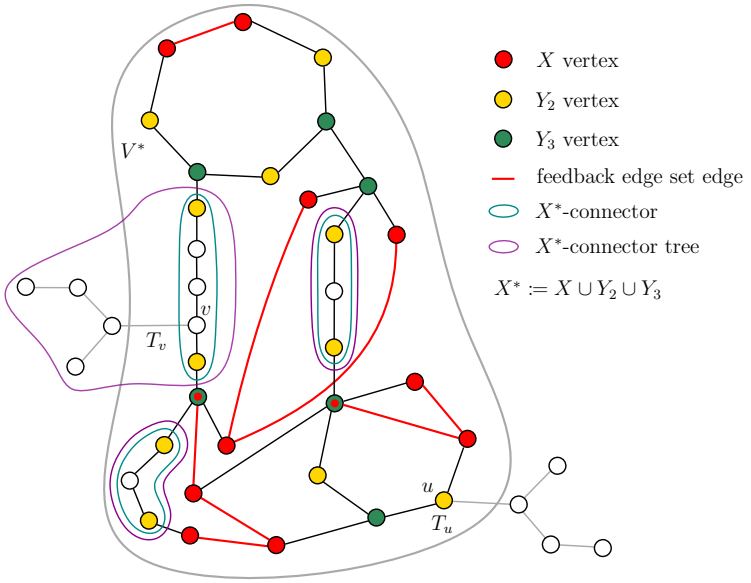


Figure 3 An illustration of the main definitions used by the FPT algorithm for parameter fes. The grey area contains the vertices of V^*, that is, the 2-core of G.

are $\mathcal{O}(\ell)$ vertices having (i) degree at least 3 or (ii) a neighbor of degree at least 3. This implies that $|Y_2 \cup Y_3| \in \mathcal{O}(|X|)$. Let $X^* := X \cup Y_2 \cup Y_3$. See Figure 3 for an illustration of these sets of vertices and some of the main definitions used in this section. By the above, $|X^*| \in \mathcal{O}(|X|) \subseteq \mathcal{O}(\text{fes})$. Recall that each vertex of Y_2 has degree exactly 2 in $G[V^*]$. Moreover, note that each vertex of $V^* \setminus X^*$ is part of a unique path P where each internal vertex of P is from $V^* \setminus X^*$ and the endpoints of P are from Y_2. For each such path, each internal vertex has degree exactly 2 in $G[V^*]$ and there are at most $|X^*| - 1$ such paths, since $G[V^*] - F$ is acyclic. Note that X^* contains all vertices that are part of triangles in $G[V^*]$ and thus in G. This implies that in each minimal realization of D, each edge incident with at least one vertex of $V \setminus X^*$ receives at most two labels (see Lemma 2). We now define the above mentioned paths between the vertices of Y_2 is a more general way.

▶ **Definition 18.** *Let W with $X^* \subseteq W \subseteq V^*$ and let P be a path of length at least 2 in $G[V^*]$ with endpoints a and b in W and all internal vertices from $V^* \setminus W$. We call P a* W-connector*. Moreover, let $C := G[V(P) \cup \bigcup_{q \in V(P) \setminus \{a,b\}} V_q]$. We call C a* W-connector tree *and the* extension *of P.*

Note that the endpoints of each X^*-connector are from Y_2 and thus have degree 2 in $G[V^*]$. Moreover, each internal vertex v of a W-connector P has degree exactly 2 in $G[V^*]$, that is, the only two neighbors of v in $G[V^*]$ are the predecessor and the successor of v in P. This also implies that each endpoint of a W-connector has degree exactly 2 in $G[V^*]$. Since each edge of P is incident with at least one vertex of $V \setminus W \subseteq V \setminus X^*$, in each minimal realization of D, each edge of each W-connector receives at most two labels (see Lemma 2). The latter also holds for each W-connector tree.

▶ **Observation 19.** *Let W with $X^* \subseteq W \subseteq V^*$. Then, there are $\mathcal{O}(|W| + \text{fes})$ edges between vertices of W in G and there are $\mathcal{O}(|W| + \text{fes})$ W-connectors. Moreover, for each vertex $q \in V \setminus (W \cup \bigcup_{w \in W} V_w)$, there is exactly one W-connector tree that contains q.*

Note that this is a general property of graphs with a feedback edge set of size fes.

Abstract Description of the Algorithm. Our algorithm uses two preprocessing steps.

In the first step, we will present a polynomial-time reduction rule based on the splitting operations presented in Section 4. With this reduction rule, we will be able to remove vertices from large pendant trees. After exhaustive application, for each vertex $x \in V^*$, the pendant tree T_x will have $\mathcal{O}(d_x)$ vertices, where d_x denotes the degree of x in $G[V^*]$.

In the second step, we extend the set X^* to a set $W^* \subseteq V^*$ such that each W^*-connector has some useful properties (we will define connectors with these properties as *nice connectors*). Intuitively, a connector P with extension C is nice if (i) in a realization for D, the arcs in $D[V(C)]$ can only be realized by temporal paths that are contained in the connector tree C, and (ii) for each arc between a vertex outside of C and a vertex inside of C, we can in polynomial time detect over which (unique) edge of the connector incident with one of its endpoints this arc is realized. As we will show, we can compute such a set W^* of size $\mathcal{O}(\text{fes})$ in polynomial time, or correctly detect that the input graph is not realizable. By the first part, we additionally get that the total number of vertices in pendant trees that have their root in W^* is $\mathcal{O}(|W^*| + \text{fes}) = \mathcal{O}(\text{fes})$.

The algorithm then works as follows: We iterate over all possible partial labelings λ on the $\mathcal{O}(\text{fes})$ edges incident with vertices of W^* or vertices of pendant trees that have their root in W^*. As we will show, we can assume that each of these edges will only receive $\mathcal{O}(\text{fes})$ labels in each minimal realization. For each such labeling λ, we need to check whether we can extend the labeling to the so far unlabeled edges, that is, the edges that are part of any W^*-connector tree. As we will show, we can compute for each such connector P with extension C in polynomial time a set L_P of $\mathcal{O}(1)$ labelings for the edges of C, such that if λ can be extended to a realization for D, then we can extend λ (independently from the other connectors) by one of the labelings in L_P. Since W^* has size $\mathcal{O}(\text{fes})$, there are only $\mathcal{O}(\text{fes})$ many W^*-connectors (see Observation 19) and for each such connector P, the set L_P of labelings has constant size. Our algorithm thus iterates over all possible $\mathcal{O}(1)^{\mathcal{O}(\text{fes})} = 2^{\mathcal{O}(\text{fes})}$ labeling combinations for the connectors and checks whether one of these combinations extends λ to a realization for D. Summarizing, we will show the following.

▶ **Proposition 20.** *In polynomial time, we can detect that D is not realizable, or compute a set W^* with $X^* \subseteq W^* \subseteq V^*$, such that the set of edges E^* that are (i) incident with at least one vertex of W^* or (ii) part of some pendant tree with root in W^*, has size $\mathcal{O}(\text{fes})$ and where for each labeling λ of the edges of E^*, we can in $2^{\mathcal{O}(\text{fes})} \cdot n^{\mathcal{O}(1)}$ time (a) compute a labeling that realizes D, or (b) detect that there no frugal realization for D that agrees with λ on the labels of all edges of E^*.*

The running time of this algorithm is then $\text{fes}^{\mathcal{O}(\text{fes}^2)} \cdot n^{\mathcal{O}(1)}$, since (i) all preprocessing steps run in polynomial time, (ii) we only have to consider $\text{fes}^{\mathcal{O}(\text{fes}^2)}$ labelings λ (since we prelabel $\mathcal{O}(\text{fes})$ edges with $\mathcal{O}(\text{fes})$ labels each), and (iii) for each labeling λ, we only have to check for $2^{\mathcal{O}(\text{fes})}$ possible ways to extend λ to a realization for D. To see the second part, we show that it is sufficient to assign only labels of $\{i \cdot 2 \cdot n \mid i \in \mathcal{O}(\text{fes}^2)\}$ in λ.

6.1 Technical Highlights and Difficulties of the Algorithm

We now describe in more detail the intuition about the two preprocessing steps, as well as the algorithm to compute a constant number of labelings for each nice connector.

Dealing with pendant trees. Recall that d_x denotes the degree of a vertex $x \in V^*$ in $G[V^*]$. To reduce each pendant tree T_x with $x \in V^*$ to a size of $\mathcal{O}(d_x)$, we first apply the three splitting operations of Section 4 to each edge of T_x incident with x. Since each of these edges

is a bridge and part of a tree, one of the two resulting instances produced by the splitting operations is then a tree and can be solved in polynomial time. We thus stick with the one resulting instance which is not a tree. As we show, by applying the splitting operations, we get that (i) for each special bridge $\{b, x\}$ of T_x incident with x, b has at most two neighbors besides x (namely, the possible in-leaf and the possible out-leaf) and (ii) for each non-special bridge $\{b, x\}$ of T_x incident with x, b is a leaf. This directly implies that T_x has depth 2 and the number of leaves of T_x is $\mathcal{O}(|B_x|)$, where B_x denotes the neighbors of x in T_x.

Thus, to reduce the size of T_x it suffices to reduce the number of vertices in B_x to $\mathcal{O}(d_x)$. To do so, we show that for each external edge e (that is, an edge that is incident with x but not contained in T_x), we can detect in polynomial time a set of at most 5 internal edges (that is, edges between x and vertices of B_x) that "surround" or "block" the edge e. Intuitively, in each realization for D, no other internal edge can share a label with e. This defines a total of $\mathcal{O}(d_x)$ internal edges that are surrounding or blocking. Let S_x denote these edges. Based on this set of edges, we then show that we can efficiently (i) detect that D is not realizable, or (ii) find a vertex b of B_x that (together with its possible leaf-neighbors) can safely be removed from D, if B_x has size $\omega(|S_x|)$. Hence, after exhaustive application of this operation, only $\mathcal{O}(|S_x|) \subseteq \mathcal{O}(d_x)$ vertices of B_x remain. Since each of these vertices only has at most two neighbors besides x, and these neighbors are leaves, the total size of T_x is then $\mathcal{O}(d_x)$.

Ensuring connectors with nice properties. We now give a more formal idea behind the definition of nice connectors. Let P be a connector with endpoints a and b, and let C be the extension of P. Then, P is nice, if (i) there is no dense path between a and b outside of $D[V(C)]$ and (ii) for each arc $(u, v) \in D$ between a vertex of C and a vertex outside of C, there is no dense (u, v)-path that goes over a or there is no dense (u, v)-path that goes over b.

Since the definition of nice connectors relies on the non-existence of dense paths between specific vertex pairs, detecting whether a connector is nice can presumably not be done in polynomial time.[2] To still achieve the goal of finding a desired set W^* where each connector is nice, we show the following: Whenever we have a set $W \supseteq X^*$ and a W-connector P, then we can in polynomial time (i) detect that D is not realizable or (ii) compute a constant number of vertices U_P of P, such that each subpath of P that is a $(W \cup U_P)$-connector is nice. That is, even though we cannot check efficiently whether the nice property is fulfilled for a given connector, we can ensure it by adding few vertices to our set. To compute the set W^*, we start with $W^* = X^*$ and iteratively add for each X^*-connector P the set U_P to W^*. Since $|X^*| \in \mathcal{O}(\text{fes})$ and there are $\mathcal{O}(\text{fes})$ X^*-connectors, the resulting set W^* also has size $\mathcal{O}(\text{fes})$. As we show, the update of W^* preserves the nice property of the connector before and after the update. In this way, we ensure that each W^*-connector is nice.

Dealing with nice connectors. Recall the definition of nice connectors. The first property essentially ensures that arcs in $D[V(C)]$ can only be realized via dense paths in $D[V(C)]$, which makes the local realization of $D[V(C)]$ in some sense independent from the remainder of D. The second property ensures that we only have one choice (with respect to dense paths inside of $D[V(C)]$) to realize an arc (u, v) between an internal and an external vertex.

We make use of these properties, since a nice connector P and its extension C only interact with the remainder of D via exactly two edges e_a and e_b. Moreover, both these edges receive at most two labels in every frugal realization. Assuming we are given the

[2] This is due to the fact that finding a dense path between two specific vertices can be shown to be NP-hard by reducing from Multicolored Clique.

labeling for e_a and e_b, we show that we can compute a constant number of labelings L_P for the edges of C, such that if there is a realization λ for D agreeing with the prelabeling of e_a and e_b, then we can replace the labels of λ on the edges of C by the labels of some labeling in L_P. The essential idea behind this algorithm is to build a constant number of tree instances in which we simulate how the connector interacts with the remainder of D in any possible labeling. As we show, we only have to (i) consider the cases for each of the edges e_a and e_b, whether there are temporal paths in λ that enter or exit C via e_a (e_b) via the smallest or largest label of e_a (e_b), and (ii) consider the cases whether some temporal paths under λ enter C via e_a and leave via e_b and whether they traverse the respective edges at their smallest or largest label. These are in total only a constant number of options. For each such option, we construct a tree instance of the problem obtained from $D[V(C)]$ by adding a constant number of vertices each to simulate the specific option. In some sense, these instances are generalizations of the instances built in the splitting rule for special bridges. Afterwards, we check for which of these instances we can find a labeling that agrees with the prelabeling on e_a and e_b and add such a labeling to L_P if it exists. The latter task can be performed in polynomial time, by using our linear program for tree instances.

7 The Complexity of Directed Reachability Graph Realizability

Finally, we consider the complexity of versions of Directed Reachability Graph Realizability. As already discussed by Döring [11], each directed graph D can be the strict reachability graph of a simple directed temporal graph. Hence, Any Strict DRGR and Simple Strict DRGR are always yes-instances and can thus be solved in polynomial time.

We now show that all DAGs and all transitive graphs are trivial yes-instances for each version of the problem.

▶ **Lemma 21** (⋆). *An instance $D = (V, A)$ of DRGR is a yes-instance, if (i) we consider Any Strict DRGR or Simple Strict DRGR, or (ii) D is a DAG or a transitive graph.*

Here, a directed graph $D = (V, A)$ is *transitive*, if for every distinct vertices $u, v, w \in V$ with $(u, v) \in A$ and $(v, w) \in A$, the arc (u, w) is also in A.

All other versions of DRGR are NP-hard even on graphs that are close to DAGs. More precisely, on graphs where a DAG can be obtained by removing a constant number of arcs.

▶ **Theorem 22** (⋆). *Proper DRGR, Happy DRGR, and all considered versions of Non-strict DRGR are NP-hard on graphs with a constant size feedback arc set. Moreover, none of these versions of DRGR can be solved in $2^{o(|V|+|A|)} \cdot n^{\mathcal{O}(1)}$ time, unless the ETH fails.*

8 Conclusion

We studied Reachability Graph Realizability and gave for both directed and undirected temporal graphs the complete picture for the classical complexity of all settings, answering this open problem posed by Casteigts et al. [3] and Döring [11]. For URGR, we additionally showed that the problem can be solved in FPT-time for the feedback edge set number fes of the solid graph. As we showed, this parameter cannot be replaced by smaller parameters like feedback vertex set number or treedepth of the solid graph, unless FPT = W[2].

There are several directions for future work: First, it would be interesting to see whether (some) versions of URGR admit a polynomial kernel for fes. Another interesting task is to determine whether Proper URGR, Happy URGR, or Non-strict URGR admits an FPT algorithm for feedback vertex set or treedepth. This is not excluded by our W[2]-hardness result, since that reduction only worked for Any Strict URGR and Simple Strict URGR.

Finally, one could analyze the parameterized complexity of Directed Reachability Graph Realizability with respect to directed graph parameters.

References

1 Eleni C. Akrida, Leszek Gąsieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61:907–944, 2017. doi:10.1007/S00224-017-9757-X.

2 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, 14(2):267–285, 2003. doi:10.1142/S0129054103001728.

3 Arnaud Casteigts, Timothée Corsini, and Writika Sarkar. Simple, strict, proper, happy: A study of reachability in temporal graphs. *Theor. Comput. Sci.*, 991:114434, 2024. doi:10.1016/J.TCS.2024.114434.

4 Arnaud Casteigts, Michelle Döring, and Nils Morawietz. Realization of Temporally Connected Graphs Based on Degree Sequences. In *36th International Symposium on Algorithms and Computation*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. To appear. Full version: https://doi.org/10.48550/arXiv.2504.17743.

5 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012. doi:10.1080/17445760.2012.668546.

6 Arnaud Casteigts, Nils Morawietz, and Petra Wolf. Distance to transitivity: New parameters for taming reachability in temporal graphs. In *49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024)*, volume 306 of *LIPIcs*, pages 36:1–36:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.MFCS.2024.36.

7 Esteban Christiann, Eric Sanlaville, and Jason Schoeters. On inefficiently connecting temporal networks. In *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SAND.2024.8.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

9 Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. *Information and Computation*, 285:104890, 2022. doi:10.1016/J.IC.2022.104890.

10 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

11 Michelle Döring. Simple, Strict, Proper, and Directed: Comparing Reachability in Directed and Undirected Temporal Graphs. In *36th International Symposium on Algorithms and Computation*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. To appear. Full version: https://doi.org/10.48550/arXiv.2501.11697.

12 Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021. doi:10.1016/J.JCSS.2021.01.007.

13 Thomas Erlebach, Nils Morawietz, and Petra Wolf. Parameterized algorithms for multi-label periodic temporal graph realization. In *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *LIPIcs*, pages 12:1–12:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SAND.2024.12.

14 F. Göbel, J. Orestes Cerdeira, and Henk Jan Veldman. Label-connected graphs and the gossip problem. *Discret. Math.*, 87(1):29–40, 1991. doi:10.1016/0012-365X(91)90068-D.

15 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 504–513, 2000. doi:10.1145/335305.335364.

16 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The complexity of computing optimum labelings for temporal connectivity. *Journal of Computer and System Sciences*, 146:103564, 2024. doi:10.1016/J.JCSS.2024.103564.

17 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Temporal graph realization from fastest paths. In *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SAND.2024.16.

18 George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019. doi:10.1007/S00453-018-0478-6.

19 George B. Mertzios, Hendrik Molter, Nils Morawietz, and Paul G. Spirakis. Realizing temporal transportation trees. In *Proceedings of the 51st Workshop on Graph-Theoretic Concepts in Computer Science (WG 2025)*, LNCS. Springer, 2025. To appear. Full version: arXiv:2403.18513.

20 George B. Mertzios, Hendrik Molter, Nils Morawietz, and Paul G. Spirakis. Temporal Graph Realization with Bounded Stretch. In Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025)*, volume 345 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 75:1–75:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. Full version: arXiv:2504.14258. doi:10.4230/LIPIcs.MFCS.2025.75.

21 Julia Meusel, Matthias Müller-Hannemann, and Klaus Reinhardt. Directed temporal tree realization for periodic public transport: Easy and hard cases. In *36th International Symposium on Algorithms and Computation*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. To appear. Full version: https://doi.org/10.48550/arXiv.2504.07920.

22 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.

23 Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72–72, 2018.

24 Stephen B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983. doi:10.1016/0378-8733(83)90028-X.

25 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020. doi:10.1016/J.JCSS.2019.07.006.

Faster Algorithm for Bounded Tree Edit Distance in the Low-Distance Regime

Tomasz Kociumaka
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Ali Shahali
Sharif University of Technology, Tehran, Iran

Abstract

The *tree edit distance* is a natural dissimilarity measure between rooted ordered trees whose nodes are labeled over an alphabet Σ. It is defined as the minimum number of node edits – insertions, deletions, and relabelings – required to transform one tree into the other. The weighted variant assigns costs ≥ 1 to edits (based on node labels), minimizing total cost rather than edit count.

The unweighted tree edit distance between two trees of total size n can be computed in $\mathcal{O}(n^{2.6857})$ time; in contrast, determining the weighted tree edit distance is fine-grained equivalent to the All-Pairs Shortest Paths (APSP) problem and requires $n^3/2^{\Omega(\sqrt{\log n})}$ time [Nogler, Polak, Saha, Vassilevska Williams, Xu, Ye; STOC'25]. These impractical super-quadratic times for large, similar trees motivate the bounded version, parameterizing runtime by the distance k to enable faster algorithms for $k \ll n$.

Prior algorithms for bounded unweighted edit distance achieve $\mathcal{O}(nk^2 \log n)$ [Akmal & Jin; ICALP'21] and $\mathcal{O}(n + k^7 \log k)$ [Das, Gilbert, Hajiaghayi, Kociumaka, Saha; STOC'23]. For weighted, only $\mathcal{O}(n + k^{15})$ is known [Das, Gilbert, Hajiaghayi, Kociumaka, Saha; STOC'23].

We present an $\mathcal{O}(n + k^6 \log k)$-time algorithm for bounded tree edit distance in both weighted/unweighted settings. First, we devise a simpler weighted $\mathcal{O}(nk^2 \log n)$-time algorithm. Next, we exploit periodic structures in input trees via an optimized universal kernel: modifying prior $\mathcal{O}(n)$-time $\mathcal{O}(k^5)$-size kernels to generate such structured instances, enabling efficient analysis.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases tree edit distance, edit distance, kernelization, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.94

Related Version *Full Version*: `https://arxiv.org/abs/2507.02701` [27]

Funding *Ali Shahali*: The work of was carried out mostly during a summer internship at the Max Planck Institute for Informatics.

1 Introduction

The edit distance between two strings – the minimum cost of insertions, deletions, and substitutions needed to transform one string into the other – is one of the most widely used string similarity measures with numerous algorithmic applications. It provides a robust model for comparing sequential data and underpins techniques in fields such as computational biology, natural language processing, and text correction. Nevertheless, many data types exhibit hierarchical rather than linear structure. For example, RNA secondary structures, syntactic parse trees of natural language sentences, and hierarchical representations of documents and code all store information in tree-like forms. One can linearize such data and still use string edit distance, but the resulting alignments disregard the original hierarchical structure, and thus more expressive similarity measures are needed in most scenarios.

First introduced by Selkow [38], tree edit distance generalizes the notion of string edit distance to rooted ordered trees and forests with nodes labeled over an alphabet Σ. It quantifies the dissimilarity between two forests as the minimal cost of a sequence of node


© Tomasz Kociumaka and Ali Shahali;
licensed under Creative Commons License CC-BY 4.0

edits – insertions, deletions, and relabelings – required to transform one forest into the other. This distance naturally captures both structural and label-based differences and serves as a core primitive in various algorithmic and applied domains, including computational biology [22, 39, 23, 45], analysis of structured data (such as XML and JSON files) [9, 13, 12, 20, 44], image processing [5, 26, 25, 36], and information extraction [16, 48]; see [6, 2] for surveys.

Tai [41] proposed the first polynomial-time algorithm for computing the tree edit distance: a dynamic programming procedure running in $\mathcal{O}(n^6)$ time, where n is the total number of nodes in the input forests. In the following decades, a sequence of works progressively improved the runtime. Zhang and Shasha [49] introduced an $\mathcal{O}(n^4)$-time algorithm, and then Klein [24] brought the complexity down to $\mathcal{O}(n^3 \log n)$. Subsequently, Demaine, Mozes, Rossman, and Weimann [17] presented an $\mathcal{O}(n^3)$-time solution, whereas Bringmann, Gawrychowski, Mozes, and Weimann [8] proved that a hypothetical $n^{3-\Omega(1)}$-time algorithm would violate the All-Pairs Shortest Paths (APSP) hypothesis in fine-grained complexity. Very recently, Nogler, Polak, Saha, Vassilevska Williams, Xu, and Ye [33] showed that computing the tree edit distance is, in fact, equivalent to the APSP problem, and the $n^3/2^{\Omega(\sqrt{\log n})}$ running time can be inherited from the state of the art for the latter task [46].

The aforementioned conditional lower bounds apply to the weighted tree edit distance only, where the cost of each edit depends on the involved labels. In the unweighted version with unit costs, a series of recent results achieved truly subcubic running time using fast matrix multiplication. Mao [29] presented an $\mathcal{O}(n^{2.9546})$-time solution, which was subsequently improved by Dürr [18] to $\mathcal{O}(n^{2.9148})$ and by Nogler et al. [33] to $\widetilde{\mathcal{O}}(n^{(3+\omega)/2}) = \mathcal{O}(n^{2.6857})$, where ω is the fast matrix multiplication exponent and $\widetilde{\mathcal{O}}(\cdot)$ hides $\operatorname{poly}\log n$ factors. The state-of-the-art conditional lower bound, inherited from string edit distance [4] and assuming the Orthogonal Vectors Hypothesis, prohibits $n^{2-\Omega(1)}$-time algorithms already if $|\Sigma| = 1$.

Bounded Tree Edit Distance. Even though the decades of research resulted in significantly faster tree edit distance algorithms, they also revealed fundamental challenges that explain the difficulty of this problem. A natural way of circumventing these barriers, paved by classical work for string edit distance [43, 31, 28], is to parameterize the running time not only in terms of the length n of the input forests but also the value k of the computed distance. This version of tree edit distance has originally been studied in the unweighted setting only. Touzet [42] adapted the ideas of Zhang and Shasha [49] to derive an $\mathcal{O}(nk^3)$-time algorithm, whereas Akmal and Jin [1] improved the running time to $\mathcal{O}(nk^2 \log n)$ based on the approach of Klein [24]. The latter running time remains the state-of-the-art for medium distances. In the low-distance regime, Das, Gilbert, Hajiaghayi, Kociumaka, Saha, and Saleh [15] achieved an $\widetilde{\mathcal{O}}(n + k^{15})$-time solution and subsequently improved the running time to $\mathcal{O}(n + k^7 \log k)$ [14]. They also studied the weighted version of the problem and presented an $\mathcal{O}(n + k^{15})$-time algorithm assuming that the weight function is normalized (each edit costs at least one unit; this prohibits arbitrary scaling) and satisfies the triangle inequality.

Still, the $\widetilde{\mathcal{O}}(n + \operatorname{poly}(k))$ running times for tree edit distance are much larger than for string edit distance and the related Dyck edit distance problem, which asks for the minimum cost of edits needed to make a given string of parentheses balanced (so that it represents a node-labeled forest). In the unweighted setting, the state-of-the-art running times are $\mathcal{O}(n + k^2)$ for string edit distance [28] and $\mathcal{O}(n + k^{5.442})$ for Dyck edit distance [21]. In the presence of weights, these complexities increase to $\widetilde{\mathcal{O}}(n + \sqrt{nk^3}) \le \widetilde{\mathcal{O}}(n + k^3)$ [10] and $\mathcal{O}(n + k^{12})$ [14], respectively. Thus, tree edit distance is a natural candidate for improvements.

Our Results. As the main contribution, we bring the time complexity of tree edit distance to $\widetilde{\mathcal{O}}(n+k^6)$. Even for unit weights, this improves upon the state of the art if $n^{1/7} \ll k \ll n^{1/4}$.

▶ **Theorem 1.1.** *There exists a deterministic algorithm that, given two forests F and G with n nodes in total, each with a label from an alphabet Σ, and oracle access to a normalized weight function $w : (\Sigma \cup \{\varepsilon\})^2 \to \mathbb{R}_{\geq 0}$ satisfying the triangle inequality, determines the tree edit distance $k \coloneqq \mathsf{ted}^w(F,G)$ in $\mathcal{O}(n + k^6 \log k)$ time.*

We also prove that the $\mathcal{O}(nk^2 \log n)$ running time by Akmal and Jin [1] remains valid in the presence of weights. This approach gives the best complexity when $n^{1/4} \ll k \ll n$.

▶ **Theorem 1.2.** *There exists a deterministic algorithm that, given two forests F and G, with n nodes in total, each with a label from an alphabet Σ, and oracle access to a normalized weight function $w : (\Sigma \cup \{\varepsilon\})^2 \to \mathbb{R}_{\geq 0}$, determines $k \coloneqq \mathsf{ted}^w(F,G)$ in $\mathcal{O}(nk^2 \log n)$ time.*

We believe that the original approach of Akmal and Jin [1] supports the weighted setting with minimal adjustments; nevertheless, we prove Theorem 1.2 using a slightly different algorithm that is better-suited for further optimizations. Both solutions are variants of Klein's dynamic programming [24] with some states pruned. The main difference is that we view the input forests as balanced sequences of parentheses and cast the tree edit distance as the minimum cost of a string alignment satisfying a certain consistency property. In our interpretation (Section 3), it is almost trivial to see that the classic pruning rules, originally devised to compute string edit distance in $\mathcal{O}(nk)$ time [43, 31], can be reused for tree edit distance in $\mathcal{O}(nk^2 \log n)$ time. In contrast, Akmal and Jin [1, Lemma 9] spend over four pages (on top of the analysis of [24]) to bound the number of states surviving their pruning rules.

Theorem 1.2 combined with the results of [14] lets us derive an $\mathcal{O}(n + k^7 \log k)$-time algorithm for weighted tree edit distance, matching the previously best running time for unit weights. This is due to the *universal kernel* that transforms the input forests to equivalent forests of size $\mathcal{O}(k^5)$ preserving the following capped tree edit distance value:

$$\mathsf{ted}^w_{\leq k}(F,G) = \begin{cases} \mathsf{ted}^w(F,G) & \text{if } \mathsf{ted}^w(F,G) \leq k, \\ \infty & \text{otherwise.} \end{cases}$$

▶ **Theorem 1.3** ([14, Corollary 3.20])**.** *There exists a linear-time algorithm that, given forests F, G and an integer $k \in \mathbb{Z}_+$, constructs forests F', G' of size $\mathcal{O}(k^5)$ such that $\mathsf{ted}^w_{\leq k}(F,G) = \mathsf{ted}^w_{\leq k}(F',G')$ holds for every normalized weight function w satisfying the triangle inequality.*

To bring the time complexity from $\widetilde{\mathcal{O}}(n+k^7)$ to $\widetilde{\mathcal{O}}(n+k^6)$, we adapt both Theorems 1.2 and 1.3. The main novel insight is that the dynamic-programming procedure behind Theorem 1.2 behaves predictably while processing regions with certain repetitive (periodic) structure. Upon an appropriate relaxation of the invariant that the dynamic-programming values satisfy, processing each repetition of the period can be interpreted as a single min-plus matrix-vector multiplication. When the period repeats many times, the same matrix is used each time, and thus we can raise the matrix to an appropriate power (with fast exponentiation) and then compute a single matrix-vector product; see Section 4 for details. This can be seen as a natural counterpart of an optimization used in [10] for string edit distance.[1]

[1] The matrices arising in the string edit distance computation in [10] are $\mathcal{O}(k) \times \mathcal{O}(k)$ Monge matrices, which allows for computing each matrix-vector product in $\mathcal{O}(k)$ time and each matrix-matrix product in $\mathcal{O}(k^2)$ time. In the context of tree edit distance, the Monge property is no longer satisfied, so the complexities increase to $\mathcal{O}(k^2)$ and $\mathcal{O}(k^3)$ respectively; in fact, computing the (weighted) tree edit distance is, in general, as hard as computing the min-plus product of two $n \times n$ arbitrary matrices [33].

Intuitively, the worst-case instances for the kernelization algorithm of Theorem 1.3 are close to having the necessary structure for our optimization to improve the worst-case running time. Unfortunately, the implementation in [14] controls the forest sizes only, and it needs to be modified to keep track of the more subtle instance difficulty measure. The biggest challenge is that the original algorithm proceeds in logarithmically many steps, each shrinking the forests by a constant fraction. In every step, the forests are partitioned into $\mathcal{O}(k)$ pieces of size $\mathcal{O}(n/k)$, and a constant fraction of pieces is shrunk to size $\mathcal{O}(k^4)$ each. A partition into pieces of equal "difficulty" would be much more challenging, so we instead take a different approach that lets us implement the reduction in a single step; see Section 5 for details. Notably, the kernelization algorithms for weighted string and Dyck edit distance already have single-step implementations in [14], but they crucially rely on the fact that the unit-weight variants of these problems can be solved faster, unlike for tree edit distance.

Related Work

The classical definition of tree edit distance [38] involves labeled rooted trees (or forests) with node labels. Many other variants have also been studied allowing, among others, unordered [47, 40], unrooted [40], and edge-labeled [3] trees; see also the surveys [2, 6].

The tree edit distance problem becomes easier not only when the computed distance is small but also when the input forests are of small depth [49, 11] or when the approximate distance suffices [3, 7, 37]. There is also a body of work focusing on practical performance and empirical evaluation, including sequential [34, 35] and parallel [19] implementations.

2 Preliminaries

Consistently with [14], we identify forests with the underlying Euler tours, interpreted as balanced strings of parentheses. This representation is at the heart of space-efficient data structures on trees [30, 32], and it allows for a simple definition of tree edits and a seamless transfer of many tools from strings to forests. Notably, Klein's algorithm [24] already uses substrings of the Euler tours of the input forests to identify the dynamic-programming states.

For an alphabet Σ, let $\mathsf{P}_\Sigma := \bigcup_{a\in\Sigma}\{(_a,)_a\}$ denote the set parentheses with labels in Σ. As formalized next, a *forest* with node labels over Σ is a *balanced* string of parentheses in P_Σ.

▶ **Definition 2.1.** *The set of* forests *with labels over* Σ *(*balanced strings *over* P_Σ*) is the smallest subset* $\mathcal{F}_\Sigma \subseteq \mathsf{P}_\Sigma^*$ *satisfying the following conditions:*

- $\varepsilon \in \mathcal{F}_\Sigma$,
- $F \cdot G \in \mathcal{F}_\Sigma$ *for every* $F, G \in \mathcal{F}_\Sigma$,
- $(_a \cdot F \cdot)_a \in \mathcal{F}_\Sigma$ *for every* $F \in \mathcal{F}_\Sigma$ *and* $a \in \Sigma$.

For a forest F, we define the set of *nodes* V_F as the set of intervals $[i\,..\,j] \subseteq [0\,..\,|F|)$ such that $F[i]$ is an opening parenthesis, $F[j]$ is a closing parenthesis, and $F(i\,..\,j)$ is balanced. A forest F is a *tree* if $[0\,..\,|F|) \in V_F$. For a node $u = [i\,..\,j] \in V_F$, we denote the positions of the opening and the closing parenthesis by $o(u) := i$ and $c(u) := j$. The label of u, that is, the character $a \in \Sigma$ such that $F[o(u)] = (_a$ and $F[c(u)] =)_a$, is denoted by $\mathsf{lbl}(u)$. A simple inductive argument shows that V_F forms a laminar family and, for every $i \in [0\,..\,|F|)$, there is a unique node $u \in V_F$ such that $i \in \{o(u), c(u)\}$; we denote this node by $\mathsf{node}_F(i)$. We also write $\mathsf{mate}_F(i)$ for the paired parenthesis, that is, $\mathsf{mate}_F(i) = j$ if $\{i, j\} = \{o(u), c(u)\}$.

We say that a node $u \in V_F$ *is contained* in a fragment $F[i\,..\,j)$ of a forest F if $i \le o(u) < c(u) < j$; we denote by $V_{F[i\,..\,j)} \subseteq V_F$ the set of nodes contained in $F[i\,..\,j)$. Moreover, u *enters* $F[i\,..\,j)$ if $o(u) < i \le c(u) < j$ and u *exits* $F[i\,..\,j)$ if $i \le o(u) < j \le c(u)$. In either of these two cases, we also say that u *straddles* $F[i\,..\,j)$.

We denote by $F_{[i..j)}$ the *subforest of F induced by $F[i..j)$*, which we obtain from F by deleting (the parentheses corresponding to) all nodes except for those contained in $F[i..j)$. Alternatively, one can obtain $F_{[i..j)}$ from $F[i..j)$ by deleting the opening parenthesis of every node that exits $F[i..j)$ and the closing parenthesis of every node that enters $F[i..j)$.

2.1 Tree Edits, Forest Alignments, and Tree Edit Distance

For an alphabet Σ, we define $\bar{\Sigma} := \Sigma \cup \{\varepsilon\}$, where ε is the empty string. We say that a function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$ is *normalized* if $w(a,a) = 0$ and $w(a,b) \geq 1$ hold for distinct $a, b \in \bar{\Sigma}$.

Tree edit distance is classically defined using elementary edits transforming $F \in \mathcal{F}_\Sigma$:

Node insertion produces $F[0..i) \cdot (_a \cdot F[i..j) \cdot)_a \cdot F[j..|F|)$ for a balanced fragment $F[i..j)$ and a label $a \in \Sigma$, at cost $w(\varepsilon, a)$.

Node relabeling produces $F[0..o(u)) \cdot (_a \cdot F(o(u)..c(u)) \cdot)_a \cdot F(c(u)..|F|)$ for a node $u \in V_F$ and a label $a \in \Sigma$, at cost $w(\mathsf{lbl}(u), a)$.

Node deletion produces $F[0..o(u)) \cdot F(o(u)..c(u)) \cdot F(c(u)..|F|)$ for a node $u \in V_F$, at cost $w(\mathsf{lbl}(u), \varepsilon)$.

The tree edit distance $\mathsf{ted}^w(F,G)$ of two forests $F, G \in \mathcal{F}_\Sigma$ is then defined as the minimum cost of a sequence of edits transforming F to G. In this context, without loss of generality, we can replace w by its transitive closure. If, say $w(a,\varepsilon) > w(a,b) + w(b,\varepsilon)$, instead of directly deleting a node with label a, it is more beneficial to first change its label to b and only then perform the deletion. When w satisfies the triangle inequality, we are guaranteed that an inserted or relabeled node is never modified (deleted or relabeled) again. Consistently with modern literature [8, 14, 33], we use a more general *alignment-based* definition of $\mathsf{ted}^w(F,G)$ that enforces the latter condition even if w does not necessarily satisfy the triangle inequality.

▶ **Definition 2.2** (Alignment Graph [10]). *For strings $X, Y \in \Sigma^*$ and a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, we define the* alignment graph $\mathsf{AG}^w(X,Y)$ *as a grid graph with vertices $[0..|X|] \times [0..|Y|]$ and the following directed edges:*

- *horizontal edges $(x,y) \to (x+1,y)$ of cost $w(X[x],\varepsilon)$ for $(x,y) \in [0..|X|) \times [0..|Y|]$,*
- *vertical edges $(x,y) \to (x,y+1)$ of cost $w(\varepsilon, Y[y])$ for $(x,y) \in [0..|X|] \times [0..|Y|)$, and*
- *diagonal edges $(x,y) \to (x+1,y+1)$ of cost $w(X[x],Y[y])$ for $(x,y) \in [0..|X|) \times [0..|Y|)$.*

The alignment graph allows for a concise definition of a string *alignment*.

▶ **Definition 2.3** (Alignment). *For strings $X, Y \in \Sigma^*$ and a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, an* alignment *of $X[x..x')$ onto $Y[y..y')$, denoted by $\mathcal{A} : X[x..x') \rightsquigarrow Y[y..y')$, is a path from (x,y) to (x',y') in $\mathsf{AG}^w(X,Y)$, interpreted as a sequence of vertices. The* cost $\mathsf{ed}^w_{\mathcal{A}}(X[x..x'), Y[y..y'))$ *of the alignment $\mathcal{A}$ is the total costs of the edges that belong to $\mathcal{A}$.*

We write $\mathbf{A}(X[x..x'), Y[y..y'))$ for the set of all alignments of $X[x..x')$ onto $Y[y..y')$.

The edges of an alignment $\mathcal{A} = \mathbf{A}(X[x..x'), Y[y..y'))$ can be interpreted as follows:

- If $\mathcal{A}$ includes an edge $(\hat{x},\hat{y}) \to (\hat{x}+1,\hat{y})$ for some $\hat{x} \in [x..x')$ and $\hat{y} \in [y..y']$, then $\mathcal{A}$ *deletes* $X[\hat{x}]$, denoted by $X[\hat{x}] \rightsquigarrow_{\mathcal{A}} \varepsilon$.
- If $\mathcal{A}$ includes an edge $(\hat{x},\hat{y}) \to (\hat{x},\hat{y}+1)$ for some $\hat{x} \in [x..x']$ and $\hat{y} \in [y..y')$, then $\mathcal{A}$ *inserts* $Y[\hat{y}]$, denoted by $\varepsilon \rightsquigarrow_{\mathcal{A}} Y[\hat{y}]$.
- If $\mathcal{A}$ includes an edge $(\hat{x},\hat{y}) \to (\hat{x}+1,\hat{y}+1)$ for some $\hat{x} \in [x..x')$ and $\hat{y} \in [y..y')$, then $\mathcal{A}$ *aligns* $X[\hat{x}]$ to $Y[\hat{y}]$, denoted by $X[\hat{x}] \rightsquigarrow_{\mathcal{A}} Y[\hat{y}]$. If $X[\hat{x}] \neq Y[\hat{y}]$, then $\mathcal{A}$ *substitutes* $X[\hat{x}]$ for $Y[\hat{y}]$. If $X[\hat{x}] = Y[\hat{y}]$, then $\mathcal{A}$ *matches* $X[\hat{x}]$ with $Y[\hat{y}]$.

Insertions, deletions, and substitutions are jointly called character *edits*.

The weighted edit distance of fragments $X[x\,..\,x')$ and $Y[y\,..\,y')$ with respect to a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$ is defined as the minimum cost of an alignment:

$$\mathsf{ed}^w(X[x\,..\,x'), Y[y\,..\,y')) = \min_{\mathcal{A}\in\mathbf{A}(X[x\,..\,x'),Y[y\,..\,y'))} \mathsf{ed}^w_{\mathcal{A}}(X[x\,..\,x'), Y[y\,..\,y')).$$

We often consider alignments of the entire string X onto the entire string Y; we then used simplified notation including $\mathbf{A}(X,Y)$, $\mathsf{ed}^w_{\mathcal{A}}(X,Y)$, and $\mathsf{ed}^w(X,Y)$.

▶ **Definition 2.4** (Forest alignment)**.** *Consider forests $F, G \in \mathcal{F}_\Sigma$. An alignment $\mathcal{A} \in \mathbf{A}(F[f\,..\,f'), G[g\,..\,g'))$ is a* forest alignment *if it satisfies the following consistency condition:*

For every two aligned characters $F[\hat{f}] \rightsquigarrow_{\mathcal{A}} G[\hat{g}]$, also $F[\mathsf{mate}_F(\hat{f})] \rightsquigarrow_{\mathcal{A}} G[\mathsf{mate}_G(\hat{g})]$.

We write $\mathbf{FA}(F[f\,..\,f'), G[g\,..\,g')) \subseteq \mathbf{A}(F[f\,..\,f'), G[g\,..\,g'))$ for the set of forest alignments.

▶ Remark 2.5. Consider a forest alignment $\mathcal{A} \in \mathbf{FA}(F[f\,..\,f'), G[g\,..\,g'))$. Then,

- $\mathcal{A}$ deletes every character $F[\hat{f}]$ with $\hat{f} \in [f\,..\,f')$ and $\mathsf{mate}_F(\hat{f}) \notin [f\,..\,f')$,
- $\mathcal{A}$ inserts every character $G[\hat{g}]$ with $\hat{g} \in [g\,..\,g')$ and $\mathsf{mate}_G(\hat{g}) \notin [g\,..\,g')$.

Define $\overline{\mathsf{P}_\Sigma} = \mathsf{P}_\Sigma \cup \{\varepsilon\}$ and a mapping $\lambda : \overline{\mathsf{P}_\Sigma} \to \bar{\Sigma}$ such that $\lambda(\langle_a) = \lambda(\rangle_a) = a$ for each $a \in \Sigma$, and $\lambda(\varepsilon) = \varepsilon$. For a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, we define a corresponding weight function $\tilde{w} : \overline{\mathsf{P}_\Sigma}^2 \to \mathbb{R}_{\geq 0}$ so that $\tilde{w}(p,q) = \frac{1}{2}w(\lambda(p), \lambda(q))$ for all $p, q \in \overline{\mathsf{P}_\Sigma}$. The *cost* of a forest alignment $\mathcal{A} \in \mathbf{FA}(F[f\,..\,f'), G[g\,..\,g'))$ with respect to a weight function w is defined as $\mathsf{ted}^w_{\mathcal{A}}(F[f\,..\,f'), G[g\,..\,g')) := \mathsf{ed}^{\tilde{w}}_{\mathcal{A}}(F[f\,..\,f'), G[g\,..\,g'))$. Moreover, we define

$$\mathsf{ted}^w(F[f\,..\,f'), G[g\,..\,g')) = \min_{\mathcal{A}\in\mathbf{FA}(F[f\,..\,f'),G[g\,..\,g'))} \mathsf{ted}^w_{\mathcal{A}}(F[f\,..\,f'), G[g\,..\,g')).$$

For a threshold $k \in \mathbb{R}_{\geq 0}$, we set

$$\mathsf{ted}^w_{\leq k}(F[f\,..\,f'), G[g\,..\,g')) = \begin{cases} \mathsf{ted}^w(F[f\,..\,f'), G[g\,..\,g')) & \text{if } \mathsf{ted}^w(F[f\,..\,f'), G[g\,..\,g')) \leq k, \\ \infty & \text{otherwise.} \end{cases}$$

By Remark 2.5, the following value is non-negative for every $\mathcal{A} \in \mathbf{FA}(F[f\,..\,f'), G[g\,..\,g'))$:

$$\widetilde{\mathsf{ted}}^w_{\mathcal{A}}(F[f\,..\,f'), G[g\,..\,g')) \coloneqq \mathsf{ted}^w_{\mathcal{A}}(F[f\,..\,f'), G[g\,..\,g')) - \sum_{\substack{\hat{f}\in[f\,..\,f'):\\ \mathsf{mate}_F(\hat{f})\notin[f\,..\,f')}} \tilde{w}(F[\hat{f}], \varepsilon) - \sum_{\substack{\hat{g}\in[g\,..\,g'):\\ \mathsf{mate}_G(\hat{g})\notin[g\,..\,g')}} \tilde{w}(\varepsilon, G[\hat{g}]).$$

We naturally generalize this value to $\widetilde{\mathsf{ted}}^w(F[f\,..\,f'), G[g\,..\,g'))$.

▶ **Observation 2.6.** *For all forests $F, G \in \mathcal{F}_\Sigma$, fragments $F[f\,..\,f')$ and $G[g\,..\,g')$, and weight functions $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, we have $\widetilde{\mathsf{ted}}^w(F[f\,..\,f'), G[g\,..\,g')) = \mathsf{ted}^w(F_{[f\,..\,f')}, G_{[g\,..\,g')})$.*

3 $\mathcal{O}(nk^2 \log n)$-Time Algorithm

In this section, we reinterpret Klein's algorithm [24] and develop its $\mathcal{O}(nk^2 \log n)$-time variant.

3.1 Klein's Algorithm

Klein's algorithm [24] uses dynamic programming to compute $\mathsf{ted}(F_{[l_F..r_F)}, G_{[l_G..r_G)}) = \widetilde{\mathsf{ted}}(F[l_F..r_F), G[l_G..r_G))$ for $\mathcal{O}(n \log n)$ selected fragments $F[l_F..r_F)$ of F and all $\mathcal{O}(n^2)$ fragments $G[l_G..r_G)$ of G. We modify the algorithm slightly so that the computed value $\mathsf{ted}(F[l_F..r_F), G[l_G..r_G))$ includes the costs of deleting the single parentheses of nodes straddling $F[l_F..r_F)$ and inserting the single parentheses of nodes straddling $G[l_G..r_G)$.

Algorithm 1 `Klein`(l_F, r_F, l_G, r_G): Klein's algorithm for computing the tree edit distance.

Input: Two fragments $F[l_F..r_F)$ and $G[l_G..r_G)$ of the input forests
Output: Compute and store the value of $\mathsf{dp}[l_F, r_F, l_G, r_G]$

1 **if** $l_F = r_F$ **then** $\mathsf{result} \leftarrow \sum_{i \in [l_G..r_G)} \tilde{w}(\varepsilon, G[i])$;
2 **else if** $l_G = r_G$ **then** $\mathsf{result} \leftarrow \sum_{i \in [l_F..r_F)} \tilde{w}(F[i], \varepsilon)$;
3 **else**
4 $\quad u_F = \mathsf{node}_F(l_F)$, $v_F = \mathsf{node}_F(r_F - 1)$, $u_G = \mathsf{node}_G(l_G)$, $v_G = \mathsf{node}_G(r_G - 1)$;
5 $\quad$ **if** $u_F \notin V_{F[l_F..r_F)}$ **or** ($v_F \in V_{F[l_F..r_F)}$ **and** $\mathsf{size}(u_F) \le \mathsf{size}(v_F)$) **then**
6 $\quad\quad \mathsf{result} \leftarrow \mathsf{dp}[l_F + 1, r_F, l_G, r_G] + \tilde{w}(F[l_F], \varepsilon)$;
7 $\quad\quad \mathsf{result} \overset{\min}{\leftarrow} \mathsf{dp}[l_F, r_F, l_G + 1, r_G] + \tilde{w}(\varepsilon, G[l_G])$;
8 $\quad\quad$ **if** $u_F \in V_{F[l_F..r_F)}$ **and** $u_G \in V_{G[l_G..r_G)}$ **then**
9 $\quad\quad\quad \mathsf{result} \overset{\min}{\leftarrow} \tilde{w}(F[l_F], G[l_G]) + \mathsf{dp}[l_F + 1, c(u_F), l_G + 1, c(u_G)] + \tilde{w}(F[c(u_F)], G[c(u_G)]) + \mathsf{dp}[c(u_F) + 1, r_F, c(u_G) + 1, r_G]$;
10 $\quad$ **else**
11 $\quad\quad \mathsf{result} \leftarrow \mathsf{dp}[l_F, r_F - 1, l_G, r_G] + \tilde{w}(F[r_F - 1], \varepsilon)$;
12 $\quad\quad \mathsf{result} \overset{\min}{\leftarrow} \mathsf{dp}[l_F, r_F, l_G, r_G - 1] + \tilde{w}(\varepsilon, G[r_G - 1])$;
13 $\quad\quad$ **if** $v_F \in V_{F[l_F..r_F)}$ **and** $v_G \in V_{G[l_G..r_G)}$ **then**
14 $\quad\quad\quad \mathsf{result} \overset{\min}{\leftarrow} \mathsf{dp}[l_F, o(v_F), l_G, o(v_G)] + \tilde{w}(F[o(v_F)], G[o(v_G)]) + \mathsf{dp}[o(v_F) + 1, r_F - 1, o(v_G) + 1, r_G - 1] + \tilde{w}(F[r_F - 1], G[r_G - 1])$;
15 $\mathsf{dp}[l_F, r_F, l_G, r_G] \leftarrow \mathsf{result}$

The algorithm's implementation is provided in Algorithm 1. We use an operator $x \overset{\min}{\leftarrow} y$ that assigns $x \leftarrow y$ if $y < x$. We also remember which of these assignments were applied so that we can later trace back the optimal alignment based on this extra information stored.

In the corner case when $F[l_F..r_F)$ or $G[l_G..r_G)$ is empty, then the unique (and thus optimal) forest alignment pays for inserting or deleting all characters in the other fragment. If both $F[l_F..r_F)$ and $G[l_G..r_G)$ are non-empty, the algorithm considers nodes $u_F = \mathsf{node}_F(l_F)$, $u_G = \mathsf{node}_G(l_G)$, $v_F = \mathsf{node}_F(r_F - 1)$, and $v_G = \mathsf{node}_G(r_G - 1)$.

Let us first suppose that u_F and v_F are contained in $F[l_F..r_F)$. If the subtree of v_F is at least as large as the subtree of u_F, that is, $\mathsf{size}(v_F) \ge \mathsf{size}(u_F)$, where $\mathsf{size}(x) = c(x) - o(x) + 1$, then algorithm considers three possibilities: $F[l_F]$ is deleted, $G[l_G]$ is inserted, or $F[l_F]$ is aligned with $G[l_G]$. In the first two possibilities, we can pay for the deleted or inserted opening parenthesis and align the remaining fragments; see Lines 6–7. In the third possibility, the consistency condition in the definition of the forest alignments requires that $F[\mathsf{mate}_F(l_F)]$ is also aligned with $G[\mathsf{mate}_G(l_G)]$. In particular, the node u_G needs to be contained in $G[l_G..r_G)$ so that $\mathsf{mate}_G(l_G) = c(u_G)$ and $\mathsf{mate}_F(l_F) = c(u_F)$. Thus, we align $F(l_F..c(u_F))$ with $G(l_G..c(u_G))$, align $F(c(u_F)..r_F)$ with $G(c(u_G)..r_G)$, and pay for aligning u_F with u_G, that is $F[l_F]$ with $G[l_G]$ and $F[c(u_F)]$ with $G[c(u_G)]$; see Line 9. If $\mathsf{size}(v_F) < \mathsf{size}(u_F)$, the algorithm handles v_F and v_G in a symmetric way; see Lines 10–14.

If $u_F \notin V_{F[l_F..r_F)}$, we follow Lines 5–9 and effectively consider deleting $F[l_F]$ and inserting $G[l_G]$. Else, if $v_F \notin V_{F[l_F..r_F)}$, we follow the symmetric Lines 10–14.

Zhang and Shasha [49] use a very similar dynamic programming; in a baseline version, their algorithm always constructs the alignment from left to right instead of picking the side depending on u_F and v_F. Klein's optimization allows for an improved running time of $\mathcal{O}(n^3 \log n)$ instead of $\mathcal{O}(n^4)$. In the full version [27], we recall the proof of the following lemma.

▶ **Lemma 3.1.** *The recursive implementation of Algorithm 1 visits $\mathcal{O}(n \log n)$ fragments of F.*

Akmal and Jin's Algorithm. Akmal and Jin [1] reduce the time complexity of Klein's algorithm to $\mathcal{O}(nk^2 \log n)$ when computing $\mathsf{ted}_{\le k}(F, G)$. Their solutions prunes some dynamic programming states so that, in the surviving states, the differences between the sizes of the subforests $F_{[l_F..r_F)}$ and $G_{[l_G..r_G)}$ is at most k. This condition also holds for the difference between the sizes of $F_{[0..r_F)} \setminus F_{[l_F..r_F)}$ and $G_{[0..l_G)} \setminus G_{[l_G..r_G)}$, as well as between $F_{[l_F..|F|)} \setminus F_{[l_F..r_F)}$ and $G_{[l_G..|G|)} \setminus G_{[l_G..r_G)}$; see [1, Lemma 12]. As shown in [1, Lemma 13], for each subforest $F_{[l_F..r_F)}$, there are $\mathcal{O}(k^2)$ subforests $G_{[l_G..r_G)}$ satisfying these three conditions. With a careful implementation, the total number of states becomes $\mathcal{O}(nk^2 \log n)$.

3.2 Our Algorithm

Our variant of Klein's algorithm simply prunes all states with $|l_F - l_G| > 2k$ or $|r_F - r_G| > 2k$. In other words, we modify Algorithm 1 so that $\mathsf{dp}[l_F, r_F, l_G, r_G]$ is set to ∞ if either condition holds, and the existing instructions are executed otherwise. This does not increase the number $\mathcal{O}(n \log n)$ of visited fragments $F[l_F..r_F)$; see Lemma 3.1. For each of these fragments, trivially, at most $\mathcal{O}(k^2)$ fragments of G survive pruning, so the total number of states is $\mathcal{O}(nk^2 \log n)$. The correctness of the pruning rules follows from the fact that, for all forests F, G, the width of any alignment in $\mathbf{FA}(F, G)$ does not exceed twice its cost, where the width of an alignment $\mathcal{A} \in \mathbf{A}(X[x..x'), Y[y..y'))$ is defined as $\mathsf{width}(\mathcal{A}) = \max\{|\hat{x} - \hat{y}| : (\hat{x}, \hat{y}) \in \mathcal{A}\}$.

In the full version [27], we formalize this intuition using the notion of bounded forest alignments.

▶ **Definition 3.2.** *Let us fix a threshold $k \in \mathbb{Z}_+$ and consider fragments $F[f..f')$ and $G[g..g')$ of forests $F, G \in \mathcal{F}_\Sigma$. We call a forest alignment $\mathcal{A} \in \mathbf{FA}(F[f..f'), G[g..g'))$* bounded *if $\mathsf{width}(\mathcal{A}) \le 2k$. The family of* bounded forest alignments *is $\mathbf{BFA}_k(F[f..f'), G[g..g')) \subseteq \mathbf{FA}(F[f..f'), G[g..g'))$. For a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\ge 0}$, we denote*

$$\mathsf{bted}^w_k(F[f..f'), G[g..g')) = \min_{\mathcal{A} \in \mathbf{BFA}_k(F[f..f'), G[g..g'))} \mathsf{ted}^w_{\mathcal{A}}(F[f..f'), G[g..g')).$$

▶ **Lemma 3.3.** *The dp values computed using the pruned version of Algorithm 1 satisfy*

$$\mathsf{ted}^w(F[l_F..r_F), G[l_G..r_G)) \le \mathsf{dp}[l_F, r_F, l_G, r_G] \le \mathsf{bted}^w_k(F[l_F..r_F), G[l_G..r_G)).$$

In particular, the $\mathsf{dp}[0, |F|, 0, |G|]$ entry stores a value between $\mathsf{ted}^w(F, G)$ and $\mathsf{bted}^w_k(F, G)$. The following observation implies that this is enough to retrieve $\mathsf{ted}^w_{\le k}(F, G)$.

▶ **Observation 3.4.** *If $\mathsf{ted}^w(F, G) \le k$, then $\mathsf{ted}^w(F, G) = \mathsf{bted}^w_k(F, G)$.*

Proof. Consider the optimal forest alignment $\mathcal{A} \in \mathbf{FA}(F, G)$ with $\mathsf{ted}^w_{\mathcal{A}}(F, G) \le k$. For each edge $(f, g) \to (f', g')$, either $f - g = f' - g'$ (if the edge is diagonal) or $\big|(f-g)-(f'-g')\big| = 1$ and the edge cost is at least $\frac{1}{2}$ (if the edge is vertical or horizontal). Since $(0, 0) \in \mathcal{A}$ and the total cost of edges in $\mathcal{A}$ is at most k, every $(f, g) \in \mathcal{A}$ satisfies $|f - g| \le 2k$, i.e., $\mathsf{width}(\mathcal{A}) \le 2k$ and $\mathcal{A} \in \mathbf{BFA}_k(F, G)$. Hence, $\mathsf{bted}^w_k(F, G) \le \mathsf{ted}^w_{\mathcal{A}}(F, G) = \mathsf{ted}^w(F, G)$. The converse inequality $\mathsf{ted}^w(F, G) \le \mathsf{bted}^w_k(F, G)$ holds trivially due to $\mathbf{BFA}_k(F, G) \subseteq \mathbf{FA}(F, G)$. ◀

With minor implementation details needed to avoid logarithmic overheads for memoization using a sparse table dp (Akmal and Jin [1] ignore this issue), we achieve the following result.

▶ **Theorem 1.2.** *There exists a deterministic algorithm that, given two forests F and G, with n nodes in total, each with a label from an alphabet Σ, and oracle access to a normalized weight function $w : (\Sigma \cup \{\varepsilon\})^2 \to \mathbb{R}_{\geq 0}$, determines $k := \mathsf{ted}^w(F, G)$ in $\mathcal{O}(nk^2 \log n)$ time.*

4 Faster Algorithm for Repetitive Inputs

In this section, we present an optimized version of our $\mathcal{O}(nk^2 \log n)$-time algorithm capable of exploiting certain repetitive structures within the input forests F and G. The following notion of *free pairs* captures the structure that our algorithm is able to utilize.

▶ **Definition 4.1** (Free pair, free block)**.** *Consider forests $F, G \in \mathcal{F}_\Sigma$ and a fixed threshold $k \in \mathbb{Z}_+$. We call a pair of fragments $F[p_F \mathinner{..} q_F)$ and $G[p_G \mathinner{..} q_G)$ a* free pair *if*

- $|p_F - p_G| \leq 2k$, *and*
- *there exists a balanced string $R \in \mathcal{F}_\Sigma$ with $4k \leq |R| < 8k$ such that $F[p_F - |R| \mathinner{..} q_F + |R|) = R^{e+2} = G[p_G - |R| \mathinner{..} q_G + |R|)$ holds for some integer exponent $e \in \mathbb{Z}_+$.*

For that free pair, we call the fragment $F[p_F \mathinner{..} q_F)$ a free block *with period R and exponent e.*

Our improved algorithm assumes that the input forests F and G are augmented with a collection $\mathbf{F}$ of disjoint free blocks $F[p_F \mathinner{..} q_F)$, each associated with the underlying period R, exponent e, and the corresponding fragment $G[p_G \mathinner{..} q_G)$. The speed-up compared to the algorithm of Section 3 is noticeable if the free blocks in $\mathbf{F}$ jointly cover most of the characters of F, that is, the number of remaining *non-free* characters is asymptotically smaller than $|F|$.

▶ **Theorem 4.2.** *There exists a deterministic algorithm that, given forests $F, G \in \mathcal{F}_\Sigma$ of total length n, oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, a threshold $k \in \mathbb{Z}_+$, and a collection $\mathbf{F}$ of t disjoint free blocks in F such that m characters of F are not contained in any free block, computes $\mathsf{ted}^w_{\leq k}(F, G)$ in $\mathcal{O}(n \log n + mk^2 \log n + tk^3 \log n)$ time.*

Intuitively, the algorithm skips all the free blocks, which allows reducing the $\mathcal{O}(nk^2 \log n)$ running time of Theorem 1.2 to $\mathcal{O}(mk^2 \log n)$, i.e., the algorithm needs to pay for non-free characters only. Nevertheless, processing each free block takes $\mathcal{O}(k^3 \log n)$ extra time, and $\mathcal{O}(n \log n)$-time preprocessing time is still needed to avoid overheads for memoization.

Processing Free Blocks. To understand our speed-up, let us consider a free pair with period R and exponent e, that is, $F[p_F \mathinner{..} q_F) = R^e = G[p_G \mathinner{..} q_G)$. We picked the parameters so that $|R| \geq 4k \geq \mathsf{width}(\mathcal{A}) + |p_F - p_G|$, and thus every alignment $\mathcal{A} \in \mathbf{BFA}_k(F, G)$ aligns $F[p_F \mathinner{..} q_F) = R^e$ with a fragment $G[p'_G \mathinner{..} q'_G)$ contained in $G[p_G - |R| \mathinner{..} q_G + |R|) = R^{e+2}$. By the same argument, the image of every copy of R within $F[p_F \mathinner{..} q_F) = R^e$ is contained within the corresponding copy of R^3 within $G[p_G - |R| \mathinner{..} q_G + |R|) = R^{e+2}$. Moreover, when we align $F[p_F \mathinner{..} q_F)$ to $G[p'_F \mathinner{..} q'_F) \subseteq G[p_G - |R| \mathinner{..} q_G + |R|)$, it suffices to partition $G[p'_F \mathinner{..} q'_F)$ into e fragments and *independently* optimally align each copy of R in $F[p_F \mathinner{..} q_F)$ with the corresponding fragment of $G[p'_F \mathinner{..} q'_F)$. This is because R is balanced, so every node of R^e is contained within a single copy of R, and thus the consistency condition in Definition 2.4 does not impose any constraints affecting multiple copies of R.

The optimal costs of aligning R with relevant fragments of R^3 can be encoded in the following matrix, constructible in $\mathcal{O}(|R|^3 \log |R|)$ time using Algorithm 1 (Klein's algorithm).

▶ **Definition 4.3.** *For a balanced string $R \in \mathcal{F}_\Sigma$, we define a matrix M_R of size $(2|R|+1) \times (2|R|+1)$ with indices i, j in the range $[-|R| \,..\, |R|]$ as follows:*

$$M_R[i,j] = \begin{cases} \mathsf{ted}^w(R, R^3[|R|+i \,..\, 2|R|+j)) & \text{if } |R|+i \le 2|R|+j, \\ \infty & \text{otherwise.} \end{cases}$$

In order to derive the optimal costs of aligning $R^e = F[p_F \,..\, q_F)$ with the relevant fragments of $R^{e+2} = G[p_G - |R| \,..\, q_G + |R|)$, we simply compute the e-th power of M_R with respect to the min-plus product. Formally, the min-plus product of matrices $A \in \mathbb{R}^{I \times J}$ and $B \in \mathbb{R}^{J \times K}$ is a matrix $C \in \mathbb{R}^{I \times K}$ such that $C[i,k] = \min_{j \in J} A[i,j] + B[j,k]$ for $(i,k) \in I \times K$.

For each free block $F[p_F \,..\, q_F) = R^e \in \mathbf{F}$, we construct the matrix M_R^e in $\mathcal{O}(k^3 \log n)$ time using Algorithm 1 followed by fast exponentiation, which reduces to computing $\mathcal{O}(\log e)$ min-plus products. We apply the matrix whenever we are tasked with filling a $\mathsf{dp}[l_F, r_F, l_G, r_G]$ entry such that $F[p_F \,..\, q_F)$ is a prefix or a suffix of $F[l_F \,..\, r_F)$. Formally, if $l_F \le p_F < q_F = r_F$, then we use the following formula instead of following Algorithm 1:

$$\mathsf{dp}[l_F, r_F, l_G, r_G] \leftarrow \min_{p'_G \in [p_F - |R| \,..\, p_F + |R|]} \mathsf{dp}[l_F, p_F, l_G, p'_G] + M_R^e[p'_G - p_G, r_G - q_G]. \tag{1}$$

In the symmetric case of $l_F = p_F < q_F \le r_F$, we apply

$$\mathsf{dp}[l_F, r_F, l_G, r_G] \leftarrow \min_{q'_G \in [q_F - |R| \,..\, q_F + |R|]} \mathsf{dp}[q_F, r_F, q'_G, r_G] + M_R^e[l_G - p_G, q'_G - q_G]. \tag{2}$$

In the full version [27], we formalize the intuition above to prove that Equations (1) and (2) preserve the invariant of Lemma 3.3, that is,

$$\mathsf{ted}^w(F[l_F \,..\, r_F), G[l_G \,..\, r_G)) \le \mathsf{dp}[l_F, r_F, l_G, r_G] \le \mathsf{bted}_k^w(F[l_F \,..\, r_F), G[l_G \,..\, r_G)).$$

For (1), this boils down to the following lemma; the case of (2) is symmetric.

▶ **Lemma 4.4.** *Consider fragments $F[l_F \,..\, r_F)$, $G[l_G \,..\, r_G)$ of forests $F, G \in \mathcal{F}_\Sigma$, a normalized weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\ge 0}$, and a threshold $k \in \mathbb{Z}_+$ such that $|r_F - r_G| \le 2k$. If there is a free pair $F[p_F \,..\, q_F) = G[p_G \,..\, q_G) = R^e$ such that $F[p_F \,..\, q_F)$ is a suffix of $F[l_F \,..\, r_F)$, then*

$$\begin{aligned} &\mathsf{bted}_k^w(F[l_F \,..\, r_F), G[l_G \,..\, r_G)) \\ &\quad \ge \min_{p'_G \in [p_F - |R| \,..\, p_F + |R|]} \mathsf{bted}_k^w(F[l_F \,..\, p_F), G[l_G \,..\, p'_G)) + M_R^e[p'_G - p_G, r_G - q_G] \\ &\quad \ge \min_{p'_G \in [p_F - |R| \,..\, p_F + |R|]} \mathsf{ted}^w(F[l_F \,..\, p_F), G[l_G \,..\, p'_G)) + M_R^e[p'_G - p_G, r_G - q_G] \\ &\quad \ge \mathsf{ted}^w(F[l_F \,..\, r_F), G[l_G \,..\, r_G)). \end{aligned}$$

We further argue in the full version [27] that the recursive implementation of Algorithm 1 augmented with the optimizations of (1) and (2) visits $\mathcal{O}(m \log n)$ fragments $F[l_F \,..\, r_F)$, where m is the number of non-free characters, including $\mathcal{O}(t \log n)$ fragments $F[l_F \,..\, r_F)$ for which (1) or (2) apply. Specifically, our optimized algorithm visits fragments $F[l_F \,..\, r_F)$ visited by the original Algorithm 1 and satisfying the following additional property: every free block $F[p_F \,..\, q_F) \in \mathbf{F}$ is either disjoint with $F[l_F \,..\, r_F)$ or contained in $F[l_F \,..\, r_F)$. Our implementation takes $\mathcal{O}(n \log n)$ extra preprocessing time to list fragments visited by Algorithm 1 and filter those satisfying the aforementioned property.

5 Universal Kernel with Improved Repetitiveness Guarantees

In this section, we outline our approach to strengthen Theorem 1.3 into the following result:

▶ **Theorem 5.1.** *There exists a linear-time algorithm that, given forests F, G and an integer $k \in \mathbb{Z}_+$, constructs forests F', G' of size $\mathcal{O}(k^5)$ such that $\mathsf{ted}^w_{\le k}(F', G') = \mathsf{ted}^w_{\le k}(F, G)$ holds for every normalized* quasimetric *w (weight function satisfying the triangle inequality), and a collection of $\mathcal{O}(k^3)$ disjoint free blocks in F' with $\mathcal{O}(k^4)$ non-free characters.*

Compared to Theorem 1.3, we require the presence of $\mathcal{O}(k^3)$ free blocks that jointly capture all but $\mathcal{O}(k^4)$ characters of the output forest F'; consult Definition 4.1. At the very high level, the proofs of both Theorems 1.3 and 5.1 consist in three steps: decomposing the input forests F and G into several pieces, identifying pieces that can be matched exactly, and replacing (pairs of) identical pieces with smaller equivalent counterparts.

Forest Decompositions and Piece Matchings. Following [14], we say that a *piece* of a forest F is a *subforest* – a balanced fragment $F[i\,..\,j)$ – or a *context* – a pair of fragments $\langle F[i\,..\,i'); F[j'\,..\,j)\rangle$ such that $F[i\,..\,j)$ is a tree and $F[i'\,..\,j')$ is balanced. We denote the set of pieces contained in a fragment $F[i\,..\,j)$ of F by $\mathcal{P}(F[i\,..\,j))$; we set $\mathcal{P}(F) = \mathcal{P}(F[0\,..\,|F|))$.

In isolation from F, a context can be interpreted as a pair of non-empty strings $C = \langle C_L; C_R\rangle \in \mathsf{P}^+_\Sigma \times \mathsf{P}^+_\Sigma$ such that $C_L \cdot C_R$ is a tree. The *composition* of contexts C, D results in a context $C \star D := \langle C_L \cdot D_L; D_R \cdot C_R\rangle$. Moreover, the composition of a context C and a forest H results in a tree $C \star H := C_L \cdot H \cdot C_R$. For any *decomposition* of a forest F into disjoint pieces, one can recover F using the concatenation and composition operations.

We define the *depth* of a context $C = \langle C_L; C_R\rangle$ to be the number nodes of $C_L \cdot C_R$ with the opening parenthesis in C_L and the closing parenthsis in C_R. Note that the depth of the context $C \star D$ is equal to the sum of the depths of C and D.

The following notion formalizes the concept of a matching between pieces of F and G.

▶ **Definition 5.2.** *For two forests F and G and a fixed threshold $k \in \mathbb{Z}_+$, a* piece matching *between F and G is a set of pairs $\mathcal{M} \subseteq \mathcal{P}(F) \times \mathcal{P}(G)$ such that:*

- *across all pairs $(f, g) \in \mathcal{M}$, the pieces $f \in \mathcal{P}(F)$ are pairwise disjoint, and*
- *there exists a forest alignment $\mathcal{A} \in \mathbf{FA}(F, G)$ of width at most $2k$ (i.e., $\mathcal{A} \in \mathbf{BFA}_k(F, G)$) that matches f to g perfectly for every $(f, g) \in \mathcal{M}$.*

The kernelization algorithm behind Theorem 1.3 repeatedly identifies a piece matching $\mathcal{M}$ of size $|\mathcal{M}| = \mathcal{O}(k)$ covering $\Omega(n)$ vertices of F and replaces each pair of matching pieces $(f, g) \in \mathcal{M}$ with a pair of "equivalent" pieces (f', g') of size $\mathcal{O}(k^4)$. After $\mathcal{O}(\log n)$ steps, this yields forests of size $\mathcal{O}(k^5)$. Our strategy relies on the following new result:

▶ **Theorem 5.3.** *There exists a linear-time algorithm that, given forests $F, G \in \mathcal{F}_\Sigma$ and a threshold $k \in \mathbb{Z}_{\ge 0}$, either certifies that $\mathsf{ted}(F, G) > k$ or constructs a size-$\mathcal{O}(k)$ piece matching $\mathcal{M}$ between F and G that leaves $\mathcal{O}(k^4)$ unmatched characters.*

The proof of Theorem 5.3, presented in the full version [27], reuses a subroutine of [14] to construct a decomposition $\mathcal{D} \subseteq \mathcal{P}(F)$ of F into $\mathcal{O}(n/k^3)$ pieces of size $\mathcal{O}(k^3)$ each. The next step is to build a piece matching $\mathcal{M} \subseteq \mathcal{D} \times \mathcal{P}(G)$ that leaves at most k pieces of $\mathcal{D}$ unmatched. We modify a dynamic-programming procedure from [14] so that, additionally, the unmatched characters of G form $\mathcal{O}(k)$ fragments. As a result, even though the obtained matching is of size $|\mathcal{M}| = \mathcal{O}(n/k^3)$, it is possible to reduce its size to $\mathcal{O}(k)$. For this, it suffices to repeatedly identify pairs of adjacent pieces $f, f' \in \mathcal{P}(F)$ matched to adjacent

pieces $g, g' \in \mathcal{P}(G)$, and then replace these pieces with their unions $f \cup f'$ and $g \cup g'$ (as formalized in the full version [27], two disjoint pieces are adjacent if their union, consisting of the characters contained in at least one of these pieces, can be interpreted as a piece).

Periodic Blocks and Red Characters. The main ingredient of our kernelization algorithm is a procedure that replaces a pair of matching pieces $(f, g) \in \mathcal{M}$ with a pair of "equivalent" smaller pieces (f', g'). Unlike [14], where the goal was to reduce the piece size to $\mathcal{O}(k^4)$, we aim to identify $\mathcal{O}(k^2)$ disjoint free blocks with $\mathcal{O}(k^3)$ non-free nodes within the replacement piece f'. Unfortunately, free blocks lack a canonical construction, and it would be tedious to maintain a specific selection while the forests change. Instead, we use the following notions:

▶ **Definition 5.4.** *For a fixed threshold $k \in \mathbb{Z}_+$, we say that a string $S \in \mathsf{P}^*_\Sigma$ forms a* periodic block *if it satisfies the following properties:*

- *$|S| \geq 42k$, that is, the fragment is of length at least $42k$, and*
- *S has a string period of length at most $4k$ with equally many opening and closing parentheses.*

For a string $T \in \mathsf{P}^*_\Sigma$, we denote by $\mathbf{B}_k(T)$ the set of fragments of T that are periodic blocks. For a context $C = \langle C_L; C_R \rangle$, we denote by $\mathbf{B}_k(C)$ the disjoint union of $\mathbf{B}_k(C_L)$ and $\mathbf{B}_k(C_R)$.

We partition the characters of $T \in \mathsf{P}^*_\Sigma$ into *black* and *red* based on the family $\mathbf{B}_k(T)$.

▶ **Definition 5.5.** *A character $T[i]$ of a string $T \in \mathsf{P}^*_\Sigma$ is* black *if there exists a periodic block $T[l\,..\,r) \in \mathbf{B}_k(T)$ such that $i \in [l + 5k\,..\,r - 5k)$. The remaining characters are* red*. We denote by $\mathsf{black}_k(T)$ and $\mathsf{red}_k(T)$ the sets of black and red characters of T.*

For a context $C = \langle C_L; C_R \rangle$, the sets $\mathsf{black}_k(C) = \mathsf{black}_k(C_L) \sqcup \mathsf{black}_k(C_R)$ and $\mathsf{red}_k(C) = \mathsf{red}_k(C_L) \sqcup \mathsf{red}_k(C_R)$ are defined as disjoint unions.

Piece Reduction. The following definitions in [14] formalize the concept of equivalent pieces.

▶ **Definition 5.6** ([14, Definition 3.4])**.** *For a threshold $k \in \mathbb{Z}_{\geq 0}$ and a weight function w, forests P, P' are called* $\mathsf{ted}^w_{\leq k}$-equivalent *if*

$$\mathsf{ted}^w_{\leq k}(F, G) = \mathsf{ted}^w_{\leq k}(F[0\,..\,l_F) \cdot P' \cdot F[r_F\,..\,|F|), G[0\,..\,l_G) \cdot P' \cdot G[r_G\,..\,|G|))$$

holds for all forests F and G with matching pieces $F[l_F\,..\,r_F) = P = G[l_G\,..\,r_G)$ satisfying $|l_F - r_G| \leq 2k$.

▶ **Definition 5.7** ([14, Definition 3.9])**.** *For a threshold $k \in \mathbb{Z}_{\geq 0}$ and a weight function w, contexts $P = \langle P_L; P_R \rangle$ and $P' = \langle P'_L; P'_R \rangle$ are called* $\mathsf{ted}^w_{\leq k}$-equivalent *if*

$$\mathsf{ted}^w_{\leq k}(F, G) = \mathsf{ted}^w_{\leq k}(F[0\,..\,l_F) \cdot P'_L \cdot F[l'_F\,..\,r'_F) \cdot P'_R \cdot F[r_F\,..\,|F|), \\ G[0\,..\,l_G) \cdot P'_L \cdot G[l'_G\,..\,r'_G) \cdot P'_R \cdot G[r_G\,..\,|G|))$$

holds for all forests F and G with matching pieces $\langle F[l_F\,..\,l'_F); F[r'_F\,..\,r_F)\rangle = P = \langle G[l_G\,..\,l'_G); G[r'_G\,..\,r_G)\rangle$ satisfying $|l_F - l_G| \leq 2k$ and $|r_F - r_G| \leq 2k$.

In the full version [27], we slightly modify the arguments of [14] to prove the following results:

▶ **Lemma 5.8** (see [14, Lemma 3.17])**.** *There is a linear-time algorithm that, given a forest P and a threshold $k \in \mathbb{Z}_+$, computes a forest P' with $|P'| \leq |P|$ and $|\mathsf{red}_k(P')| \leq 158k^2$ such that P and P' are $\mathsf{ted}^w_{\leq k}$-equivalent for every normalized quasimetric weight function w.*

▶ **Lemma 5.9** (see [14, Lemma 3.18])**.** *There is a linear-time algorithm that, given a context P and a threshold $k \in \mathbb{Z}_+$, computes a context P' with $|P'| \leq |P|$ and $|\mathsf{red}_k(P')| \leq 1152k^3$ such that P and P' are $\mathsf{ted}^w_{\leq k}$-equivalent for every normalized quasimetric weight function w.*

Complete Kernelization Algorithm. In the full version [27], we combine the above ingredients to formally prove Theorem 5.1. Our procedure first applies Theorem 5.3. Then, for every pair of matched pieces $(f, g) \in \mathcal{M}$, we use Lemma 5.8 or 5.9 (depending on piece type) to obtain an equivalent piece P' with $\mathcal{O}(k^3)$ red characters. The remaining (black) characters in P' can be traced back to $\mathcal{O}(k^2)$ periodic blocks and each periodic block within P' yields a free block in the output forest F' that covers the underlying black characters.

6 Summary

▶ **Theorem 1.1.** *There exists a deterministic algorithm that, given two forests F and G with n nodes in total, each with a label from an alphabet Σ, and oracle access to a normalized weight function $w : (\Sigma \cup \{\varepsilon\})^2 \to \mathbb{R}_{\geq 0}$ satisfying the triangle inequality, determines the tree edit distance $k := \mathsf{ted}^w(F, G)$ in $\mathcal{O}(n + k^6 \log k)$ time.*

Proof. First, suppose that the task is to compute $\mathsf{ted}^w_{\leq k}(F, G)$ for a given threshold k. In this case, we use the algorithm of Theorem 5.1, resulting in a pair of forests F', G' of size $\mathcal{O}(k^5)$ such that $\mathsf{ted}^w_{\leq k}(F', G') = \mathsf{ted}^w_{\leq k}(F, G)$, as well as a collection of $\mathcal{O}(k^3)$ free blocks in F' with $\mathcal{O}(k^4)$ non-free characters. Based on this, the algorithm of Theorem 4.2 computes $\mathsf{ted}^w_{\leq k}(F', G') = \mathsf{ted}^w_{\leq k}(F, G)$ in $\mathcal{O}(k^5 \cdot \log k^5 + k^4 \cdot k^2 \log k^5 + k^3 \cdot k^3 \log k^5) = \mathcal{O}(k^6 \log k)$ time. Including the $\mathcal{O}(n)$ running time of Theorem 5.1, we get $\mathcal{O}(n + k^6 \log k)$ total time.

In the absence of a given threshold, we consider a geometric sequence of thresholds $(d_i)_{i \in \mathbb{Z}_{\geq 0}}$, with $d_i = 2^i \cdot \lceil (n/\log n)^{1/6} \rceil$, and we compute $\mathsf{ted}^w_{\leq d_i}(F, G)$ for subsequent $i \in \mathbb{Z}_{\geq 0}$ until $\mathsf{ted}^w_{\leq d_j}(F, G) \leq d_j$ holds for some $j \in \mathbb{Z}_{\geq 0}$, which indicates $\mathsf{ted}^w(F, G) = \mathsf{ted}^w_{\leq d_j}(F, G)$.

Since $d_0 = \mathcal{O}((n/\log n)^{1/6})$, the initial iteration costs $\mathcal{O}(n + d_0^6 \log d_0) = \mathcal{O}(n)$ time. Consequently, if $k \leq d_0$, then the whole algorithm runs in $\mathcal{O}(n)$ time.

Due to $d_i \geq d_0 \geq (n/\log n)^{1/6}$, the running time of the ith iteration iteration is $\mathcal{O}(d_i^6 \log d_i)$. This sequence grows geometrically, so the total running time of the algorithm is dominated by the running time of the last iteration, which is $\mathcal{O}(d_j^6 \log d_j)$. If $k > d_0$, then $j > 0$ and, since the algorithm has not terminated one iteration earlier, $d_j = 2d_{j-1} < 2k$. Consequently, the overall running time is $\mathcal{O}(k^6 \log k)$ when $k > d_0$. ◀

References

1 Shyan Akmal and Ce Jin. Faster algorithms for bounded tree edit distance. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 12:1–12:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.12.

2 Tatsuya Akutsu. Tree edit distance problems: Algorithms and applications to bioinformatics. *IEICE Trans. Inf. Syst.*, 93-D(2):208–218, 2010. doi:10.1587/TRANSINF.E93.D.208.

3 Tatsuya Akutsu, Daiji Fukagawa, and Atsuhiro Takasu. Approximating tree edit distance through string edit distance. *Algorithmica*, 57(2):325–348, 2010. doi:10.1007/s00453-008-9213-z.

4 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

5 John Bellando and Ravi Kothari. Region-based modeling and tree edit distance as a basis for gesture recognition. In *1oth International Conference on Image Analysis and Processing (ICIAP 1999), 27-29 September 1999, Venice, Italy*, pages 698–703. IEEE Computer Society, 1999. doi:10.1109/ICIAP.1999.797676.

6 Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1–3):217–239, June 2005. doi:10.1016/j.tcs.2004.12.030.

7 Mahdi Boroujeni, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. $1+\epsilon$ approximation of tree edit distance in quadratic time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 709–720. ACM, 2019. doi:10.1145/3313276.3316388.

8 Karl Bringmann, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless apsp can). *ACM Trans. Algorithms*, 16(4), July 2020. doi:10.1145/3381878.

9 Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed xml. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 141–152. VLDB Endowment, 2003. doi:10.1016/b978-012722442-8/50021-5.

10 Alejandro Cassis, Tomasz Kociumaka, and Philip Wellnitz. Optimal algorithms for bounded weighted edit distance. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2177–2187. IEEE, 2023. doi:10.1109/FOCS57990.2023.00135.

11 Panagiotis Charalampopoulos, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. On the hardness of computing the edit distance of shallow trees. In Diego Arroyuelo and Barbara Poblete, editors, *String Processing and Information Retrieval - 29th International Symposium, SPIRE 2022, Concepción, Chile, November 8-10, 2022, Proceedings*, volume 13617 of *Lecture Notes in Computer Science*, pages 290–302. Springer, 2022. doi:10.1007/978-3-031-20643-6_21.

12 Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 90–101. Morgan Kaufmann, 1999. URL: http://www.vldb.org/conf/1999/P8.pdf.

13 Gregory Cobena, Serge Abiteboul, and Amélie Marian. Detecting changes in XML documents. In Rakesh Agrawal and Klaus R. Dittrich, editors, *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pages 41–52. IEEE Computer Society, 2002. doi:10.1109/ICDE.2002.994696.

14 Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, and Barna Saha. Weighted edit distance computation: Strings, trees, and dyck. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 377–390. ACM, 2023. doi:10.1145/3564246.3585178.

15 Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, Barna Saha, and Hamed Saleh. $\tilde{O}(n + poly(k))$-time algorithm for bounded tree edit distance. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*. IEEE, 2022. doi:10.1109/FOCS54457.2022.00071.

16 Davi de Castro Reis, Paulo Braz Golgher, Altigran Soares da Silva, and Alberto H. F. Laender. Automatic web news extraction using tree edit distance. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 502–511. ACM, 2004. doi:10.1145/988672.988740.

17 Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, 6(1), December 2010. doi:10.1145/1644015.1644017.

18 Anita Dürr. Improved bounds for rectangular monotone min-plus product and applications. *Inf. Process. Lett.*, 181:106358, 2023. doi:10.1016/J.IPL.2023.106358.

19 Dayi Fan, Rubao Lee, and Xiaodong Zhang. X-TED: massive parallelization of tree edit distance. *Proc. VLDB Endow.*, 17(7):1683–1696, 2024. doi:10.14778/3654621.3654634.

20 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1), November 2009. doi:10.1145/1613676.1613680.

21 Dvir Fried, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, Ely Porat, and Tatiana Starikovskaya. An improved algorithm for the k-dyck edit distance problem. *ACM Trans. Algorithms*, 20(3):26, 2024. doi:10.1145/3627539.

22 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, USA, 1997. doi:10.1017/cbo9780511574931.

23 Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in RNA secondary structures. In *2nd IEEE Computer Society Bioinformatics Conference, CSB 2003, Stanford, CA, USA, August 11-14, 2003*, pages 159–168. IEEE Computer Society, 2003. doi:10.1109/CSB.2003.1227315.

24 Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms*, ESA '98, pages 91–102, Berlin, Heidelberg, 1998. Springer-Verlag. doi:10.1007/3-540-68530-8_8.

25 Philip N. Klein, Thomas B. Sebastian, and Benjamin B. Kimia. Shape matching using edit-distance: an implementation. In S. Rao Kosaraju, editor, *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*, pages 781–790. ACM/SIAM, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365779.

26 Philip N. Klein, Srikanta Tirthapura, Daniel Sharvit, and Benjamin B. Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In David B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pages 696–704. ACM/SIAM, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338628.

27 Tomasz Kociumaka and Ali Shahali. Faster algorithm for bounded tree edit distance in the low-distance regime, 2025. doi:10.48550/arXiv.2507.02701.

28 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

29 Xiao Mao. Breaking the cubic barrier for (unweighted) tree edit distance. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 792–803. IEEE, 2021. doi:10.1109/FOCS52979.2021.00082.

30 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

31 Eugene W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986. doi:10.1007/BF01840446.

32 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3):16:1–16:39, 2014. doi:10.1145/2601073.

33 Jakob Nogler, Adam Polak, Barna Saha, Virginia Vassilevska Williams, Yinzhan Xu, and Christopher Ye. Faster weighted and unweighted tree edit distance and APSP equivalence. In *57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 2167–2178. ACM, 2025. doi:10.1145/3717823.3718116.

34 Mateusz Pawlik and Nikolaus Augsten. Efficient computation of the tree edit distance. *ACM Trans. Database Syst.*, 40(1):3:1–3:40, 2015. doi:10.1145/2699485.

35 Mateusz Pawlik and Nikolaus Augsten. Tree edit distance: Robust and memory-efficient. *Inf. Syst.*, 56:157–173, 2016. doi:10.1016/J.IS.2015.08.004.

36 Thomas B. Sebastian, Philip N. Klein, and Benjamin B. Kimia. Recognition of shapes by editing their shock graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):550–571, 2004. doi:10.1109/TPAMI.2004.1273924.

37 Masoud Seddighin and Saeed Seddighin. $3 + \epsilon$ approximation of tree edit distance in truly subquadratic time. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022*, volume 215, pages 115:1–115:22, 2022. doi:10.4230/LIPIcs.ITCS.2022.115.

38 Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977. doi:10.1016/0020-0190(77)90064-3.

39 Bruce A. Shapiro and Kaizhong Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Comput. Appl. Biosci.*, 6(4):309–318, 1990. doi:10.1093/bioinformatics/6.4.309.

40 Raghavendra Sridharamurthy, Talha Bin Masood, Adhitya Kamakshidasan, and Vijay Natarajan. Edit distance between merge trees. *IEEE Trans. Vis. Comput. Graph.*, 26(3):1518–1531, 2020. doi:10.1109/TVCG.2018.2873612.

41 Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, July 1979. doi:10.1145/322139.322143.

42 Hélène Touzet. A linear tree edit distance algorithm for similar ordered trees. In *Proceedings of the 16th Annual Conference on Combinatorial Pattern Matching*, CPM'05, pages 334–345, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11496656_29.

43 Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1):100–118, 1985. International Conference on Foundations of Computation Theory. doi:10.1016/S0019-9958(85)80046-2.

44 Yuan Wang, David J. DeWitt, and Jin-yi Cai. X-diff: An effective change detection algorithm for XML documents. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 519–530. IEEE Computer Society, 2003. doi:10.1109/ICDE.2003.1260818.

45 Michael S. Waterman. *Introduction to computational biology - maps, sequences, and genomes: interdisciplinary statistics*. CRC Press, 1995.

46 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

47 Yoshiyuki Yamamoto, Kouichi Hirata, and Tetsuji Kuboyama. Tractable and intractable variations of unordered tree edit distance. *Int. J. Found. Comput. Sci.*, 25(3):307–330, 2014. doi:10.1142/S0129054114500154.

48 Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. Answer extraction as sequence tagging with tree edit distance. In Lucy Vanderwende, Hal Daumé III, and Katrin Kirchhoff, editors, *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 858–867. The Association for Computational Linguistics, 2013. URL: https://aclanthology.org/N13-1106/.

49 Kaizhong Zhang and Dennis E. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989. doi:10.1137/0218082.

The Planted Orthogonal Vectors Problem

David Kühnemann
University of Amsterdam, The Netherlands

Adam Polak
Bocconi University, Milan, Italy

Alon Rosen
Bocconi University, Milan, Italy

Abstract

In the k-Orthogonal Vectors (k-OV) problem we are given k sets, each containing n binary vectors of dimension $d = n^{o(1)}$, and our goal is to pick one vector from each set so that at each coordinate at least one vector has a zero. It is a central problem in fine-grained complexity, conjectured to require $n^{k-o(1)}$ time in the worst case.

We propose a way to *plant* a solution among vectors with i.i.d. p-biased entries, for appropriately chosen p, so that the planted solution is the unique one. Our conjecture is that the resulting k-OV instances still require time $n^{k-o(1)}$ to solve, *on average*.

Our planted distribution has the property that any subset of strictly less than k vectors has the *same* marginal distribution as in the model distribution, consisting of i.i.d. p-biased random vectors. We use this property to give average-case search-to-decision reductions for k-OV.

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

Keywords and phrases Average-case complexity, fine-grained complexity, orthogonal vectors

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.95

Related Version *Full Version*: https://arxiv.org/abs/2505.00206

Funding Work supported by European Research Council (ERC) under the EU's Horizon 2020 research and innovation programme (Grant agreement No. 101019547) and Cariplo CRYPTONOMEX grant. Part of this work was done when the first author was visiting Bocconi University.

Acknowledgements We are grateful to Andrej Bogdanov, Antoine Joux, Moni Naor, Nicolas Resch, Nikolaj Schwartzbach, and Prashant Vasudevan for insightful discussions.

1 Introduction

The security of cryptographic systems crucially relies on heuristic assumptions about average-case hardness of certain computational problems. Sustained cryptanalysis alongside technological advances such as large-scale quantum computers, put these hardness assumptions under constant risk of being invalidated. It is therefore desirable to try to design cryptographic schemes based on new computational problems, preferably ones whose hardness is well-studied.

The field of computational complexity developed over the last fifty years a good understanding of the hardness of certain problems – e.g., SAT is widely believed to require at least superpolynomial, maybe even exponential time [28] – however these are worst-case problems, and hence unsuitable for direct use as a basis for cryptography.

Fine-grained complexity [27] is a younger branch of computational complexity that studies "hardness of easy problems", i.e., problems known to be solvable in polynomial time but supposedly not faster than some specified polynomial, say not faster than in cubic time. It gives rise to *fine-grained cryptography* [6, 23, 25], the idea that it might be possible to build


© David Kühnemann, Adam Polak, and Alon Rosen;
licensed under Creative Commons License CC-BY 4.0



cryptography, notably public-key encryption, based on conjectured average-case hardness of polynomial-time problems studied in fine-grained complexity. These problems are easier than NP-hard ones, but for polynomials of sufficiently high degree may still be hard enough to give honest parties an adequate advantage over malicious attackers.

1.1 The k-Orthogonal Vectors Problem

The Orthogonal Vectors (OV) problem [29], together with its generalization k-OV, is one of the three main hard problems studied in fine-grained complexity, alongside 3SUM and APSP [27]. Arguably, among the three, OV is the one whose (worst-case) hardness we understand the most – in particular because it is implied by the Strong Exponential Time Hypothesis (SETH) [16], which is about the very well studied SAT problem.

We say that vectors $u_1, ..., u_k \in \{0,1\}^d$ are *orthogonal* if, for all $j \in [d]$, $\prod_{\ell=1}^{k} u_\ell[j] = 0$, meaning that for every coordinate there is at least one zero entry among the k vectors. For $k \geq 2$, let $U_1, ..., U_k$ each be a collection of n d-dimensional binary vectors, which we view as matrices in $\{0,1\}^{n \times d}$. We denote by $U_{\ell,i}$ the i-th vector of U_ℓ. The k-Orthogonal Vectors problem (k-OV) asks whether there exist $(s_1, \ldots, s_k) \in [n]^k$ such that $U_{1,s_1}, U_{2,s_2}, \ldots, U_{k,s_k}$ are orthogonal.

Worst-case complexity. The naive algorithm solves k-OV in time $O(n^k d)$. For any fixed constant c, the algorithms by Abboud et al. [1] and Chan and Williams [8] solve OV in dimension $d = c \log n$ in time $O(n^{2-\varepsilon_c})$ with $\varepsilon_c > 0$. However, Gao et al. [13] conjecture that no such strongly subquadratic algorithm exists for superlogarithmic dimension $d = \omega(\log n)$. This conjecture (known as Low-dimension Orthogonal Vector Conjecture) is also implied by SETH [29]. Both the upper bound for $d = O(\log n)$ and the SETH-implied hardness for $d = \omega(\log n)$ generalize to k-OV, for any constant $k \geq 2$, where the running time barrier is n^k [27].

Average-case complexity. For cryptographic purposes we care about *average-case* hardness – because we want to be able to efficiently sample instances that are hard to solve (in contrast to only having an existential knowledge that there are some hard instances). Moreover, the sampler shall correctly tell (with good probability) whether its output is a YES- or NO-instance.

One way to achieve this is to embed a solution in an instance sampled from a distribution that generates NO-instances with good probability. This method of *planting* a solution has been applied to a number of problems, e.g., k-Clique [18] and (in the fine-grained setting) k-SUM [12, 2] (a generalization of 3SUM) and Zero-k-Clique [23] (a generalization of Zero-Triangle, which is a problem harder than both 3SUM and APSP), but not for (k-)OV. The following question remains wide-open [6, 11, 10]:

How to plant orthogonal vectors (so that they are hard to find)?

1.2 Our results

We propose a way of planting a solution in k-OV instances where each vector entry is i.i.d. according to a p-biased[1] coin flip, for an appropriately chosen value of p so that the planted solution is the only one in the instance, with good probability. We conjecture that solving these instances requires $n^{k-o(1)}$ time on average.

[1] We say that a random bit is p-biased if it equals 1 with probability p and equals 0 with probability $1-p$.

Let us remark that all our results are already nontrivial for $k = 2$, i.e., for the Orthogonal Vectors problem. However, from the point of view of cryptographic applications, larger values of k are more interesting (as they potentially offer a bigger advantage for the honest parties), so we present all our results in full generality.

Superlogarithmic dimension. The k-OV problem might have appeared as a poor candidate for a fine-grained average-case hard problem, as Kane and Williams [19] showed that for any fixed $p \in (0, 1)$, k-OV instances of i.i.d. p-biased entries can be solved in $O(n^{k-\varepsilon_p})$ time for some $\varepsilon_p > 0$ by AC0 circuits. However, such instances are only nontrivial for $d = \Theta(\log n)$,[2] a parameter setting which can be anyway solved in time $O(n^{k-\varepsilon})$, even in the worst case, using the algorithm of Chan and Williams [8]. To obtain a candidate hard distribution based on i.i.d. entries, we therefore choose to sample the entries as 1 with subconstant probability $p(n) = o(1)$, which leads to nontrivial instances in the superlogarithmic dimension regime $d = \omega(\log n)$. In the full version of the paper we present another simple argument why a logarithmic dimension is not sufficient, further justifying our choice.

The $(k-1)$-wise independence. Our planting procedure has the following notable property: any subset of $k - 1$ (or less) out of the k vectors that form the planted solution has the marginal distribution identical to that of $k - 1$ independently sampled vectors with i.i.d. p-biased random entries. In particular, each individual vector of the solution has the same marginal distribution as any other vector in the instance. This would not be true if we planted k random vectors conditioned on orthogonality (i.e., a type of solution that may appear spontaneously with small probability), because such vectors tend to be sparser than the expectation. This sparsity is what makes the Kane–Williams algorithm [19] work, and lack thereof makes our instances immune to that algorithmic idea.[3]

We note that the $(k - 1)$-wise independence property holds "for free" in natural distributions for k-SUM [2, 12] and Zero-k-Clique [23] because of the natural symmetry of cyclic groups $\mathbb{Z}_m$. However, it is a priori unclear how to get it for k-OV.

Finally, in Theorem 7, we argue that our distribution is *the unique* distribution over k-OV instances that has this property, explaining the title of this paper.

Search-to-decision reductions. To demonstrate the usefulness of the $(k - 1)$-wise independence property, we give a fine-grained average-case search-to-decision reduction for our conjectured hard k-OV distribution. Actually, we give two such reductions. The first one, in Section 6, is very simple, but it introduces an $O(\log n)$ overhead in the failure probability, so it is relevant only if the decision algorithm succeeds with probability higher than $1 - \frac{1}{\log n}$. The other reduction, in Section 7, looses only a constant factor in the failure probability. Even though we present both reductions specifically for k-OV, they are likely to generalize to any planted problem with the $(k - 1)$-wise independence property.

Planting multiple solutions. In the full version of the paper, we also argue that $(k-1)$-wise independence allow planting more than one solution in a single instance, which we believe might be useful for building cryptographic primitives.

[2] For larger (resp. smaller) d, almost all instances will be NO-instances (resp. YES-instances).

[3] Note though that the Kane–Williams algorithm does not run in truly subquadratic time in superlogarithmic dimension anyway, so above all it is the high dimension, not the $(k - 1)$-wise independence, that makes our distribution immune to all known attacks.

1.3 Technical overview

Planting. How do we generate k orthogonal vectors such that any $k-1$ of them look innocent? First of all, we can focus on generating a single coordinate, and then repeat the process independently for each of the d coordinates. Consider the joint distribution of k i.i.d. p-biased random bits. We need to modify it to set the probability of k ones to 0. If we just do it, and scale up the remaining probabilities accordingly, the probability of $k-1$ ones turns out wrong. After we fix that, the probability of $k-2$ ones is off, and so on, in a manner similar to the inclusion-exclusion principle. By doing this mental exercise we end up with a formula for the joint distribution of k bits in a single coordinate of the k vectors to be planted. How do we actually sample from this distribution? Since it has the $(k-1)$-wise independence property, the following approach must work: First sample $k-1$ i.i.d. p-biased bits, and then sample the k-th bit with probability depending on the number of ones among the first $k-1$ bits. In Section 3 we show how to set this last probability exactly.

Search-to-decision reductions. Both our reductions are based on the same basic idea: In order to find the planted solution, we replace some of the vectors in the input instance with newly sampled vectors with i.i.d. p-biased entries and run the decision algorithm on such a modified instance. If at least one of the planted vectors got resampled, the resulting instance has the same distribution as if no planting occurred (thanks to the $(k-1)$-wise independence), and so the decision algorithm returns NO with good probability. Otherwise the planted solution is still there and the decision algorithm likely says YES.

Our first reduction (see Section 6) applies this idea to perform a binary search. It introduces a factor of $k \log n$ overhead in the running time and also in the failure probability, because we need to take a union bound over all invocations of the decision algorithm returning correct answers.

Our second reduction (see Section 7) is an adaptation of a search-to-decision reduction for k-SUM due to Agrawal et al. [2]. In short, the reduction repeatedly resamples a random subset of vectors, runs the decision algorithm, and keeps track for each of the original vectors, how many times the decision algorithm returned YES when this vector was not resampled. Statistically, this count should be larger for vectors in the planted solution. A careful probabilistic analysis shows that this is indeed the case.

1.4 Open problems

Hardness self-amplification. Could a black-box method increase the success probability of algorithms solving (the search or decision variants of) the planted k-OV problem, at a small cost in the running time? If so, the lack of algorithms with high success probability for planted k-OV would then suggest that no algorithm can solve the problem even with just a small success probability – a property desirable, e.g., from the point of view of potential cryptographic applications.

Such hardness self-amplification was recently shown for (both the search and decision variants of) the planted clique problem by Hirahara and Shimizu [15]. In the world of fine-grained complexity, Agrawal et al. [2] showed hardness self-amplification for the planted k-SUM search problem and closely related problems. Hardness self-amplification for planted k-OV remains an open problem.

Because of the overhead in the failure probability induced by both of our search-to-decision reductions, hardness self-amplification for the decision variant of the planted k-OV problem in particular would mean that the hardness of the search problem could be based on a weak conjecture about the hardness of the decision problem, such as Conjecture 10.

Fine-grained asymmetric cryptography. A key goal of fine-grained cryptography is to devise an advanced asymmetric cryptography scheme – such as public key encryption – whose security is based on hardness of a well understood problem from fine-grained complexity. So far the closest to this goal seems to be the key exchange protocol due to LaVigne, Lincoln, and Vassilevska Williams [23], which is based on hardness of the planted Zero-k-Clique problem. Despite being based on a parameterized problem (that allows for arbitrary polynomial $n^{k-o(1)}$-hardness by simply choosing a large enough k), the protocol offers only quadratic security, i.e., breaking the encryption takes only quadratically more than it takes to encrypt and decrypt a message. This limitation seems inherent to the protocol because it is based on a similar idea as Merkle puzzles [24].

It is an open problem if fine-grained cryptography with superquadratic security is possible. We believe that k-OV could be a good hard problem for that purpose, because of a different structure, which addition-based problems, like k-SUM and Zero-k-Clique, are lacking.

We remark that the key exchange protocol of LaVigne, Lincoln, and Vassilevska Williams [23] can be adapted to work with our planted k-OV instead of the planted Zero-k-Clique problem, but naturally the protocol's security remains quadratic. One needs new techniques to break the quadratic barrier.

In recent work, Alman, Huang, and Yeo [5] show that if one-way functions do not exist then average-case hardness fine-grained assumptions on planted k-SUM and Zero-k-Clique are false for sufficiently large constant k. It might be possible to generalize their results to k-OV. However, a construction of public-key encryption from planted k-OV would be interesting even in a world where one-way functions do exist, as they are not known to imply superquadratic-gap public-key encryption [17].

Faster algorithms for average-case OV. Algorithms for random OV instances seem to be underexplored. Up until recently [4] it was not known if the average-case OV admits even a subpolynomial improvement compared to the worst case. With this paper we hope to inspire more research in this direction. We would even be happy to see our conjecture refuted.

A natural starting point for such an attack is the recent Alman–Andoni–Zhang algorithm [4] for random OV. It works in time $n^{2-\Omega(\log\log c/\log c)}$ for dimension $d = c\log n$, so it is not truly subquadratic for $d = \omega(\log n)$. However, in their setting, the hardest case is when the probability p of a one entry is chosen to make the *expected number of orthogonal pairs* a constant, while in our setting we use a higher value of p to lower the expectation to inverse polynomial – which means that in our setting the orthogonal pair "stands out more". It might seem plausible to adjust internal parameters of the algorithm (in particular, the so-called *group size*) to better exploit our setting. However, under closer inspection, it turns out that the key quantity in the analysis of the algorithm is the *expected inner product* of two random vectors, equal to $p^2 d$, which happens to be $\Theta(\log n)$ in both settings. Refuting our conjecture likely requires a new technical development beyond such an adjustment.

Finally, let us point out a related problem: the planted approximate maximum inner product problem, often referred to as the *light bulb* problem. Unlike planted OV, it is known to admit truly subquadratic $O(n^{1.582})$-time algorithms [26, 20, 21, 3]. This is in contrast with the worst-case complexities of the two problems, which are known to be equivalent under fine-grained reductions [9, 22].

A worst-case to average-case reduction. In the opposite direction than the previous open problem, one could try to show that our conjectured hardness of the planted k-OV problem is implied by one of well-studied worst-case hypotheses in fine-grained complexity, e.g., SETH.

This would require a worst-case to average-case reduction. So far, in fine-grained complexity, such reductions are only known for algebraic or counting problems [6, 14, 7, 11, 10], but not for decision nor search problems like ours.

2 The model distribution

Fix $k \geq 2$, and let $d = \alpha(n) \log n$ for $\alpha(n) = \omega(1)$. We define the family of *model distributions* $k{-}\mathrm{OV}_0^\alpha(n)$ that generate k matrices $U_1, ..., U_k \in \{0,1\}^{n \times d}$ where all entries are i.i.d. p-biased bits with probability

$$p = \left(1 - 2^{-\frac{2k}{\alpha(n)}}\right)^{\frac{1}{k}} \approx \left(\frac{2k\ln(2)}{\alpha(n)}\right)^{\frac{1}{k}}.$$

As will become apparent later, for the planting algorithm to work it is crucial that $p \leq 1/2$, but thanks to $\alpha(n) = \omega(1)$ this holds for large enough n.

We show that the model distribution indeed generates NO-solutions with good probability.

▶ **Lemma 1.** *A k-OV instance sampled from the model distribution $k{-}\mathrm{OV}_0^\alpha(n)$ is a NO-instance with probability at least $1 - \frac{1}{n^k}$.*

Proof. For a k-OV instance $\mathbf{U} = (U_1, \ldots, U_k) \sim k{-}\mathrm{OV}_0^\alpha(n)$, a fixed combination of vectors $u_1, \ldots, u_k$ (where $u_\ell \in U_\ell$) is orthogonal iff, for each coordinate $j \in [d]$, not all of the k vectors feature a one in that coordinate. Since k i.i.d. p-biased bits are all ones with probability p^k, the probability that $u_1, \ldots, u_k$ are orthogonal (determined by the all-ones event not occurring in any of the d coordinates) is:

$$\Pr[u_1, ..., u_k \text{ are orthogonal}] = \left(1 - p^k\right)^d = \left(2^{-\frac{2k}{\alpha(n)}}\right)^{\alpha(n)\log(n)} = n^{-2k}.$$

By linearity of expectation, the expected value for the number of solutions among all n^k possible combinations of k vectors, denoted by $c(\mathbf{U})$, is

$$\mathbb{E}[c(\mathbf{U})] = \sum_{\substack{u_i \in U_i \\ (1 \leq i \leq k)}} \Pr[u_1, \ldots, u_k \text{ are orthogonal}] = n^k \cdot n^{-2k} = \frac{1}{n^k}.$$

By Markov's inequality, this is also a bound on the probability of *any* solution occurring, i.e., $\Pr[c(\mathbf{U}) \geq 1] \leq \mathbb{E}[c(\mathbf{U})] = \frac{1}{n^k}$. Therefore, an instance sampled from $k{-}\mathrm{OV}_0^\alpha(n)$ is a NO-instance with probability at least $1 - \frac{1}{n^k}$. ◀

We remark that one can make the probability of sampling a NO-instance arbitrarily high. Indeed, in order to get the probability $1 - \frac{1}{n^c}$ it suffices to replace $2k$ with $k + c$ in the formula for the probability parameter p. However, having in mind the cryptographic motivation, $\frac{1}{n^k}$ seems to be a reasonable default choice for the failure probability of the sampler, because with the same probability the attacker can just guess the solution.

3 The planted distribution

To plant a solution at locations $s_1, \ldots, s_k \in [n]$ in an instance $\mathbf{U}$ sampled from $k{-}\mathrm{OV}_0^\alpha(n)$, we apply the following randomized algorithm.

Plant$(\mathbf{U}, s_1, \ldots, s_k)$.

1. For each coordinate $1 \leq j \leq d$:
   a. Let m be the number of ones among $U_{1,s_1}[j], \ldots, U_{k,s_k}[j]$.
   b. If $k - m$ is even, flip $U_{k,s_k}[j]$ with probability $\left(\frac{p}{1-p}\right)^{k-m}$. (Here we need $p \leq 1/2$.)
2. Return $\mathbf{U}$.

We justify this way of planting in Section 4. For now, observe that if all vectors $U_{1,s_1}, \ldots, U_{k,s_k}$ feature a one at coordinate j, we have $m = k$ and Plant flips the final bit $U_{k,s_k}[j]$ to a zero with probability

$$\left(\frac{p}{1-p}\right)^{k-m} = \left(\frac{p}{1-p}\right)^{0} = 1.$$

On the other hand, if the last coordinate is the single zero alongside $m = k - 1$ ones, then $k - m = 1$ is odd and Plant will never break orthogonality by flipping the last bit to a one. Thus, Plant$(\mathbf{U}, s_1, \ldots, s_k)$ outputs a YES-instance of k-OV with a solution at $s_1, \ldots, s_k$. We call the k vectors at these positions the *planted vectors*.

We sample YES-instances of k-OV by planting a solution in an instance $\mathbf{U} \sim k\text{−OV}_0^\alpha(n)$ at locations $s_1, \ldots, s_k$ chosen uniformly at random.

Distribution $k\text{−OV}_1^\alpha(n)$.

1. Sample $\mathbf{U}$ from $k\text{−OV}_0^\alpha(n)$.
2. Sample $(s_1, \ldots, s_k)$ uniformly at random from $[n]^k$.
3. Return Plant$(\mathbf{U}, s_1, \ldots, s_k)$.

The above observation about Plant immediately yields the following.

▶ **Lemma 2.** *A k-OV instance sampled from the planted distribution $k\text{−OV}_1^\alpha(n)$ is a YES-instance with probability* 1.

4 The $(k-1)$-wise independence of planted vectors

Our method of planting orthogonal vectors arises from the idea that for any planted problem, any proper "piece" of the planted solution should be indistinguishable from any comparable piece of the instance as a whole, conditioned on the latter still being consistent with being a part of a solution itself.

For example, in the case of planting a k-clique in a graph G this requirement is trivial. Indeed, the projection of the clique onto a smaller subset of $k' < k$ vertices yields a k'-clique, which are exactly those subgraphs of G of size k' which could feasibly belong to a solution.

In contrast to the previous example, in the case of k-SUM, any set of $k - 1$ elements $x_1, \ldots, x_{k-1}$ in an instance could feasibly be part of a solution, as one can always construct a k−th number x_k such that $\sum_{i=1}^{k} x_i = 0$. Thus, by the principle we described, to plant a solution in an instance with i.i.d. uniformly random elements, the marginal distribution of the distribution of planted solutions $(x_1, \ldots, x_k)$ given by any projection to $k - 1$ elements should itself be uniformly random. This holds true in the case of the planted k-SUM [2], where the planted solution is distributed uniformly over the set of all k-tuples that form valid k-SUM solutions. The case of planted Zero-k-Clique [23] is analogous. For both of these problems, planting by inserting k elements drawn from the model distribution conditioned on them forming a solution yields a distribution that follows the described principle.

This is different from the k-OV problem with a model distribution of i.i.d. vector entries. Here, as with k-SUM and Zero-k-Clique, any set of $k-1$ elements (in this case vectors) could form a solution to k-OV. All that is needed is for the last vector to feature a zero in all those coordinates where the other $k-1$ all were one. However, sparse vectors are far more likely to be part of a solutions than dense ones. Therefore, conditioning k i.i.d. p-biased vectors on being orthogonal yields a distribution which does not follow our principle: projecting onto any subset of $k' < k$ vectors results in vectors that are on average sparser than (and thus different from) k' i.i.d. p-biased vectors. As we will show now, our method of planting *does* satisfy this principle: Any subset of $k-1$ planted vectors are independent and identically distributed p-biased vectors.

Let $M \sim k\text{–}\mathrm{OV}_0^\alpha(n)$ and $\mathbf{U} = \mathsf{Plant}(M, s_1, \ldots, s_k)$. Recall that both sampling from the model distribution $k\text{–}\mathrm{OV}_0^\alpha(n)$ and the planting by Plant are independent and identical for each coordinate $j \in [d]$. Hence, all k-bit sequences $x = (U_{1,s_1}[j], U_{2,s_2}[j], \ldots, U_{k,s_k}[j]) \in \{0,1\}^k$, for all $j \in [d]$, are independent and identically distributed, according to a distribution whose probability density function we denote by $\mathcal{P}_k : \{0,1\}^k \to \mathbb{R}$.

▶ **Lemma 3.** *Let $x \in \{0,1\}^k$ and let m be the number of ones in x. Then*

$$\mathcal{P}_k(x) = p^m(1-p)^{k-m} - (-1)^{k-m}p^k.$$

Proof. Fix a coordinate $j \in [d]$. Let $X = (M_{1,s_1}[j], M_{2,s_2}[j], \ldots, M_{k,s_k}[j])$ be the random variable denoting the entries of the j-th coordinate among the vectors at locations $s_1, \ldots, s_k$ before planting. We proceed by case distinction.

Case 1. If $m-k$ is even, the probability of x occurring in the given coordinate $j \in [d]$ of the planted solution is given by

$$\begin{aligned}
\mathcal{P}_k(x) &= \Pr\left[X = x \text{ and } \mathsf{Plant} \text{ does not flip the final bit}\right] \\
&= p^m(1-p)^{k-m} \cdot \left(1 - \left(\frac{p}{1-p}\right)^{k-m}\right) \\
&= p^m(1-p)^{k-m} - 1 \cdot p^k \\
&= p^m(1-p)^{k-m} - (-1)^{k-m}p^k.
\end{aligned}$$

Case 2a. If $m-k$ is odd and $x = y1$ for some $y \in \{0,1\}^{k-1}$, then $x = y1$ may occur either directly in the model instance, or by $y0$ (for which $m-k$ is even) occurring in the model instance and Plant flipping the final bit:

$$\begin{aligned}
\mathcal{P}_k(x) &= \Pr\left[X = y1\right] + \Pr\left[X = y0 \text{ and } \mathsf{Plant} \text{ flips the final bit}\right] \\
&= p^m(1-p)^{k-m} + p^{m-1}(1-p)^{k-(m-1)} \cdot \left(\frac{p}{1-p}\right)^{k-(m-1)} \\
&= p^m(1-p)^{k-m} + p^k \\
&= p^m(1-p)^{k-m} - (-1)^{k-m}p^k.
\end{aligned}$$

Case 2b. Similarly, if $m-k$ is odd and $x = y0$ for some $y \in \{0,1\}^{k-1}$, then $x = y0$ can occur either directly in the model instance or by Plant flipping the final bit of the sequence $y1$:

$$\begin{aligned}\mathcal{P}_k(x) &= \Pr[X = y0] + \Pr[X = y1 \text{ and } \mathsf{Plant} \text{ flips the final bit}] \\ &= p^m(1-p)^{k-m} + p^{m+1}(1-p)^{k-(m+1)} \cdot \left(\frac{p}{1-p}\right)^{k-(m+1)} \\ &= p^m(1-p)^{k-m} + p^k \\ &= p^m(1-p)^{k-m} - (-1)^{m-k}p^k.\end{aligned}$$

◀

▶ Remark 4. Despite Plant acting only on the last collection U_k, Lemma 3 implies that the resulting distribution $k{-}\mathrm{OV}_1^\alpha(n)$ is invariant under permutation of the sequence of the k collections $U_1, \ldots, U_k$.

Having $\mathcal{P}_k$ as the distribution of planted vectors, rather than, e.g., the k-vector joint model distribution conditioned on orthogonality, ensures $(k-1)$-*wise independence* among the planted vectors. I.e., the projection of k planted vectors onto any subset of size $k' < k$ is identically distributed to k' vectors from the model distribution.

▶ **Lemma 5** ($(k-1)$-wise independence)**.** *Marginalizing any one of the k bits of $\mathcal{P}_k$ yields $k-1$ independent p-biased bits.*

Proof. By Remark 4 we may assume w.l.o.g. that the last bit is the one marginalized out. The lemma then follows from the definition of Plant, as the first $k-1$ entries of any coordinate in the planted vectors are unchanged from the model instance, and are therefore independent p-biased bits. ◀

This property is useful in bounding the probability of a planted instance containing a solution besides the planted one.

▶ **Lemma 6.** *A k-OV instance sampled from the planted distribution $k{-}\mathrm{OV}_1^\alpha(n)$ has more than one solution with probability less than $\frac{1}{n^k}$.*

Proof. While the k vectors at positions $s_1, \ldots, s_n$ are guaranteed to form a solution, by $(k-1)$-wise independence, *all* combinations of $0 \le k' < k$ of these vectors and $k-k'$ non-planted vectors form a set of k independent p-biased vectors which is therefore a solution to the k-OV problem with probability $(1-p^k)^d = \frac{1}{n^{2k}}$. By linearity of expectation,

$$\mathbb{E}[c(\mathbf{U})] = 1 + (1-p^k)^d \cdot (n^k - 1) < 1 + \frac{1}{n^k},$$

and the claim follows from Markov's inequality. ◀

4.1 Uniqueness

Our way of planting is unique in the following sense.

▶ **Theorem 7.** *Let $Q : \{0,1\}^k \to \mathbb{R}$ be a probability distribution such that $Q(1^k) = 0$ and that marginalizing any one of the k bits yields $k-1$ independent p-biased bits. Then $Q = \mathcal{P}_k$.*

Proof. We show that $Q(x) = \mathcal{P}_k(x)$ for all $x \in \{0,1\}^k$. Let m denote the number of ones in x. We proceed by induction over $k-m$, i.e., the number of *zeros* in x.

Base case: $k-m=0$. Then $m=k$ and $x=1^k$. Thus $Q(x) = Q(1^k) = 0 = \mathcal{P}_k(x)$.

Inductive case: $k - m > 0$. We assume w.l.o.g. that the $k - m$ zeros are the *last* bits of x, i.e., $x = 1^m 0^{k-m}$. Marginalizing the final $k - m > 0$ bits of Q yields $k - (k - m) = m$ independent p-biased bits, whereby the probability of all m remaining bits being ones is

$$p^m = \sum_{y \in \{0,1\}^{k-m}} Q(1^m y) = Q(\underbrace{1^m 0^{m-k}}_{=x}) + \sum_{\substack{y \in \{0,1\}^{k-m} \\ y \neq 0^{k-m}}} Q(1^m y).$$

Thereby,

$$\begin{aligned} Q(x) &= p^m - \sum_{\substack{y \in \{0,1\}^{k-m} \\ y \neq 0^{k-m}}} Q(1^m y) \\ &= p^m - \sum_{\substack{y \in \{0,1\}^{k-m} \\ y \neq 0^{k-m}}} \mathcal{P}_k(1^m y) && \text{(by the induction hypothesis)} \\ &= p^m + \mathcal{P}_k(x) - \sum_{y \in \{0,1\}^{k-m}} \mathcal{P}_k(1^m y) \end{aligned}$$

where the sum term is merely the probability of 1^m in the marginal distribution of $\mathcal{P}_k$, which by Lemma 5 in turn consists of m independent p-biased bits. Hence,

$$= p^m + \mathcal{P}_k(x) - p^m = \mathcal{P}_k(x). \qquad \blacktriangleleft$$

5 Conjectured hard problems

In this section we formally define the problems that we conjecture to require $n^{k-o(1)}$ time.

▶ **Definition 8** (Solving planted **decision** k-OV)**.** *Let $\mathcal{A}$ be an algorithm that given a k-OV instance $\mathbf{U}$ outputs either 0 or 1. For $\alpha(n) = \Omega(1)$, we say $\mathcal{A}$* solves *the* decision k-OV^α *problem with success probability $\delta(n)$, if for both $b \in \{0,1\}$ and large enough n,*

$$\Pr_{\mathbf{U} \sim k\text{-}OV_b^\alpha(n)}[\mathcal{A}(\mathbf{U}) = b] \geq \delta(n),$$

where randomness is taken over both the instance $\mathbf{U}$ and the random coins used by $\mathcal{A}$.

Similarly, we define a notion of recovering a solution from a planted instance.

▶ **Definition 9** (Solving planted **search** k-OV)**.** *Let $\mathcal{A}$ be an algorithm that given a k-OV instance $\mathbf{U}$ outputs a tuple $(s_1, ..., s_k) \in \{1, ..., n\}^k$. For a given $\alpha(n) = \Omega(1)$, we say $\mathcal{A}$* solves *the* planted search k-OV^α problem *with success probability $\delta(n)$ if for large enough n,*

$$\Pr_{\substack{\mathbf{U} \sim k-\mathrm{OV}_1^\alpha(n) \\ (s_1,...,s_k) \leftarrow \mathcal{A}(\mathbf{U})}}[U_{1,s_1}, ..., U_{k,s_k} \text{ are orthogonal}] \geq \delta(n),$$

where randomness is taken over both the instance $\mathbf{U}$ and the random coins used by $\mathcal{A}$.

Now we are ready to formally state our main conjecture.

▶ **Conjecture 10.** *For any $\alpha(n) = \omega(1)$ and $\varepsilon > 0$, there exists no algorithm $\mathcal{A}$ that solves the planted decision k-OV problem with any constant success probability $\delta > \frac{1}{2}$ in time $O(n^{k-\varepsilon})$.*

6 Search-to-decision reduction via binary search

We reduce the search problem of finding the planted solution to the decision problem of determining whether an instance contains a planted solution. This means that given a decision algorithm that can correctly distinguish whether an instance was sampled from the model or planted distribution with sufficient probability, one can recover the planted secret through this reduction. The reduction introduces a factor $O(\log n)$ increase in both the running time and error probability of the algorithm.

The idea is to find each planted vector using something akin to binary search on each collection U_i. We can split U_i into two partitions of roughly equal size and run the decision algorithm twice, on instances where one of the two partitions is first replaced by newly sampled p-biased vectors. The vector planted in U_i is guaranteed to be replaced in one of these cases, and by $(k-1)$-wise independence the resulting instance follows the model distribution. The search space is thus cut in half and we can recurse on this smaller search space to eventually find the planted vector.

▶ **Theorem 11** (Search-to-decision reduction). *Let $\alpha(n) = \mathrm{polylog}(n)$ and let $\mathcal{A}^{\mathrm{decide}}$ be an algorithm that solves the planted decision k-OV problem with success probability $1-\delta(n)$ in time $T(n)$. Then there exists an algorithm $\mathcal{A}^{\mathrm{search}}$ that solves the planted search k-OV problem with success probability at least $1-k\lceil\log n\rceil\cdot\delta(n)$ in expected time $\widetilde{O}(T(n)+n)$.*

Proof. Consider an instance $\mathbf{U} = (U_1,\ldots,U_k) \sim k{-}\mathrm{OV}_1^\alpha(n)$. First let us focus only on recovering the location $i\in[n]$ of the planted vector in the first collection U_1. The reduction begins with the "full" search space $S := [n]$, and narrows it down by half in each iteration, so that the desired i is recovered after $\lceil\log n\rceil$ iterations.

At each iteration, the current search space S is arbitrarily partitioned in two sets of equal size (up to one vector when $|S|$ is odd). The decision algorithm $\mathcal{A}^{\mathrm{decide}}$ is then executed on two new instances, where the respective sets of vectors in U_1 are replaced with newly sampled p-biased vectors.

By the $(k-1)$-wise independence, if the vector belonging to the solution is replaced, all vectors are independently and identically distributed p-biased vectors, i.e., the instance is distributed according to $k{-}\mathrm{OV}_0^\alpha(n)$. On the other hand, if the solution survives resampling, the instance remains distributed according to $k{-}\mathrm{OV}_1^\alpha(n)$. Therefore, the output of $\mathcal{A}^{\mathrm{decide}}$ is used to decide which of the two partition blocks should be assumed as the new search space.

The reduction is correct if $\mathcal{A}^{\mathrm{decide}}$ decides correctly at every iteration. Of course, $\mathcal{A}^{\mathrm{decide}}$ might fail with probability $\delta(n)$. By a union bound over all $\lceil\log n\rceil$ invocations of $\mathcal{A}^{\mathrm{decide}}$ this happens with probability at most $\lceil\log n\rceil\cdot\delta(n)$. Thereby $\mathcal{A}^{\mathrm{search}}$ recovers the location i of the first planted vector with success probability at least $1-\lceil\log n\rceil\cdot\delta(n)$.

As for the runtime, $\mathcal{A}^{\mathrm{decide}}$ with runtime $T(n)$ is invoked $O(\log n)$ times, and across all iterations $n-1$ vectors are resampled in total. Since a single p-biased bit can be sampled in expected time $O(-\log p) = O(\log\alpha(n)) = O(\log\log n)$, sampling a d-dimensional vector takes $\mathrm{polylog}(n)$ time in expectation. Therefore, recovering the location of the first planted vector takes time $\widetilde{O}(T(n)+n)$.

The same process is repeated another $k-1$ times to recover the locations of the planted vectors among $U_2,\ldots,U_k$. As k is constant, this does not increase the running time asymptotically but the success probability drops to $1-k\lceil\log n\rceil\cdot\delta(n)$. ◀

7 Search-to-decision reduction via counters

We present a second search-to-decision reduction, adapted from that of Agrawal et al. [2] for planted k-SUM. As in the method in Section 6, we use the fact that an algorithm $\mathcal{A}^{\text{decide}}$ for the decision k-OV problem, when given a planted k-OV instance with some of the vectors resampled, correctly detects whether any of the planted vectors were among the resampled vectors. However, instead of iteratively narrowing a pool of candidate vectors, we iterate this process on the entire instance, and, for each vector u, we keep count of the number of iterations in which u survived and $\mathcal{A}^{\text{decide}}$'s output was 1. After $O(\log(n))$ iterations we output the vectors with the highest counts among each of the k collections, which, as we show, coincides with the planted solution (with good probability).

▶ **Theorem 12** (Search-to-decision reduction). *For any $\alpha(n) = \text{polylog}(n)$, if there exists an algorithm that solves the planted decision k-OV problem with success probability at least $1 - \delta(n)$ in time $T(n)$, then there exists an algorithm that solves the planted search k-OV problem with success probability at least $1 - 13k \cdot \delta(n) - \frac{1}{n^k}$ in expected time $O\left(\left(T(n) + n\,\text{polylog}(n)\right)\log\frac{n}{\delta(n)}\right)$.*

In more detail, let Mix be the following randomized algorithm, which takes a k-OV instance $\mathbf{U}$ and resamples some of the vectors:

Algorithm Mix($\mathbf{U}$).

1. For each $\ell \in [k]$ and $i \in [n]$:
   a. With probability $1 - 2^{-\frac{1}{k}}$, replace $U_{\ell,i}$ by a newly-sampled p-biased vector
2. Output $\mathbf{U}$

For $\ell \in [k]$, let $R_\ell \subseteq [n]$ indicate the indices of the vectors of U_ℓ which are replaced by Mix. For a vector u in $\mathbf{U}$ and a given execution of Mix, we say u *survives* if Mix does not replace u. We say $\mathbf{s} = (s_1, \ldots, s_k)$ *survives* if U_{ℓ,s_ℓ} survives for each $\ell \in [k]$, i.e., $\forall_{\ell\in[k]} s_\ell \notin R_\ell$.

Now, let $B(\mathbf{s})$ be the binary random variable indicating whether $\mathbf{s}$ survives. Our chosen probability for Mix to resample a vector yields the following.

▶ **Lemma 13.** *For a k-OV instance $\mathbf{U}$ and any $\mathbf{s} \in [n]^k$, $\mathbf{s}$ survives with probability one half, i.e., $\Pr_{\text{Mix}}[B(\mathbf{s}) = 1] = \frac{1}{2}$.*

Proof. Each vector is independently picked to be replaced with probability $1 - 2^{-\frac{1}{k}}$. The chance of all k vectors surviving is

$$\Pr_{\text{Mix}}[B(\mathbf{s}) = 1] = \left(1 - \left(1 - 2^{-\frac{1}{k}}\right)\right)^k = 2^{-\frac{1}{k}\cdot k} = \frac{1}{2}. \qquad \blacktriangleleft$$

As laid out before, our search algorithm repeatedly executes Mix and then the given decision algorithm $\mathcal{A}^{\text{decide}}$. The hope is that the output of the latter correlates with the survival of the planted solution. We therefore also keep track of which vectors survive whenever $\mathcal{A}^{\text{decide}}$ believes there is still a planted solution in the instance output by Mix.

Algorithm $\mathcal{A}^{\text{search}}(\mathbf{U})$.

1. Initialize a counter $C_{\ell,i} := 0$ for each $\ell \in [k]$ and $i \in [n]$.
2. Repeat $m = \Theta(\log n)$ times:
   a. $\mathbf{V} \leftarrow \mathsf{Mix}(\mathbf{U})$
   b. $b := \mathcal{A}^{\text{decide}}(\mathbf{V})$
   c. If $b = 1$:
      i. Set $C_{\ell,i} := C_{\ell,i} + 1$ for every $U_{\ell,i}$ that was not replaced by Mix
3. Set $s_\ell := \arg\max_{i \in [n]} C_{\ell,i}$ for each $\ell \in [k]$
4. Output $\mathbf{s} = (s_1, \ldots, s_k)$

This works well for instances $\mathbf{U}$ where $\mathcal{A}^{\text{decide}}$ is good at detecting whether a particular solution survives. To capture this notion, we say an instance $\mathbf{U}$ is *good*, if it has only one solution, at some location $\mathbf{s}$, *and* the output of $\mathcal{A}^{\text{decide}}$ indicates whether $\mathbf{s}$ survives except for a small constant probability, i.e.,

$$\Pr[\mathcal{A}^{\text{decide}}(\mathsf{Mix}(\mathbf{U})) \neq B(\mathbf{s})] < \frac{1}{12k},$$

where the probability is taken over the internal randomness used by Mix.

In the following, let Sample be a randomized algorithm that outputs a planted instance sampled from $k{-}\mathrm{OV}_1^\alpha(n)$ as well as the location $\mathbf{s}$ of the planted solution.

▶ **Lemma 14.** *If $\mathcal{A}^{\text{decide}}$ solves the planted decision k-OV problem with success probability at least $1 - \delta(n)$, an instance $\mathbf{U} \sim k{-}\mathrm{OV}_1^\alpha(n)$ is* good *except with probability at most $12k \cdot \delta(n) + \frac{1}{n^k}$.*

Proof. An instance $\mathbf{U} \sim k{-}\mathrm{OV}_1^\alpha(n)$ contains only a single solution except with probability less than $\frac{1}{n^k}$. We will now show that for such $\mathbf{U}$ with only the planted solution, the decision algorithm $\mathcal{A}^{\text{decide}}$ correctly detects if this solution survives except with probability $< 12k \cdot \delta(n)$, from which the claim follows by a union bound.

First, consider the following distribution:

Distribution $k{-}\mathrm{OV}_B^\alpha(n)$.

1. $(\mathbf{U}, \mathbf{s}) \leftarrow \mathsf{Sample}$
2. $\mathbf{V} \leftarrow \mathsf{Mix}(\mathbf{U})$
3. Output $(\mathbf{V}, B(\mathbf{s}))$.

Observe that, if we condition on $B(\mathbf{s}) = 1$, the planted solution survives Mix, which merely resamples some of the i.i.d. p-biased vectors in the instance. Thus, $\mathbf{V}$ is distributed according to $k{-}\mathrm{OV}_1^\alpha(n)$. On the other hand, if $B(\mathbf{s}) = 0$, at least one of the planted vectors is replaced by a newly-sampled p-biased vector. By $(k-1)$-wise independence, the subset of planted vectors that survive are i.i.d. p-biased vectors, as are all other vectors in $\mathbf{V}$. Hence, conditioning on $B(\mathbf{s}) = 0$ yields the model distribution $k{-}\mathrm{OV}_0^\alpha(n)$.

Therefore, for the algorithm $\mathcal{A}^{\text{decide}}$, which solves the planted decision k-OV problem with success probability at least $1 - \delta(n)$, we have

$$\Pr_{(\mathbf{V},B) \leftarrow k{-}\mathrm{OV}_B^\alpha(n)}[\mathcal{A}^{\text{decide}}(\mathbf{V}) \neq B] \leq \delta(n).$$

Now, let $Z(\mathbf{U}, \mathbf{s})$ be the random variable that, for a given instance $\mathbf{U}$ with a solution planted at $\mathbf{s}$, denotes the probability of $\mathcal{A}^{\text{decide}}(\mathsf{Mix}(U)) \neq B(\mathbf{s})$, where the randomness is taken over the internal coins used by Mix. Then

$$\begin{aligned}\delta(n) &\geq \Pr_{(\mathbf{V},B)\leftarrow k\text{-OV}_B^\alpha(n)}[\mathcal{A}^{\text{decide}}(\mathbf{V}) \neq B] \\ &= \mathop{\mathbb{E}}_{(\mathbf{U},\mathbf{s})\leftarrow\mathsf{Sample}}\left[\Pr_{\mathsf{Mix}}[\mathcal{A}^{\text{decide}}(\mathsf{Mix}(\mathbf{U})) \neq B(\mathbf{s})]\right] \\ &= \mathop{\mathbb{E}}_{(\mathbf{U},\mathbf{s})\leftarrow\mathsf{Sample}}\left[Z(\mathbf{U},\mathbf{s})\right].\end{aligned}$$

By Markov's inequality,

$$\Pr_{(\mathbf{U},\mathbf{s})\leftarrow\mathsf{Sample}}\left[Z(\mathbf{U},\mathbf{s}) \geq \frac{1}{12k}\right] \leq 12k\,\mathbb{E}[Z(\mathbf{U},\mathbf{s})] \leq 12k\cdot\delta(n).$$

Next, observe that $\mathbf{s}$ is the only solution in the instance $\mathbf{U}$ output by Sample with good probability,

$$\Pr_{(\mathbf{U},\mathbf{s})\leftarrow\mathsf{Sample}}[\mathbf{U} \text{ has a solution besides } \mathbf{s}] = \Pr_{\mathbf{U}\leftarrow k\text{-OV}_1^\alpha(n)}[c(\mathbf{U}) > 1] < \frac{1}{n^k}.$$

Thus, by a union bound over this and our result in the first step, we find that an instance $\mathbf{U} \sim k\text{-OV}_1^\alpha(n)$ is good except with probability at most $12k\cdot\delta(n) + \frac{1}{n^k}$:

$$\begin{aligned}&\Pr_{\mathbf{U}\leftarrow k\text{-OV}_1^\alpha(n)}[\mathbf{U} \text{ is good}] \\ &= \Pr_{\mathbf{U},\mathbf{s}\leftarrow\mathsf{Sample}}\left[\mathbf{s} \text{ is the only solution of } \mathbf{U} \text{ and } Z(\mathbf{U},\mathbf{s}) < \frac{1}{12k}\right] \\ &\geq 1 - \Pr_{\mathbf{U},\mathbf{s}\leftarrow\mathsf{Sample}}[\mathbf{U} \text{ has a solution besides } \mathbf{s}] - \Pr_{\mathbf{U},\mathbf{s}\leftarrow\mathsf{Sample}}\left[Z(\mathbf{U},\mathbf{s}) \geq \frac{1}{12k}\right] \\ &> 1 - \frac{1}{n^k} - 12k\cdot\delta(n).\end{aligned}$$

◀

We now show that the search algorithm performs well on this large fraction of good instances.

▶ **Lemma 15.** *Let $\mathcal{A}^{\text{decide}}$ be an algorithm that solves the planted decision k-OV problem with success probability $1-\delta(n)$. Then $\mathcal{A}^{\text{search}}$ fails to recover the solution $\mathbf{s}$ of good instances with probability less than $\delta(n)$.*

Proof. Let $\mathbf{U}$ be a good instance with its only solution at $\mathbf{s}$. After t iterations, we expect the counters for vectors in the solution $\mathbf{s}$ to be the highest. Using the fact that $\frac{1}{\sqrt[k]{2}} < 1 - \frac{1}{2k}$, we find that for non-planted vectors, i.e., where $s_\ell \neq i$,

$$\begin{aligned}\mathbb{E}[C_{\ell,i}] &= t\cdot\Pr_{\mathsf{Mix}}\left[U_{\ell,i} \text{ survives and } \mathcal{A}^{\text{decide}}(\mathsf{Mix}(\mathbf{U})) = 1\right] \qquad (*) \\ &\leq t\cdot\left(\Pr_{\mathsf{Mix}}\left[U_{\ell,i} \text{ survives and } B(\mathbf{s}) = 1\right] + \Pr_{\mathsf{Mix}}\left[\mathcal{A}^{\text{decide}}(\mathsf{Mix}(\mathbf{U})) \neq B(\mathbf{s})\right]\right) \\ &< t\cdot\left(\frac{1}{\sqrt[k]{2}}\cdot\frac{1}{2} + \frac{1}{12k}\right) < t\cdot\left(\left(1-\frac{1}{2k}\right)\frac{1}{2} + \frac{1}{12k}\right) \\ &= t\left(\frac{1}{2} - \frac{2}{12k}\right).\end{aligned}$$

On the other hand, for planted vectors, i.e., where $s_\ell = i$,

$$\begin{aligned}\mathbb{E}[C_{\ell,i}] &\geq t \cdot \Pr_{\mathsf{Mix}}\left[\mathbf{s} \text{ survives and } \mathcal{A}^{\text{decide}}(\mathsf{Mix}(\mathbf{U})) = 1\right] \qquad (**)\\ &= t \cdot \Pr_{\mathsf{Mix}}\left[B(\mathbf{s}) = 1 \text{ and } \mathcal{A}^{\text{decide}}(\mathsf{Mix}(\mathbf{U})) = B(\mathbf{s})\right]\\ &\geq t \cdot \left(1 - \Pr_{\mathsf{Mix}}\left[B(\mathbf{s}) = 0\right] - \Pr_{\mathsf{Mix}}\left[\mathcal{A}^{\text{decide}}(\mathsf{Mix}(\mathbf{U})) \neq B(\mathbf{s})\right]\right)\\ &> t \cdot \left(1 - \frac{1}{2} - \frac{1}{12k}\right) = t \cdot \left(\frac{1}{2} - \frac{1}{12k}\right).\end{aligned}$$

Picking the k highest counters is guaranteed to yield the solution $\mathbf{s}$ if the ranges of the counters of the planted and non-planted vectors do not overlap, that is to say if no counter deviates from its expected value by half the difference of (the bounds on) the two expected values $(**)$ and $(*)$, which we denote by Δ:

$$\Delta := \frac{1}{2}\left[t \cdot \left(\frac{1}{2} - \frac{1}{12k}\right) - t \cdot \left(\frac{1}{2} - \frac{2}{12k}\right)\right] = t\frac{1}{24k}.$$

Each counter $C_{\ell,i}$ is the sum of t i.i.d. binary random variables. By a Chernoff bound, a counter $C_{\ell,i}$ for a vector which is not in the planted solution, i.e., $s_\ell \neq i$, exceeds its expected value by Δ with probability at most

$$\Pr\left[C_{\ell,i} \geq \mathbb{E}[C_{\ell,i}] + \Delta\right] < \exp\left(-2t\left(\frac{\Delta}{t}\right)^2\right) = \exp\left(-\frac{2t}{24^2 \cdot k^2}\right).$$

Similarly, a counter $C_{\ell,i}$ for a vector that *is* part of a planted solution ($s_\ell = i$) falls short of its expected value by Δ with at most the same probability

$$\Pr\left[C_{\ell,i} \leq \mathbb{E}[C_{\ell,i}] - \Delta\right] < \exp\left(-2t\left(\frac{\Delta}{t}\right)^2\right) = \exp\left(-\frac{2t}{24^2 \cdot k^2}\right).$$

For a choice of $t = \frac{24^2 \cdot k^2}{2} \cdot \log\frac{kn}{\delta(n)} = O\left(\log\frac{n}{\delta(n)}\right)$ iterations, both of these probabilities are at most $\frac{\delta(n)}{kn}$. If *none* of the $k \cdot n$ counters deviate by Δ, the ranges of counters for vectors at $\mathbf{s}$ and those for vectors not at $\mathbf{s}$ are disjoint and selecting for the highest counters is guaranteed to yield $\mathbf{s}$. Thus, by a union bound over all $k \cdot n$ counters, we fail to recover $\mathbf{s}$ with probability at most $kn \cdot \frac{\delta(n)}{kn} = \delta(n)$. ◀

We can now complete the proof of Theorem 12.

Proof of Theorem 12. By Lemma 14, an instance $\mathbf{U} \sim k\text{–OV}_1^\alpha(n)$ is good except with probability at most $12k \cdot \delta(n) + \frac{1}{n^k}$. By Lemma 15, $\mathcal{A}^{\text{search}}$ is able to recover the solution $\mathbf{s}$ from a good instance except with probability at most $\delta(n)$. By a union bound against the instance not being good or $\mathcal{A}^{\text{search}}$ failing on a good instance, $\mathcal{A}^{\text{search}}$ solves the planted search k-OV problem with success probability at least $1 - 13k\delta(n) - \frac{1}{n^k}$.

In each of the $O\left(\log\frac{n}{\delta(n)}\right)$ iterations, we execute both Mix and $\mathcal{A}^{\text{decide}}$ once and update the counters. Each execution of Mix samples, in expectation, $k \cdot n \cdot (1 - 2^{-\frac{1}{k}}) = O(n)$ new vectors, each of which can be sampled in $\text{polylog}(n)$ expected time as explained in the proof of Theorem 11. $\mathcal{A}^{\text{decide}}$ runs in time $T(n)$ and updating the counters takes linear time, for the total expected runtime of $O\left(\left(T(n) + n\,\text{polylog}(n)\right)\log\frac{n}{\delta(n)}\right)$. ◀

References

1 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.

2 Shweta Agrawal, Sagnik Saha, Nikolaj I. Schwartzbach, Akhil Vanukuri, and Prashant Nalini Vasudevan. k-SUM in the sparse regime: Complexity and applications. In *Advances in Cryptology – CRYPTO 2024 – 44th Annual International Cryptology Conference*, volume 14921 of *Lecture Notes in Computer Science*, pages 315–351. Springer, 2024. doi:10.1007/978-3-031-68379-4_10.

3 Josh Alman. An illuminating algorithm for the light bulb problem. In *2nd Symposium on Simplicity in Algorithms, SOSA 2019*, volume 69 of *OASIcs*, pages 2:1–2:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASICS.SOSA.2019.2.

4 Josh Alman, Alexandr Andoni, and Hengjie Zhang. Faster algorithms for average-case orthogonal vectors and closest pair problems. In *2025 Symposium on Simplicity in Algorithms (SOSA)*, pages 473–484. SIAM, 2025. doi:10.1137/1.9781611978315.35.

5 Josh Alman, Yizhi Huang, and Kevin Yeo. Fine-grained complexity in a world without cryptography. In *Advances in Cryptology – EUROCRYPT 2025 – 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 15607 of *Lecture Notes in Computer Science*, pages 375–405. Springer, 2025. doi:10.1007/978-3-031-91098-2_14.

6 Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 483–496. ACM, 2017. doi:10.1145/3055399.3055466.

7 Enric Boix-Adserà, Matthew S. Brennan, and Guy Bresler. The average-case complexity of counting cliques in Erdős-Rényi hypergraphs. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1256–1280. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00078.

8 Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. Announced at SODA 2016. doi:10.1145/3402926.

9 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 21–40. SIAM, 2019. doi:10.1137/1.9781611975482.2.

10 Mina Dalirrooyfard, Andrea Lincoln, Barna Saha, and Virginia Vassilevska Williams. Average-case hardness of parity problems: Orthogonal vectors, k-SUM and more. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 4613–4643. SIAM, 2025. doi:10.1137/1.9781611978322.158.

11 Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. New techniques for proving fine-grained average-case hardness. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 774–785. IEEE, 2020. doi:10.1109/FOCS46700.2020.00077.

12 Itai Dinur, Nathan Keller, and Ohad Klein. Fine-grained cryptanalysis: Tight conditional bounds for dense k-SUM and k-XOR. *J. ACM*, 71(3):23, 2024. Announced at FOCS 2021. doi:10.1145/3653014.

13 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. *ACM Trans. Algorithms*, 15(2):23:1–23:35, 2019. Announced at SODA 2017. doi:10.1145/3196275.

14 Oded Goldreich and Guy N. Rothblum. Counting t-cliques: Worst-case to average-case reductions and direct interactive proof systems. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 77–88. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00017.

15 Shuichi Hirahara and Nobutaka Shimizu. Hardness self-amplification: Simplified, optimized, and unified. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023*, pages 70–83. ACM, 2023. doi:10.1145/3564246.3585189.

16 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. Announced at FOCS 1998. doi:10.1006/JCSS.2001.1774.

17 Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 44–61. ACM, 1989. doi:10.1145/73007.73012.

18 Ari Juels and Marcus Peinado. Hiding cliques for cryptographic security. *Designs, Codes and Cryptography*, 20(3):269–280, 2000. Announced at SODA 1998.

19 Daniel M. Kane and Richard Ryan Williams. The orthogonal vectors conjecture for branching programs and formulas. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019*, volume 124 of *LIPIcs*, pages 48:1–48:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ITCS.2019.48.

20 Matti Karppa, Petteri Kaski, and Jukka Kohonen. A faster subquadratic algorithm for finding outlier correlations. *ACM Trans. Algorithms*, 14(3):31:1–31:26, 2018. Announced at SODA 2016. doi:10.1145/3174804.

21 Matti Karppa, Petteri Kaski, Jukka Kohonen, and Padraig Ó Catháin. Explicit correlation amplifiers for finding outlier correlations in deterministic subquadratic time. *Algorithmica*, 82(11):3306–3337, 2020. Announced at ESA 2016. doi:10.1007/S00453-020-00727-1.

22 Karthik C. S. and Pasin Manurangsi. On closest pair in euclidean metric: Monochromatic is as hard as bichromatic. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019*, volume 124 of *LIPIcs*, pages 17:1–17:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ITCS.2019.17.

23 Rio LaVigne, Andrea Lincoln, and Virginia Vassilevska Williams. Public-key cryptography in the fine-grained setting. In *Advances in Cryptology – CRYPTO 2019 – 39th Annual International Cryptology Conference*, volume 11694 of *Lecture Notes in Computer Science*, pages 605–635. Springer, 2019. doi:10.1007/978-3-030-26954-8_20.

24 Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, 1978. doi:10.1145/359460.359473.

25 Alon Rosen. Fine-grained cryptography: A new frontier? *IACR Cryptol. ePrint Arch.*, page 442, 2020. URL: https://eprint.iacr.org/2020/442.

26 Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *J. ACM*, 62(2):13:1–13:45, 2015. Announced at FOCS 2012. doi:10.1145/2728167.

27 Virginia Vassilevska Williams. *On some fine-grained questions in algorithms and complexity*, pages 3447–3487. World Scientific, 2018. doi:10.1142/9789813272880_0188.

28 R. Ryan Williams. Some estimated likelihoods for computational complexity. In *Computing and Software Science – State of the Art and Perspectives*, volume 10000 of *Lecture Notes in Computer Science*, pages 9–26. Springer, 2019. doi:10.1007/978-3-319-91908-9_2.

29 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/J.TCS.2005.09.023.

Counting Small Induced Subgraphs: Scorpions Are Easy but Not Trivial

Radu Curticapean
University of Regensburg, Germany
IT University of Copenhagen, Denmark

Simon Döring
Max Planck Institute for Informatics, Saarbrücken, Germany
Saarland University (SIC), Saarbrücken, Germany

Daniel Neuen
Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

In the parameterized problem #IndSub(Φ) for fixed graph properties Φ, given as input a graph G and an integer k, the task is to compute the number of induced k-vertex subgraphs satisfying Φ. Dörfler et al. [Algorithmica 2022] and Roth et al. [SICOMP 2024] conjectured that #IndSub(Φ) is #W[1]-hard for all non-meager properties Φ, i.e., properties that are nontrivial for infinitely many k. This conjecture has been confirmed for several restricted types of properties, including all hereditary properties [STOC 2022] and all edge-monotone properties [STOC 2024].

We refute this conjecture by showing that induced k-vertex graphs that are *scorpions* can be counted in time $O(n^4)$ for all k. Scorpions were introduced more than 50 years ago in the context of the evasiveness conjecture. A simple variant of this construction results in graph properties that achieve arbitrary intermediate complexity assuming ETH.

Moreover, we formulate an updated conjecture on the complexity of #IndSub(Φ) that correctly captures the complexity status of scorpions and related constructions.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms

Keywords and phrases induced subgraphs, counting complexity, parameterized complexity, scorpions

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.96

Related Version *Full Version*: https://arxiv.org/abs/2505.22300

Funding The research is funded by the European Union (ERC, CountHom, 101077083). Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

Counting small patterns in graphs is a fundamental problem in computer science, with applications in bioinformatics [39], network analysis [28, 38], databases [18], and other areas. In this paper, we focus on counting certain induced k-vertex graphs in a large n-vertex input graph G, for fixed k.

Counting Induced Copies of a Fixed Graph. The starting point of our investigation is the problem of counting induced H-copies for a fixed *individual* graph H. Formally, this problem asks for the number of sets $X \subseteq V(G)$ such that the induced subgraph $G[X]$ is isomorphic to H. We stress that H is considered fixed in this problem, and only G is the input. In particular, each such problem can be solved in time $O(n^k)$, which is polynomial in n for fixed k.

© Radu Curticapean, Simon Döring, and Daniel Neuen;
licensed under Creative Commons License CC-BY 4.0

Some improvements over the trivial $O(n^k)$ running time are known: For example, triangles can be counted in $O(n^\omega)$ time [21], where $\omega < 2.372$ is the optimal exponent of $n \times n$ matrix multiplication [1]. This improvement can be lifted to cliques beyond K_3: Denoting by $C(K_k)$ the optimal exponent for counting k-cliques in n-vertex graphs, similar to the exponent of matrix multiplication, we have $C(K_k) \leq \omega \cdot \lceil k/3 \rceil$ (see [30]). Under the Exponential-Time Hypothesis ETH, there exists a fixed constant α such that $C(K_k) \geq \alpha \cdot k$ (see [4]).

Writing $C_{\text{ind}}(H)$ to denote the optimal exponent of counting induced H-subgraphs [5] for fixed H, a straightforward reduction shows that $C_{\text{ind}}(H) = C(K_k)$ for all k-vertex graphs H. In other words, for fixed k, all induced H-counting problems with k-vertex H are equally hard, and they require an exponent of $\Omega(k)$ under ETH. Compare this to counting not necessarily induced subgraphs, where different patterns can yield different complexity exponents: The number of subgraph copies of the edgeless k-vertex graph IS_k in an n-vertex graph is $\binom{n}{k}$ and can thus be computed in linear time, while counting k-cliques is hard.

Counting Patterns From a Set. In recent years, counting occurrences of individual k-vertex patterns H has been generalized to counting pattern occurrences from a fixed *set* of patterns [22, 23]. In this setting, we fix a number $k \in \mathbb{N}$ and a set $\mathcal{H}$ of k-vertex graphs. On input G, we wish to count the induced k-vertex subgraphs of G isomorphic to some $H \in \mathcal{H}$. This subsumes the problem of counting induced H-copies, but also allows us to address, e.g., the problem of counting connected k-vertex graphs [22].

Of course, for every set $\mathcal{H}$, this problem can be solved by counting induced H-copies for the individual graphs $H \in \mathcal{H}$, which readily implies an $O(n^{C(K_k)})$ time algorithm. Significantly faster algorithms were only known for *trivial* pattern sets $\mathcal{H}$, i.e., if the pattern set $\mathcal{H}$ is empty or contains all k-vertex graphs. In these cases, the output is just 0 or $\binom{n}{k}$, respectively. Also, for some pattern sets $\mathcal{H}$, slight improvements can be obtained over the $O(n^{C(K_k)})$ time algorithm [9], but no nontrivial set $\mathcal{H}$ of k-vertex graphs with exponent strictly less than $C(K_{\lfloor k/2 \rfloor})$ was known. In other words, counting induced patterns from a fixed set $\mathcal{H}$ of k-vertex graphs appeared to be either trivial or very hard.

Parameterized Complexity. In the literature, pattern counting is often phrased in terms of graph properties Φ that may hold on infinitely many graphs rather than finite sets $\mathcal{H}$. More specifically, in the problem #IndSub(Φ) for a fixed graph property Φ, the input is a graph G and $k \in \mathbb{N}$, and we ask to count the induced k-vertex subgraphs of G satisfying Φ. Compared to the previous setting, the pattern size k is now part of the input rather than a fixed constant, so the problem #IndSub(Φ) may not be polynomial-time solvable. We say that #IndSub(Φ) is *fixed-parameter tractable* if it can be solved in time $f(k) \cdot n^{O(1)}$ for a computable function f, and we would like to understand which properties Φ render #IndSub(Φ) fixed-parameter tractable.

In prior literature, all known properties Φ with fixed-parameter tractable #IndSub(Φ) are essentially trivial: Formally, we call Φ *meager* if the restriction $\Phi^{(k)}$ of Φ to k-vertex graphs (i.e., the k-th slice of Φ) is trivial for all but finitely many $k \in \mathbb{N}$. Meager computable properties Φ trivially render #IndSub(Φ) fixed-parameter tractable. Complementing this, the problem #IndSub(Φ) was conjectured to be #W[1]-hard for all other computable properties Φ. Here, #W[1] is the parameterized analogue of #P; it is known that ETH rules out fixed-parameter tractable algorithms for all #W[1]-hard problems (see, e.g., [15]).

▶ **Conjecture 1** ([12, 16, 37]). *For every property Φ that is computable and not meager, the problem* #IndSub(Φ) *is* #W[1]*-hard.*

This conjecture has been verified for general classes of properties Φ, e.g., properties that are closed under deleting edges [9, 13, 14] or deleting vertices [16], and other natural classes of properties [9, 22, 23, 35, 37]. This made it a plausible working hypothesis in the area.

Our Results. We show that **Conjecture 1 fails**. That is, we exhibit non-meager properties Φ such that $\#\textsc{IndSub}(\Phi)$ is fixed-parameter tractable – in fact, even polynomial-time solvable.

These counterexamples are derived from simple constructions that were introduced 50 years ago in the context of the *evasiveness conjecture* [27, 33, 26], a major open problem about the worst-case query complexity of graph properties. In the setting of the evasiveness conjecture, we are given a fixed graph property Φ, and we wish to decide whether an unknown k-vertex graph G is contained in Φ, where G can only be accessed via edge queries. More precisely, the vertex set $V(G)$ is known, and for two vertices v, w, we can query whether vw is an edge of G. Clearly, every property can be decided by querying all $\binom{k}{2}$ pairs of vertices, and the *evasiveness conjecture* postulates that this trivial worst-case upper bound is optimal for all *(edge-)monotone* properties.

However, when we drop the monotonicity requirement, properties Φ with linear query complexity $O(k)$ are known [27, 33], and it turns out that those examples can be used to refute Conjecture 1.

- As a simple counterexample, the property Ψ_{sink} of having a sink in a directed graph is nontrivial on graphs of fixed size $k \geq 2$, and hence not meager. Here, a graph D satisfies Ψ_{sink} if there is a vertex $s \in V(D)$ with $(v, s) \in E(D)$ for all $v \neq s$.[1] Nevertheless, $\#\textsc{IndSub}(\Psi_{\text{sink}})$ can be solved in linear time on directed graphs without antiparallel edges. We invite the reader to discover the algorithm themselves before proceeding to Section 3.1. This property actually presented the first counterexample to a (too strong) version of the evasiveness conjecture on directed graphs [33].
- A marginally more involved construction also works for undirected graphs: The *scorpion property* Ψ is a non-meager property of undirected graphs such that $\#\textsc{IndSub}(\Psi)$ can be solved in $O(n^4)$ time on general undirected graphs. Scorpions were the first counterexample to a (too strong) variant of the evasiveness conjecture for undirected graphs [27], prompting the restriction of this conjecture to monotone properties.
- We show more generally that $\#\textsc{IndSub}(\Psi)$ can be made gradually harder: For every $\ell \in \mathbb{N}$, we construct a *generalized scorpion property* Ψ_ℓ such that $\#\textsc{IndSub}(\Psi_\ell)$ can be solved in $O(n^{\ell+3})$ time, while ETH rules out $O(n^{\alpha \cdot \ell})$ time algorithms for a fixed constant $\alpha > 0$. Our construction also allows for ℓ to be a function in k. Under the Strong Exponential-Time Hypothesis SETH, we rule out $O(n^{\ell+2-\varepsilon})$ time algorithms.

Finally, informed by these counterexamples, we formulate a new hypothesis on the computational complexity of $\#\textsc{IndSub}(\Phi)$ for properties Φ. This new hypothesis is more technical and explained in Section 4. In a nutshell, it is based around the well-established fact that sums of induced pattern counts like the k-th slices $\#\text{IndSub}(\Phi^{(k)} \to G)$ can be expressed as linear combinations of (not necessarily induced) subgraph counts [5], and that such basis changes may help in understanding the complexity of a problem [5, 6, 9, 12, 13, 14, 16, 17, 24, 35, 37, 36].

[1] Contrary to contemporary notation, in which a *sink* is a vertex of out-degree 0, we use the term *sink* to refer to a vertex $s \in V(D)$ such that $(v, s) \in E(D)$ for all $v \neq s$. This follows the terminology of the original paper [33], which is still common in the context of the evasiveness conjecture.

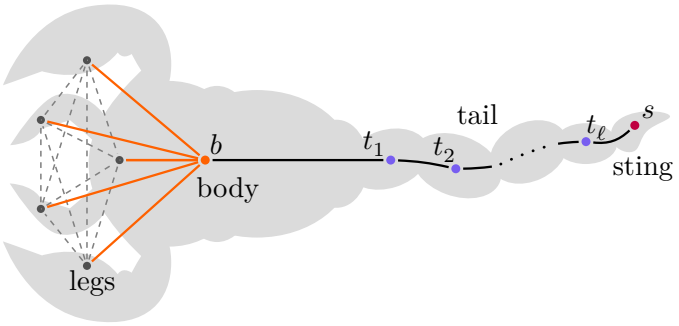


Figure 1 A graph H is an ℓ-scorpion if it has the above form: Dashed edges *may* be present in H or not, solid edges *must* be present, and non-drawn edges *must not* be present.

More specifically, the new hypothesis postulates that a useful phenomenon occurs when expressing the graph parameter $\#\mathrm{IndSub}(\Phi^{(k)} \to G)$ as a linear combination of subgraph counts: Any hard term in such a linear combination ensures hardness of the entire linear combination. General linear combinations of k-vertex subgraph counts do not enjoy this useful phenomenon. We demonstrate in Section 4 that our conjecture would imply useful new ways of proving hardness of $\#\mathrm{IndSub}(\Phi^{(k)} \to G)$.

Further Connections to the Evasiveness Conjecture. Notably, this work is not the first to draw connections between the evasiveness conjecture and the computational complexity of $\#\textsc{IndSub}(\Phi)$. Indeed, this connection was already made by Roth and Schmitt [35], who adapted the topological approach to the evasiveness conjecture, dating back to Kahn, Saks and Sturtevant [25], to prove lower bounds for $\#\textsc{IndSub}(\Phi)$ for certain graph properties Φ. In a similar spirit, [9] adapts a lower bound of $\Omega(k^2)$ on the query complexity of monotone graph properties due to Rivest and Vuillemin [32] to obtain hardness of $\#\textsc{IndSub}(\Phi)$ for monotone properties Φ. The present work complements these connections on the upper-bound side: We observe that standard examples of properties Φ with linear query complexity admit efficient algorithms for $\#\textsc{IndSub}(\Phi)$.

Currently, these connections seem to be restricted to specific examples and proof techniques. We believe that it is an interesting open question whether a general connection can be established in either direction. Without going into details, we point out that the *Fourier degree* of a property Φ (when Φ is restricted to a slice and viewed as a Boolean function) provides a lower bound both for the query complexity of Φ [31], and for the computational complexity of $\#\textsc{IndSub}(\Phi)$ [9]. However, in both cases, large Fourier degree is only known to give a *sufficient* criterion for hardness, so it does not establish a direct link between the query complexity of Φ and the complexity of $\#\textsc{IndSub}(\Phi)$.

2 Preliminaries

We write $\mathbb{N} = \{1, 2, 3, \dots\}$ and $[n] \coloneqq \{1, ..., n\}$ for $n \in \mathbb{N}$. For a set A, we write $\binom{A}{k}$ for the set of all k-element subsets of A.

Graph Theory. We follow standard textbooks [11] for graph-theoretic notation. Unless stated otherwise, graphs are simple (i.e., without multiedges or self-loops) and undirected. We write $V(G)$ and $E(G)$ for the vertex and edge set of G, respectively, and we write $N_G(v) \coloneqq \{u \mid uv \in E(G)\}$ for the neighborhood of $v \in V(G)$ in G.

A graph H is a *subgraph* of G, written $H \subseteq G$, if it can be obtained by deleting vertices and edges from G. For $X \subseteq V(G)$, we write $G[X]$ to denote the *induced subgraph* on X. For $S \subseteq E(G)$, we write $G[S]$ for the subgraph with vertex set $V(G)$ and edge set S. For $k, \ell \in \mathbb{N}$, we write K_ℓ for the complete ℓ-vertex graph, IS_ℓ for the edgeless ℓ-vertex graph, and $K_{\ell,k}$ for the complete bipartite graph on $\ell + k$ vertices.

Induced Subgraph Counts. A *graph property* Φ is a function that maps each graph G to $\{0,1\}$ and is invariant under isomorphisms, i.e., $\Phi(G) = \Phi(H)$ for all isomorphic graphs G, H. For $k \in \mathbb{N}$, the *k-th slice of* Φ, denoted by $\Phi^{(k)}$, is the restriction of Φ to k-vertex graphs. We implicitly identify $\Phi^{(k)}$ with the set of all k-vertex graphs G satisfying $\Phi(G) = 1$, and we use graph properties and sets of graphs interchangeably. For $k \in \mathbb{N}$ and a graph G, the number of k-vertex induced subgraphs $G[X]$ satisfying Φ will be denoted by

$$\#\mathrm{IndSub}(\Phi^{(k)} \to G) \coloneqq \sum_{\substack{X \subseteq V(G) \\ |X|=k}} \Phi(G[X]).$$

Moreover, we write $\#\mathrm{IndSub}(\Phi^{(k)} \to \star)$ for the map $G \mapsto \#\mathrm{IndSub}(\Phi^{(k)} \to G)$.

Complexity Theory. A *parameterized problem* consists of a function $P\colon \Sigma^* \to \mathbb{N}$ and a computable parameterization $\kappa\colon \Sigma^* \to \mathbb{N}$. It is *fixed-parameter tractable* (FPT) if there is a computable function f, a constant $c \in \mathbb{N}$, and a deterministic algorithm $\mathbb{A}$ that computes $P(x)$ in time $O(f(\kappa(x)) \cdot |x|^c)$ for all $x \in \Sigma^*$. We write $\#\textsc{IndSub}(\Phi)$ for the parametrized problem that gets as input a graph G and a *parameter* k, and computes $\#\mathrm{IndSub}(\Phi^{(k)} \to G)$.

For lower bounds, we rely on the *Exponential-Time Hypothesis* (ETH) [20], which asserts the existence of some $\varepsilon > 0$ such that the Boolean satisfiability problem on n-variable 3-CNF formulas cannot be solved in time $O(2^{\varepsilon \cdot n})$ (see also [10, Conjecture 14.1]). The *Strong Exponential-Time Hypothesis* (SETH) [19] states that for all $\varepsilon > 0$, there is a $k \geq 3$ such that Boolean Satisfiability on k-CNF formulas cannot be solved in $O(2^{(1-\varepsilon)\cdot n})$ (see also [10, Conjecture 14.2]).

3 Main Result

To present the idea underlying the tractability of scorpions, we first consider a variant for directed graphs, where this idea becomes particularly simple. Then we introduce scorpions and their generalizations and prove the claimed upper and lower complexity bounds. Finally, we use scorpions to show that a useful and previously known complexity lower bound on the complexity of $\#\textsc{IndSub}(\Phi)$ is tight.

3.1 Directed Graphs Containing a Sink

As a warm-up, we consider a property Ψ_{sink} of directed graphs that is nontrivial but yields a linear-time counting problem $\#\textsc{IndSub}(\Psi_{\mathrm{sink}})$. For this subsection, we momentarily consider directed graphs without antiparallel edges, i.e., at most one of the edges (u, v) and (v, u) may be present. The property Ψ_{sink} is defined to hold on H if there is a *sink* vertex $s \in V(H)$, i.e., a vertex s such that $(u, s) \in E(H)$ for all $u \in V(H) \setminus \{s\}$. This property is clearly nontrivial in every slice $k \geq 2$, as there are k-vertex graphs with a sink and k-vertex graphs without a sink (e.g., an in-star versus the edgeless graph).

Towards an algorithm, we observe crucially that every graph without antiparallel edges contains at most one sink, since two distinct sinks u, v would imply the presence of both edges (u, v) and (v, u). Hence, for an input graph G, the set of k-vertex sets containing a sink

$$\mathcal{X} \coloneqq \left\{ X \in \binom{V(G)}{k} \;\middle|\; G[X] \in \Psi_{\text{sink}} \right\}$$

can be partitioned, according to the unique sink, into

$$\mathcal{X} = \bigcup_{v \in V(G)} \mathcal{X}_v \quad \text{with} \quad \mathcal{X}_v \coloneqq \{X \in \mathcal{X} \mid v \text{ is the sink of } G[X]\}.$$

Finally, for fixed $v \in V(G)$, every set $X \in \mathcal{X}_v$ has the form $X = \{v, w_1, \ldots, w_{k-1}\}$ with all $w_1, \ldots, w_{k-1}$ pairwise distinct, distinct from v, and incoming neighbors of v, i.e., they satisfy $(w_i, v) \in E(G)$. Writing $\text{in}_G(v)$ for the number of incoming neighbors of v, it follows that

$$|\mathcal{X}_v| = \binom{\text{in}_G(v)}{k-1}.$$

Combining the above equations, we readily obtain a linear-time algorithm for $\#\textsc{IndSub}(\Psi_{\text{sink}})$ by computing $\text{in}_G(v)$ for all $v \in V(G)$ and evaluating the resulting formula

$$\#\text{IndSub}(\Psi^{(k)}_{\text{sink}} \to G) = \sum_{v \in V(G)} |\mathcal{X}_v| = \sum_{v \in V(G)} \binom{\text{in}_G(v)}{k-1}. \tag{1}$$

3.2 Generalized Scorpions

The algorithmic idea for counting k-vertex graphs with a sink vertex applies whenever the set $\mathcal{X}$ of induced subgraphs to be counted admits a partition into few sets $\mathcal{X}_i$ such that each $|\mathcal{X}_i|$ is easily determined. More specifically, we consider partitions in which each $\mathcal{X}_i$ is determined by the manifestation of a special small set of uniquely identifiable vertices (e.g., the sink vertex), while the subgraph induced by the other vertices is irrelevant.

To apply this idea to undirected graphs, we use a construction that was first presented in [27] for the so-called *evasiveness conjecture*, and which has become a standard example in this context (see, e.g., [26, Section 13.1]): A graph H is a *scorpion* if it can be obtained from an arbitrary graph H' with $|V(H')| \geq 2$ by adding fresh vertices b, t, s, making b adjacent to all of H', and then adding the edges bt and ts. The vertices b, t, s are usually called *body*, *tail* and *sting*, and it can be shown crucially (see Lemma 3 below) that these vertices are uniquely recoverable from H. Similarly to the arguments above, this allows us (in Theorem 4) to design an efficient algorithm to count induced scorpions.

In this paper, we prove these statements for a slightly generalized version of scorpions, since this allows us to obtain gradually harder properties. Towards this end, we replace the tail vertex t by a path of ℓ vertices $t_1, \ldots, t_\ell$. See also Figure 1.

▶ **Definition 2.** *For $\ell \in \mathbb{N}$, an* ℓ-scorpion *is a graph H with $|V(H)| \geq \ell + 4$ that admits a tuple of pairwise distinct vertices*

$$(\underbrace{b}_{\text{body of } H}, \underbrace{t_1, \ldots, t_\ell}_{\text{tail of } H}, \underbrace{s}_{\text{sting of } H}) \in V(H)^{\ell+2},$$

such that the following holds: Writing $Q \coloneqq \{b, t_1, \ldots, t_\ell, s\}$ and calling the vertices in $V(H) \setminus Q$ the legs *of H, we have that*

- *the graph $H[Q]$ is an induced path from the body b to the sting s,*
- *the body b is adjacent to all legs, and*
- *the body b is the only vertex in Q adjacent to legs.*

We define Ψ_ℓ as the class of all ℓ-scorpions.

The property Ψ_ℓ is non-meager for all $\ell \geq 1$: Indeed, for $k \geq \ell + 4$, at least one k-vertex scorpion exists (e.g., consisting of the sting, tail, body, and an independent set of legs), while the k-vertex graph K_k is not a scorpion.

Note that scorpion graphs are 1-scorpions. Also note that the definition speaks about "the" body, tail, and sting vertices, as if they were unique. Indeed, they are:

▶ **Lemma 3.** *If H is an ℓ-scorpion, then its body, tail, and sting are unique.*

Proof. Let $k = |V(H)|$ and recall that $k \geq \ell + 4$ by Definition 2. The body has degree $k - \ell - 1 \geq 3$. Since every leg has degree at most $k - \ell - 2$, we conclude that H contains exactly one vertex of degree $k - \ell - 1$; this uniquely identifies b.

The vertices not adjacent to b are precisely $t_2, \ldots, t_\ell, s$. Among these, s is the only vertex of degree 1 and is thus uniquely identified. The vertices $t_1, \ldots, t_\ell$ are uniquely identified by their distance from s, since vertex t_i is the only vertex at distance $\ell - i + 1$ from s. ◀

We use this lemma to show that #IndSub(Ψ_ℓ) can be solved in polynomial time for every fixed $\ell \geq 1$, similarly to the algorithm presented in Section 3.1.

▶ **Theorem 4.** *There is an algorithm that, given $k \geq \ell + 4$ and an n-vertex graph G as input, computes $\#\mathrm{IndSub}(\Psi_\ell^{(k)} \to G)$ in time $O(\ell \cdot n^{\ell+3})$.*

Proof. Let $\mathcal{X}$ be the set of induced k-vertex graphs in G that are ℓ-scorpions. In our proof, we use the uniqueness of body, tail, and sting to partition $\mathcal{X}$ into classes $\mathcal{X}_{\mathbf{q}}$ for $\mathbf{q} \in V(G)^{\ell+2}$ and then determine each $|\mathcal{X}_q|$ in linear time. Then the algorithm follows from

$$\#\mathrm{IndSub}(\Psi_\ell^{(k)} \to G) = |\mathcal{X}| = \sum_{\mathbf{q} \in V(G)^{\ell+2}} |\mathcal{X}_{\mathbf{q}}|. \tag{2}$$

More specifically, let $\mathcal{P} \subseteq V(G)^{\ell+2}$ denote the tuples inducing a path in G. Given $\mathbf{q} \in \mathcal{P}$ with $\mathbf{q} = (b, t_1, \ldots, t_\ell, s)$, let $\mathcal{X}_{\mathbf{q}} \subseteq \mathcal{X}$ denote the set of k-vertex ℓ-scorpions in G with body b, tail $t_1, \ldots, t_\ell$ and sting s. By Lemma 3, the sets $\mathcal{X}_{\mathbf{q}}$ for $\mathbf{q} \in \mathcal{P}$ partition $\mathcal{X}$, so (2) holds with $\mathcal{X}_{\mathbf{q}} = \emptyset$ for $\mathbf{q} \notin \mathcal{P}$. It remains to determine $|\mathcal{X}_{\mathbf{q}}|$ for $\mathbf{q} = (b, t_1, \ldots, t_\ell, s) \in \mathcal{P}$. We show that

$$|\mathcal{X}_{\mathbf{q}}| = \binom{|X_G(\mathbf{q})|}{k - \ell + 2} \quad \text{with } X_G(\mathbf{q}) \coloneqq \left| N_G(b) \setminus \bigcup_{v \in \{t_1, \ldots, t_\ell, s\}} N_G(v) \right|. \tag{3}$$

Indeed, since the $\ell + 2$ vertices in $\mathbf{q}$ induce a path, the set $\mathcal{X}_{\mathbf{q}}$ consists of all k-vertex sets X that contain $\{b, t_1, \ldots, t_\ell, s\}$ and $k - \ell - 2$ additional vertices (the legs) that are adjacent to b and not adjacent to any t_i or s. The number of such sets X is precisely $|\mathcal{X}_{\mathbf{q}}| = \binom{X_G(\mathbf{q})}{k-\ell+2}$.

An algorithm with the claimed running time follows from evaluating (2) term by term while using (3) to determine $|\mathcal{X}_{\mathbf{q}}|$. Indeed, $\mathcal{P}$ can be enumerated in time $O(n^{\ell+2})$ and $|\mathcal{X}_{\mathbf{q}}|$ for $\mathbf{q} \in \mathcal{P}$ can be computed via (3) in time $O(\ell n)$. ◀

Since Ψ_ℓ is non-meager for every $\ell \geq 1$, Theorem 4 refutes Conjecture 1 even with $\ell = 1$.

3.3 Lower Bounds Based on ETH

Next, we show that the running time obtained in Theorem 4 is essentially optimal under ETH. In particular, by choosing ℓ as a function of k, we can use ℓ-scorpions to obtain properties with varying computational difficulty.

To obtain the lower bound, we rely on a result from [6, 9, 37] that provides a lower bound based on the set of possible Hamming weights attained by a property Φ: Given a graph property Φ and $k \in \mathbb{N}$, we say that Φ *attains weight* ℓ *on slice* k if there is a graph $H \in \Phi$ with k vertices and ℓ edges. It *avoids weight* ℓ *on slice* k if it does not attain it. Note that every property attains between 0 and $\binom{k}{2}+1$ different weights on slice k, since the attained weights form a subset of $\{0, \dots, \binom{k}{2}\}$.

▶ **Theorem 5** ([9, Lemma 5.1] & [6, Theorem 7.1])**.** *Assuming* ETH*, there are* $N_0, \delta > 0$ *such that the following holds: If* $k \geq N_0$ *and* $0 < d \leq k/2$ *and* Φ *avoids at least* $d \cdot k$ *distinct weights and attains at least one weight, all on slice* k*, then no algorithm computes* $\#\mathrm{IndSub}(\Phi^{(k)} \to G)$ *in time* $O(n^{\delta \cdot d})$*.*

For $k \geq \ell + 4$, we can easily determine the number of weights attained by Ψ_ℓ on slice k: Since there is no freedom in choosing edges incident to body, tail and sting, this number is precisely $\binom{k-\ell-2}{2}+1$. Thus, the number of weights avoided by Ψ_ℓ on slice k is

$$\binom{k}{2} - \binom{k-\ell-2}{2} - 1 = k(\ell+2) - \frac{\ell(\ell+5)}{2} - 4.$$

For $k \geq 2\ell + 4$, it follows that Ψ_ℓ avoids at least $\ell/2 \cdot k$ weights on slice k. Being nontrivial, it attains at least one weight. Theorem 5 readily implies:

▶ **Corollary 6.** *Assuming* ETH*, there are* $N_0, \delta > 0$ *such that the following holds: If* $k \geq N_0$ *and* $0 < \ell \leq (k-4)/2$*, then no algorithm computes* $\#\mathrm{IndSub}(\Psi_\ell^{(k)} \to G)$ *in time* $O(n^{\delta \cdot \ell})$*.*

▶ Remark 7. The algorithm from Theorem 4 also demonstrates that the lower bound in Theorem 5 cannot be further improved. Indeed, Ψ_ℓ avoids at most $(\ell+2) \cdot k$ weights on slice k, but $\#\mathrm{IndSub}(\Phi^{(k)} \to G)$ can be solved in $O(n^{\ell+3})$ time. Therefore, ℓ-scorpions show that the bound in Theorem 5 is essentially tight.

With the lower bound of Corollary 6 at hand, we can obtain properties with varying computational complexity, by choosing ℓ dependent on k. For example, setting $\ell \approx \sqrt{k}$ yields a property Ψ_{sqrt} for which $\#\textsc{IndSub}(\Psi_{\mathrm{sqrt}})$ takes $n^{\Theta(\sqrt{k})}$ time under ETH. More generally, consider a monotone increasing $f\colon \mathbb{N} \to \mathbb{N}$ with $1 \leq f(k) \leq (k-4)/2$ for all $k \in \mathbb{N}$. We define $\Psi_f = \bigcup_{k \in \mathbb{N}} \Psi_{f(k)}^{(k)}$ to contain exactly the k-vertex $f(k)$-scorpions for all $k \in \mathbb{N}$.

▶ **Corollary 8.** *Let* $f\colon \mathbb{N} \to \mathbb{N}$ *be monotone increasing with* $1 \leq f(k) \leq (k-4)/2$*. Then* $\#\textsc{IndSub}(\Psi_f)$ *can be solved in time* $O(k \cdot n^{f(k)+3})$*, and assuming* ETH*, not in time* $O(n^{\alpha \cdot f(k)})$ *for a fixed constant* $\alpha > 0$*.*

In the full version of the paper, we combine results from Section 4 with a result from [3] to obtain the following lower bound under the Strong Exponential-Time Hypothesis SETH.

▶ **Proposition 9.** *Assuming* SETH*, for every* $\ell \geq 3$ *and* $\varepsilon > 0$*, there is no algorithm solving* $\#\textsc{IndSub}(\Psi_\ell)$ *in time* $O(n^{\ell+2-\varepsilon})$*.*

We stress that, in contrast to Corollaries 6 and 8, the SETH lower bound from Proposition 9 only holds for fixed ℓ, and not if ℓ is chosen as a function of k.

4 An Updated Conjecture Using the Subgraph Basis

Seeing what scorpions have done to Conjecture 1, it is natural to update the conjecture and reconsider the question when #IndSub(Φ) is fixed-parameter tractable. Towards this, we first observe that our counterexamples can be generalized in a number of ways while keeping their key features. This suggests that a dichotomy theorem for #IndSub(Φ) will have to encompass various tractable cases, as the following facts indicate:

- The key to the efficient algorithm in Theorem 4 is that body, tail and sting are uniquely identified in every ℓ-scorpion, and that no restrictions are imposed on the subgraph induced by the legs. As long as there is a constant-sized set of uniquely identifiable vertices and no restrictions on the remaining vertices, an efficient algorithm for #IndSub(Φ) follows.
- This however is not the final word: Observe that #IndSub(Φ) has the same complexity as #IndSub($\neg\Phi$), where $\neg\Phi$ contains exactly those graphs that are not contained in Φ (see, e.g., [37, Fact 2.3]). Notably, the non-scorpion property $\neg\Psi_\ell$ contains the complete graph on k vertices for every $k \geq \ell + 4$, which arguably has no constant-sized set of "uniquely identifiable" vertices.
- Beyond, we can also "nest" easy properties: For example, let Λ be the graph property containing all graphs H with $|V(H)| \geq 9$ so that (a) H is a 1-scorpion, with some set of legs X, and (b) $H[X]$ is not a 2-scorpion. Then #IndSub(Λ) can be solved in polynomial time using similar arguments as in Theorem 4.

The diversity of these examples suggests that a complexity classification for all properties Φ may be quite intricate. To obtain a new classification conjecture, we first express the induced subgraph counts arising in #IndSub(Φ) as linear combinations of (not necessarily induced) subgraph counts. Such basis changes among counting problems have already been used for #IndSub(Φ) before [6, 9, 12, 13, 14, 16, 35, 37].[2]

In the following, recall that $\Phi^{(k)}$ for $k \in \mathbb{N}$ is the restriction of Φ to k-vertex graphs and let #IndSub($\Phi^{(k)} \to \star$) be the number of induced k-vertex graphs satisfying $\Phi^{(k)}$ in an input graph. Likewise let #Sub($H \to \star$) denote the number of not necessarily induced H-subgraph copies in an input graph. Then there exists a finite set $\mathcal{H}$ of unlabeled graphs on exactly k vertices, and coefficients α_H for $H \in \mathcal{H}$, such that

$$\#\text{IndSub}(\Phi^{(k)} \to \star) = \sum_{H \in \mathcal{H}} \alpha_H \cdot \#\text{Sub}(H \to \star); \tag{4}$$

see [2, 5]. In fact, an explicit formula for the coefficients can be found through inclusion-exclusion: Given a graph property Φ and graph H, the coefficient α_H in (4) is given by the so-called *alternating enumerator* $\widehat{\Phi}(H)$ (see [9, 35]), sometimes defined without the $(-1)^{|E(H)|}$ factor below [12, 13, 14],

$$\widehat{\Phi}(H) \coloneqq (-1)^{|E(H)|} \sum_{S \subseteq E(H)} (-1)^{|S|} \Phi(H[S]).$$

Evaluating fixed finite linear combinations of k-vertex graphs as in (4) is asymptotically no harder than evaluating the individual H-subgraph counts with $\widehat{\Phi}(H) \neq 0$. For fixed graphs H, individual H-subgraph counts in turn can be evaluated in time $O(n^{\tau(H)+1})$, where

[2] Going further, the problem #IndSub(Φ) has already been expressed in the similar basis of *homomorphism counts*, and a dichotomy criterion was shown in terms of this basis [5]. However, the transformation into the homomorphism basis renders combinatorial interpretations of Φ opaque and only yields an implicit dichotomy criterion. In particular, all complexity results listed above were achieved using the *subgraph* basis rather than the homomorphism basis.

$\tau(H)$ is the vertex-cover number of H (see [5, Theorem 1.1]). In particular, this implies that $\#\textsc{IndSub}(\Phi)$ is fixed-parameter tractable when only graphs H of small vertex-cover number satisfy $\widehat{\Phi}(H) \neq 0$. More quantitatively, writing $\tau_\Phi(k)$ for the maximal vertex cover number among k-vertex graphs H with $\widehat{\Phi}(H) \neq 0$, we have:

▶ **Theorem 10** (see, e.g., [8, Theorem 3.6]). *For every computable graph property Φ, the problem $\#\textsc{IndSub}(\Phi)$ can be solved in time $f(k) \cdot n^{\tau_\Phi(k)+1}$ for a computable function f.*

Conversely, under certain conditions, a k-vertex graph H with $\widehat{\Phi}(H) \neq 0$ that is hard on its own also implies hardness of the entire linear combination. For example, under the guarantee that all patterns in the linear combination have the same number k of vertices, this occurs when some graph H with $\widehat{\Phi}(H) \neq 0$ has large treewidth (see, e.g., [12]). Large treewidth is however only a sufficient criterion, since counting k-matchings (and more generally, patterns of large vertex-cover number) is hard as well [7].

Knowing about a similar phenomenon for the related case of homomorphism counts, one[3] might be led to believe that linear combinations of k-vertex subgraph counts are hard if at least one pattern has large vertex-cover number. However, this hypothesis is too optimistic, as shown by k-partial determinants: These are weighted counts of k-vertex cycle covers C in a graph G, where C is counted with weight -1 if it contains an odd number of cycles, and with weight 1 otherwise. The patterns in the subgraph count expansion of k-partial determinants are precisely the cycle covers on k vertices. These do have large vertex-cover number, yet k-partial determinants can be computed in time $O(n^{\omega+1})$ due to the same cancellations that render usual determinants tractable.

In our updated version of Conjecture 1, we assert that such cancellations cannot occur for $\#\textsc{IndSub}(\Phi)$. That is, we assert that graphs of large vertex-cover number in the subgraph expansion of $\#\textsc{IndSub}(\Phi)$ indeed render the linear combination hard. Recall that $\tau_\Phi(k)$ denotes the maximal vertex cover number among k-vertex graphs H with $\widehat{\Phi}(H) \neq 0$.

▶ **Conjecture 11.** *If Φ is a computable property and the function τ_Φ is unbounded, then the problem $\#\textsc{IndSub}(\Phi)$ is $\#\mathsf{W}[1]$-hard.*

Note that, by Theorem 10, the problem $\#\textsc{IndSub}(\Phi)$ is fixed-parameter tractable if τ_Φ is bounded. Hence, Conjecture 11 formulates a complete characterization in terms of the subgraph basis expression of a property: For a computable property Φ, the problem $\#\textsc{IndSub}(\Phi)$ is fixed-parameter tractable if τ_Φ is bounded, and $\#\mathsf{W}[1]$-hard otherwise.

To the best of our knowledge, no counterexamples are known for Conjecture 11. Since k-partial determinants can be negative, they cannot be written as $\#\textsc{IndSub}(\Phi)$ for a property Φ, so they do not form a counterexample to our updated conjecture. Moreover, we verify that none of the generalized scorpion properties Ψ_ℓ for fixed $\ell \in \mathbb{N}$ refute it. More specifically, we prove that, for every $k \geq \ell + 4$,

$$\tau_{\Psi_\ell}(k) = \ell + 2. \tag{5}$$

Thus $\tau_{\Psi_\ell}(k) \in O(1)$ for fixed ℓ, so Ψ_ℓ does not satisfy the premises of Conjecture 11.

4.1 Scorpions in the Subgraph Basis

We prove (5) in this section. Recall the definition of tail, sting, body, and legs of an ℓ-scorpion S. We call an ℓ-scorpion S a *skeleton* if its legs form an independent set. Moreover, an *ℓ-scorpion fossil* is any graph S' obtained from an ℓ-scorpion skeleton S by adding an arbitrary

[3] Some did, see a superseded arXiv version of a related paper: `https://arxiv.org/abs/2407.07051v1`

number of edges uv with $u, v \in V(S)$ such that at least one of u, v is not a leg. (Considering Figure 1, a scorpion skeleton is obtained by removing all dashed edges. A scorpion fossil is obtained from a scorpion skeleton by adding arbitrary edges, but not between legs.)

To show (5), we prove the stronger statement that the graphs H occurring with non-zero coefficients $\widehat{\Psi_\ell}(H) \neq 0$ in the subgraph expansion of Ψ_ℓ are precisely the ℓ-scorpion fossils. This is indeed stronger: The vertex-cover number of ℓ-scorpion fossils is at most $\ell + 2$, since they retain the independent set on the legs. On the other hand, the augmented biclique $K^+_{\ell+2,k-\ell-2}$ obtained from a complete bipartite graph by turning the left side (one $\ell + 2$ vertices) into a clique is an ℓ-scorpion fossil of vertex-cover number $\ell + 2$.

▶ **Lemma 12.** *For every $\ell \in \mathbb{N}$ and every k-vertex graph H with $k \geq \ell+4$, we have $\widehat{\Psi_\ell}(H) \neq 0$ if and only if H is an ℓ-scorpion fossil.*

Proof. Let $\mathcal{S} \coloneqq \{S \subseteq E(H) \mid H[S] \in \Psi_\ell\}$. Similar to Theorem 4, we use the uniqueness of body, tail, and sting to partition $\mathcal{S}$ into classes $\mathcal{S}_{\mathbf{q}}$ for $\mathbf{q} \in V(H)^{\ell+2}$. More specifically, for a tuple $\mathbf{q} = (b, t_1, \ldots, t_\ell, s) \in V(H)^{\ell+2}$, let $\mathcal{S}_{\mathbf{q}}$ denote the set of all $S \in \mathcal{S}$ such that the scorpion $H[S]$ has body b, tail $t_1, \ldots, t_\ell$ and sting s. Note that we may have $\mathcal{S}_{\mathbf{q}} = \emptyset$.

By Lemma 3, the sets $\mathcal{S}_{\mathbf{q}}$ for $\mathbf{q} \in V(H)^{\ell+2}$ partition $\mathcal{S}$, which implies

$$\widehat{\Psi_\ell}(H) = (-1)^{|E(H)|} \cdot \sum_{S \in \mathcal{S}} (-1)^{|S|} = (-1)^{|E(H)|} \cdot \sum_{\mathbf{q} \in V(H)^{\ell+2}} \sum_{S \in \mathcal{S}_{\mathbf{q}}} (-1)^{|S|}. \tag{6}$$

In the following, for a tuple $\mathbf{q} \in V(H)^{\ell+2}$, we write $\mathrm{set}(\mathbf{q})$ for the set of entries of $\mathbf{q}$.

▷ Claim 13. If $\mathbf{q} \in V(H)^{\ell+2}$ is such that $\mathrm{set}(\mathbf{q})$ is not a vertex cover of H, then

$$\sum_{S \in \mathcal{S}_{\mathbf{q}}} (-1)^{|S|} = 0.$$

Proof. Let us first observe that the statement is trivial if $\mathcal{S}_{\mathbf{q}} = \emptyset$.

Let $E_{\mathbf{q}} \coloneqq \{e \in E(H) \mid e \cap \mathrm{set}(\mathbf{q}) = \emptyset\}$ be the edges of H without an endpoint in $\mathrm{set}(\mathbf{q})$. Since $\mathrm{set}(\mathbf{q})$ is not a vertex cover, we get $E_{\mathbf{q}} \neq \emptyset$. Observe that $F_{\mathbf{q}} \coloneqq S \setminus E_{\mathbf{q}} = S' \setminus E_{\mathbf{q}}$ for all $S, S' \in \mathcal{S}_{\mathbf{q}}$, because edges incident to body, tail or sting (i.e., edges incident to the vertices in $\mathbf{q}$) are fixed. Since edges between leg vertices can be chosen arbitrarily, we conclude that

$$\mathcal{S}_{\mathbf{q}} = \{F_{\mathbf{q}} \cup E'_{\mathbf{q}} \mid E'_{\mathbf{q}} \subseteq E_{\mathbf{q}}\}.$$

In particular, we obtain

$$\sum_{S \in \mathcal{S}_{\mathbf{q}}} (-1)^{|S|} = \sum_{E'_{\mathbf{q}} \subseteq E_{\mathbf{q}}} (-1)^{|F_{\mathbf{q}}| + |E'_{\mathbf{q}}|} = (-1)^{|F_{\mathbf{q}}|} \sum_{E'_{\mathbf{q}} \subseteq E_{\mathbf{q}}} (-1)^{|E'_{\mathbf{q}}|} = 0$$

since $E_{\mathbf{q}} \neq \emptyset$. ◁

Now, for the forward direction of the lemma, suppose that $\widehat{\Psi_\ell}(H) \neq 0$. Then there is some $\mathbf{q} = (b, t_1, \ldots, t_\ell, s)$ such that $\sum_{S \in \mathcal{S}_{\mathbf{q}}} (-1)^{|S|} \neq 0$. This implies that

(a) $\mathrm{set}(\mathbf{q})$ is a vertex cover of H by Claim 13, so the vertices in $V(H) \setminus \mathrm{set}(\mathbf{q})$ form an independent set in H, and

(b) $\mathcal{S}_{\mathbf{q}} \neq \emptyset$, i.e., there is some $S \subseteq E(H)$ such that $H[S]$ is an ℓ-scorpion with body b, tail $t_1, \ldots, t_\ell$ and sting s. Thus, the vertices in $V(H) \setminus \mathrm{set}(\mathbf{q})$ form the legs of $H[S]$.

These two properties together directly imply that H is an ℓ-scorpion fossil.

For the backward direction, suppose H is an ℓ-scorpion fossil, and let $\mathbf{q} = (b, t_1, \ldots, t_\ell, s)$ denote the tuple of body, tail, and sting of an underlying ℓ-scorpion skeleton. Then

(i) $\mathcal{S}_{\mathbf{q}} \neq \emptyset$,
(ii) $\mathrm{set}(\mathbf{q})$ is a vertex cover of H.

The first item holds because the underlying ℓ-scorpion skeleton is a witness to $\mathcal{S}_{\mathbf{q}} \neq \emptyset$, while the second item holds from the definition of ℓ-scorpion skeletons.

Now, let $A \subseteq V(H)^{\ell+2}$ denote the set of all tuples $\mathbf{q}'$ satisfying (i) and (ii). Then, for every $\mathbf{q}' \in A$ and every $S \in \mathcal{S}_{\mathbf{q}'}$, the legs of $H[S]$ form an independent set, since the vertices in $\mathbf{q}' \in A$ form a vertex-cover. This implies that $H[S]$ is an ℓ-scorpion skeleton, so S contains exactly $k-1$ edges, and we obtain

$$\sum_{S \in \mathcal{S}_{\mathbf{q}'}} (-1)^{|S|} = (-1)^{k-1} |\mathcal{S}_{\mathbf{q}'}| \quad \text{for all } \mathbf{q}' \in A. \tag{7}$$

On the other hand, for $\mathbf{q}' \in V(H)^{\ell+2} \setminus A$, the tuple $\mathbf{q}'$ violates (i), or $\mathbf{q}'$ violates (ii). We obtain directly (in the first case) or via Claim 13 (in the second case) that

$$\sum_{S \in \mathcal{S}_{\mathbf{q}'}} (-1)^{|S|} = 0 \quad \text{for all } \mathbf{q}' \in V(H)^{\ell+2} \setminus A. \tag{8}$$

Let us abbreviate $X_{\mathbf{q}'} \coloneqq \sum_{S \in \mathcal{S}_{\mathbf{q}'}} (-1)^{|S|}$. Then it follows that

$$\begin{aligned}
\widehat{\Psi_\ell}(H) &= (-1)^{|E(H)|} \sum_{\mathbf{q}' \in V(H)^{\ell+2}} X_{\mathbf{q}'} \\
&= (-1)^{|E(H)|} \left(\sum_{\mathbf{q}' \in V(H)^{\ell+2} \setminus A} X_{\mathbf{q}'} + \sum_{\mathbf{q}' \in A} X_{\mathbf{q}'} \right) \\
&= (-1)^{|E(H)|} \left(0 + \sum_{\mathbf{q}' \in A} (-1)^{k-1} |\mathcal{S}_{\mathbf{q}'}| \right) = (-1)^{|E(H)|+k-1} \sum_{\mathbf{q}' \in A} |\mathcal{S}_{\mathbf{q}'}| \neq 0,
\end{aligned}$$

where we used (7) and (8) in the third equality, and where the last inequality holds since $A \neq \emptyset$ and $\mathcal{S}_{\mathbf{q}'} \neq \emptyset$ for all $\mathbf{q}' \in A$. ◀

As discussed above, the lemma implies Equation (5) on the maximum vertex-cover number in the subgraph expansion of the scorpion property. Together with Theorem 10, the upper bound on the vertex-cover number gives an alternative proof that $\#\textsc{IndSub}(\Psi_\ell)$ is fixed-parameter tractable for $\ell \in \mathbb{N}$, with the same polynomial degree as in Theorem 4. We stress that similar arguments also work for the adaptations of the scorpion property discussed in the beginning of Section 4. For example, for the property Λ defined there, we get $\tau_\Lambda(k) \leq 7$ for every $k \in \mathbb{N}$.

4.2 Implications of the Conjecture

To conclude the paper and as a potential avenue for future work, we show that our refined Conjecture 11 would easily imply generalizations and variants of known hardness results.

Most of the recent works (see, e.g., [9, 12, 13, 14, 37]) prove hardness of $\#\textsc{IndSub}(\Phi)$ by identifying graphs with non-zero alternating enumerator and unbounded *treewidth* rather than vertex-cover number; this is a stronger property. More concretely, let $\eta_\Phi(k)$ denote the maximal treewidth among k-vertex graphs H with $\widehat{\Phi}(H) \neq 0$. If Φ is computable and η_Φ is unbounded, then $\#\textsc{IndSub}(\Phi)$ is $\#\mathsf{W}[1]$-hard (see, e.g., [9, Theorem 3.1]). However, showing that η_Φ is unbounded is often quite a challenge. Conjecture 11 postulates that it suffices for τ_Φ to be unbounded to obtain $\#\mathsf{W}[1]$-hardness, which is often much easier to show.

As a concrete example, previous work [12] established hardness of #IndSub(Φ) in the presence of infinitely many primes p with $\Phi(\mathrm{IS}_{2p}) \neq \Phi(K_{p,p})$, where IS_{2p} and $K_{p,p}$ are the edge-less and complete bipartite graphs, respectively. Similarly, for edge-monotone properties Φ, which are studied in [9, 13, 14], we typically obtain $\Phi(\mathrm{IS}_k) \neq \Phi(K_k)$ for all sufficiently large k. As we show below, Conjecture 11 implies that infinitely many *arbitrary* prime-order vertex-transitive graphs H_1, H_2 with $\Phi(H_1) \neq \Phi(H_2)$ suffice for hardness. A graph H is called *vertex-transitive* if for all vertices $u, v \in V(H)$ there exists an automorphism $\varphi \in \mathrm{Aut}(H)$ with $\varphi(u) = v$.

▶ **Proposition 14.** *Let Φ be a graph property such that, for infinitely many primes p, there are vertex-transitive graphs H_p^1, H_p^2 with p vertices and $\Phi(H_p^1) \neq \Phi(H_p^2)$. Then #IndSub($\Phi$) is #W[1]-hard assuming Conjecture 11.*

Proof. For a prime p, we say that a group Γ is a p-group if its order is a power of p, and we establish some facts about groups and graphs:

- **(I)** Each transitive group that operates on a set of p elements contains a transitive p-subgroup. This holds since Γ contains a cyclic group of order p, which is transitive (see the discussion before [29, Theorem 1] for more details).
- **(II)** Each vertex-transitive graph is regular. This is trivial.
- **(III)** Every d-regular k-vertex graph H with $d > 1$ has vertex-cover number $\tau(H) \geq k/2 - 1$: By [34], the independence number $\alpha(H)$ of H is at most $k/2+1$, so $\tau(H) = k - \alpha(H) \geq k/2 - 1$.

By the requirements of the proposition, there are infinitely many primes p such that there are vertex-transitive graphs H_p^1, H_p^2 with p vertices and $\Phi(H_p^1) \neq \Phi(H_p^2)$. Hence, there is an $i \in \{1, 2\}$ with $\Phi(H_p^i) \neq \Phi(\mathrm{IS}_p)$. Let $H_p \coloneqq H_p^i$ be this graph. Then $\mathrm{Aut}(H_p)$ contains a transitive p-subgroup Γ_p by (I). We say a subgraph F_p of H_p is a *fixed point of Γ_p in H_p* if $V(F_p) = V(H_p)$ and $\gamma(E(F_p)) = E(F_p)$ for all $\gamma \in \Gamma_p$, where $\gamma(E(F_p)) \coloneqq \{\gamma(u)\gamma(v) \mid uv \in E(F_p)\}$. Let $\mathrm{FP}(\Gamma_p, H_p)$ denote the set of fixed points of Γ_p in H_p (see [14, Appendix A] for more details on fixed points of group actions in graphs).

We show that there is a fixed point $F_p \in \mathrm{FP}(\Gamma_p, H_p)$ that has a non-zero alternating enumerator and $\tau(F_p) \geq p/2 - 1$. For this, let F_p be a fixed point in $\mathrm{FP}(\Gamma_p, H_p)$ with $\Phi(F_p) \neq \Phi(\mathrm{IS}_p)$ and $\Phi(F_p') = \Phi(\mathrm{IS}_p)$ for all $F_p' \in \mathrm{FP}(\Gamma_p, H_p)$ with $F_p' \subsetneq F_p$ (i.e., F_p' is a proper subgraph of F_p). Such a graph F_p exists since $H_p \in \mathrm{FP}(\Gamma_p, H_p)$ and $\Phi(H_p) \neq \Phi(\mathrm{IS}_p)$. Now, [13, Lemma 4.8] implies that $\widehat{\Phi}(F_p) \neq 0$. Moreover, [13, Lemma 3.1] implies that $\Gamma_p \subseteq \mathrm{Aut}(F_p)$, and hence F_p is vertex-transitive. Lastly, (II) and (III) yield that $\tau(F_p) \geq p/2 - 1$.

This means that we can find a sequence of graphs F_p with unbounded vertex cover number and $\widehat{\Phi}(F_p) \neq 0$, and thus #IndSub($\Phi$) is #W[1]-hard by Conjecture 11. ◀

References

1 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.

2 Christian Borgs, Jennifer Chayes, László Lovász, Vera T. Sós, and Katalin Vesztergombi. Counting graph homomorphisms. In *Topics in discrete mathematics*, volume 26 of *Algorithms Combin.*, pages 315–371. Springer, Berlin, 2006. doi:10.1007/3-540-33700-8_18.

3 Karl Bringmann and Jasper Slusallek. Current algorithms for detecting subgraphs of bounded treewidth are probably optimal. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 40:1–40:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.40.

4 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006. doi:10.1016/J.JCSS.2006.04.007.

5 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223. ACM, 2017. doi:10.1145/3055399.3055502.

6 Radu Curticapean, Simon Döring, Daniel Neuen, and Jiaheng Wang. Can you link up with treewidth? In Olaf Beyersdorff, Michal Pilipczuk, Elaine Pimentel, and Kim Thang Nguyen, editors, *42nd International Symposium on Theoretical Aspects of Computer Science, STACS 2025, March 4-7, 2025, Jena, Germany*, volume 327 of *LIPIcs*, pages 28:1–28:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.STACS.2025.28.

7 Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 130–139. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.22.

8 Radu Curticapean and Daniel Neuen. Counting small induced subgraphs: Hardness via Fourier analysis. *CoRR*, abs/2407.07051, 2024. doi:10.48550/arXiv.2407.07051.

9 Radu Curticapean and Daniel Neuen. Counting small induced subgraphs: Hardness via Fourier analysis. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 3677–3695. SIAM, 2025. doi:10.1137/1.9781611978322.122.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Reinhard Diestel. *Graph Theory*. Springer Berlin, 5 edition, 2017. doi:10.1007/978-3-662-53622-3.

12 Julian Dörfler, Marc Roth, Johannes Schmitt, and Philip Wellnitz. Counting induced subgraphs: An algebraic approach to #W[1]-hardness. *Algorithmica*, 84(2):379–404, 2022. doi:10.1007/S00453-021-00894-9.

13 Simon Döring, Dániel Marx, and Philip Wellnitz. Counting small induced subgraphs with edge-monotone properties. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1517–1525. ACM, 2024. doi:10.1145/3618260.3649644.

14 Simon Döring, Dániel Marx, and Philip Wellnitz. From graph properties to graph parameters: Tight bounds for counting on small subgraphs. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 3637–3676. SIAM, 2025. doi:10.1137/1.9781611978322.121.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

16 Jacob Focke and Marc Roth. Counting small induced subgraphs with hereditary properties. *SIAM J. Comput.*, 53(2):189–220, 2024. doi:10.1137/22M1512211.

17 Leslie Ann Goldberg and Marc Roth. Parameterised and fine-grained subgraph counting, modulo 2. *Algorithmica*, 86(4):944–1005, 2024. doi:10.1007/S00453-023-01178-0.

18 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 657–666. ACM, 2001. doi:10.1145/380752.380867.

19 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.

20 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

21 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978. doi:10.1137/0207033.

22 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting connected subgraphs and graph motifs. *J. Comput. Syst. Sci.*, 81(4):702–716, 2015. doi:10.1016/J.JCSS.2014.11.015.

23 Mark Jerrum and Kitty Meeks. Some hard families of parameterized counting problems. *ACM Trans. Comput. Theory*, 7(3):11:1–11:18, 2015. doi:10.1145/2786017.

24 Emily Jin, Michael M. Bronstein, İsmail İlkan Ceylan, and Matthias Lanzinger. Homomorphism counts for graph neural networks: All about that basis. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL: https://openreview.net/forum?id=zRrzSLwNHQ.

25 Jeff Kahn, Michael E. Saks, and Dean Sturtevant. A topological approach to evasiveness. *Comb.*, 4(4):297–306, 1984. doi:10.1007/BF02579140.

26 Dmitry N. Kozlov. *Combinatorial Algebraic Topology*, volume 21 of *Algorithms and computation in mathematics*. Springer, 2008. doi:10.1007/978-3-540-71962-5.

27 Hendrik Lenstra, Marc R. Best, and Peter van Emde Boas. A sharpened version of the Aanderaa-Rosenberg conjecture. *Report 30/74, Mathematisch Centrum Amsterdam (1974)*, pages 1–20, 1974. URL: https://hdl.handle.net/1887/3792.

28 Ron Milo, Shai S. Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri B. Chklovskii, and Uri Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.

29 Peter M. Neumann. Transitive permutation groups of prime degree. In *Proceedings of the Second International Conference on the Theory of Groups*, volume Vol. 372 of *Lecture Notes in Math.*, pages 520–535. Springer Berlin Heidelberg, 1974. doi:10.1007/978-3-662-21571-5_55.

30 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Comment. Math. Univ. Carolin.*, 26(2):415–419, 1985. URL: http://dml.cz/dmlcz/106381.

31 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Comput. Complex.*, 4:301–313, 1994. doi:10.1007/BF01263419.

32 Ronald L. Rivest and Jean Vuillemin. On recognizing graph properties from adjacency matrices. *Theor. Comput. Sci.*, 3(3):371–384, 1976. doi:10.1016/0304-3975(76)90053-0.

33 Arnold L. Rosenberg. On the time required to recognize properties of graphs: a problem. *SIGACT News*, 5(4):15–16, 1973. doi:10.1145/1008299.1008302.

34 Moshe Rosenfeld. Independent sets in regular graphs. *Israel J. Math.*, 2:262–272, 1964. doi:10.1007/BF02759743.

35 Marc Roth and Johannes Schmitt. Counting induced subgraphs: A topological approach to #W[1]-hardness. *Algorithmica*, 82(8):2267–2291, 2020. doi:10.1007/S00453-020-00676-9.

36 Marc Roth, Johannes Schmitt, and Philip Wellnitz. Detecting and counting small subgraphs, and evaluating a parameterized tutte polynomial: Lower bounds via toroidal grids and cayley graph expanders. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 108:1–108:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.108.

37 Marc Roth, Johannes Schmitt, and Philip Wellnitz. Counting small induced subgraphs satisfying monotone properties. *SIAM J. Comput.*, 53(6):FOCS20–139–FOCS20–174, 2024. doi:10.1137/20M1365624.

38 Benjamin Schiller, Sven Jager, Kay Hamacher, and Thorsten Strufe. StreaM - A stream-based algorithm for counting motifs in dynamic graphs. In Adrian-Horia Dediu, Francisco Hernández Quiroz, Carlos Martín-Vide, and David A. Rosenblueth, editors, *Algorithms for Computational Biology - Second International Conference, AlCoB 2015, Mexico City, Mexico, August 4-5, 2015, Proceedings*, volume 9199 of *Lecture Notes in Computer Science*, pages 53–67. Springer, 2015. doi:10.1007/978-3-319-21233-3_5.

39 Falk Schreiber and Henning Schwöbbermeyer. Frequency concepts and pattern detection for the analysis of motifs in networks. *Trans. Comp. Sys. Biology*, 3:89–104, 2005. doi:10.1007/11599128_7.

A Fast and Simple Algorithm for the Resource Constrained Shortest Path Problem

Saman Ahmadi
School of Engineering, RMIT University, Melbourne, Australia

Andrea Raith
Department of Engineering Science, University of Auckland, New Zealand

Mahdi Jalili
School of Engineering, RMIT University, Melbourne, Australia

Abstract

Constrained pathfinding is a classic yet challenging network optimization problem with broad applicability across many real-world domains. The Resource-Constrained Shortest Path (RCSP) problem focuses on finding cost-optimal paths that satisfy multiple resource constraints. In this paper, we propose a novel heuristic-guided search framework that accelerates constrained search in large-scale networks, including those with negative costs and resources, by leveraging efficient queuing and pruning strategies. Experimental results on real-world benchmark maps show that our framework achieves up to two orders of magnitude speedup over state-of-the-art methods, demonstrating its effectiveness in solving challenging RCSP instances within limited time.

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

Keywords and phrases constrained pathfinding, shortest path problem, heuristic search

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.97

Related Version *arXiv version*: https://doi.org/10.48550/arXiv.2503.11037

Funding This research was supported by the Department of Climate Change, Energy, the Environment and Water under the International Clean Innovation Researcher Networks (ICIRN) program grant number ICIRN000077. Mahdi Jalili is supported by Australian Research Council through projects DP240100963, DP240100830, LP230100439 and IM240100042.

1 Introduction

The Resource Constrained Shortest Path (RCSP) problem is a fundamental yet challenging network optimization problem. It focuses on finding cost-optimal paths between two nodes in a graph while adhering to constraints on the utilization of limited, potentially multidimensional resources. Many real-world problems can be modeled as an RCSP instance. Examples are planning time-constrained explorable non-shortest routes in roaming navigation applications [24], and finding energy efficient paths with detour limits for electric vehicles [4]. As a subproblem, RCSP has been utilized to solve orienteering problems [33], and the vehicle routing problem [12]. RCSP is an NP-Hard problem even for single resource constraint [16].

RCSP and its single-constraint variant, the Weight Constrained Shortest Path (WCSP) problem, have been studied in the literature for decades. Traditional solutions to RCSP are generally based on path ranking, dynamic programming (labeling) or branch-and-bound (B&B) approaches [25, 13], but recent studies have demonstrated that the labeling method is more effective for solving large-scale RCSP instances. The recent RCBDA* algorithm [31] is a labeling method that utilizes the dynamic programming approach presented in [29] to perform a bidirectional A* search guided by cost heuristics, using rigorous yet costly pruning rules to avoid exploring unpromising paths. Complete paths in this algorithm are


© Saman Ahmadi, Andrea Raith, and Mahdi Jalili;
licensed under Creative Commons License CC-BY 4.0

obtained by matching backward and forward partial paths, and the amount of a critical resource available to each search is half of the total budget. The authors concluded that RCBDA* can perform faster than two other existing RCSP approaches on many instances, namely the Pulse algorithm (B&B method) [22] and the CSP (path ranking) approach in [30]. A parallel bidirectional search framework on the basis of Pulse was later developed in [10], known as BiPulse. This algorithm utilizes the the queuing mechanism developed in [9] to limit the depth of the Pulse search and postpone the exploration of deep branches of the search tree, where halted partial paths are explored in a breadth-first search manner. The authors reported better performance and more solved cases with BiPulse compared to RCBDA*. Both BiPulse and RCBDA* received awards for their contribution to RCSP [15]. WCEBBA* [3], WCA* [6] and WCBA* [5] are three other A*-based methods that leverage fast pruning rules in recent bi-objective search literature [32, 2] to tackle constrained pathfinding more efficiently. Although all of the algorithms have been shown to outperform RCBDA* and BiPulse on large graphs, their application is limited to instances with a single resource constraint. Another recent A*-based RCSP method, called ERCA* [27], takes advantage of recent advancements in multi-objective search with A* [26, 28] to improve the efficiency of its pruning rules through binary search trees. Like other A*-based methods, the search in ERCA* is guided by cost heuristics and prunes unpromising paths. Its pruning involves rigorously discarding any path that exceeds the resource budget or is dominated (i.e., no better in cost and resource usage) by another previously extended path to the same node, both before and during path extension. The authors reported several orders of magnitude faster runtime for ERCA* when compared with BiPulse. However, the performance of ERCA* against RCBDA* remains unknown. All of the above algorithms are designed and evaluated for problem instances with non-negative cost and resources.

Contributions. This paper presents a novel and straightforward label-setting framework built upon A*, called NWRCA*, which incorporates effective search and lazy pruning strategies to accelerate constrained pathfinding in large-scale networks with negative costs and resources. We evaluate the framework in two variants that differ slightly in how they maintain search labels, and we investigate a faster method for computing A*'s heuristics that can potentially replace the conventional reweighting approach when dealing with negative-weight graphs. Extensive experiments demonstrate the success of the framework in solving difficult RCSP instances several orders of magnitude faster than ERCA* and RCBDA*.

2 Problem Definition

Consider an RCSP problem with d resource limits $\{R_1, R_2, \ldots, R_d\}$ is provided as a directed graph G with a set of vertices V and a set of edges $E \subseteq V \times V$. Every edge $e \in E$ of the graph has $d + 1 \in \mathbb{N}$ attributes, represented as $(cost, \mathbf{resources})$ where $\mathbf{resources} = \{resource_1, \ldots, resource_d\}$ is a form of vector (denoted boldface). An acyclic path π consists of n distinct vertices $\{v_1, v_2, v_3, \ldots, v_n\}$, where $n \leq |V|$ and $(v_i, v_{i+1}) \in E$ for $i \in \{1, \ldots, n-1\}$. The cost and resource usage of a path π are determined by summing the attributes of all edges that make up the path. The RCSP problem aims to find (at least) one *cost*-optimal paths π^* from $v_s \in V$ (start or origin) to $v_g \in V$ (goal or destination) such that the resource usages of the path are within the given limits, that is, we must have $resource_k(\pi^*) \leq R_k$ for each resource $k \in \{1, \ldots, d\}$.

In line with the notation in the RCSP literature, we refer to our search objects as *labels*. Each label l encapsulates the key information about a partial path from v_s and is associated with an end vertex $v \in V$ (last vertex of the path). Label l traditionally stores value pairs

g_l and $\mathbf{r}_l$ that measure the *cost* and **resources** of the path, respectively. Additionally, l includes p_l, a reference to the parent label of l, and f_l, an estimate of the *cost* of a complete path to v_g by extending l. Specifically, for each search label l associated with vertex v, we have $f_l = g_l + h_c(v)$ where $h_c(v)$ is a consistent *cost* heuristic function [17].

▶ **Definition 1.** *The heuristic function $h_c : V \to \mathbb{R}$ is admissible iff $h_c(v) \leq cost(\lambda^*)$ for every $v \in V$, where λ^* is a cost-optimal path from v to v_g. h_c is consistent if we have $h_c(v) \leq cost(v, u) + h_c(u)$ for every $(v, u) \in E$.*

We assume all operations on the resource vectors are performed element-wise. In addition, we use the $\preceq$ symbol for direct comparisons of resource vectors, e.g., $\mathbf{r}_l \preceq \mathbf{r}_{l'}$ denotes $resource_k$ of label l is no greater than that of label l' for all $k \in \{1, \ldots, d\}$. Analogously, we use the $\not\preceq$ symbol if the relation cannot be satisfied. We now define *dominance* over labels.

▶ **Definition 2.** *Label l' is weakly dominated by label l if we have $g_l \leq g_{l'}$ and $\mathbf{r}_l \preceq \mathbf{r}_{l'}$; l' is strongly dominated by l if $g_l \leq g_{l'}$ and $\mathbf{r}_l \preceq \mathbf{r}_{l'}$ but $g_l \neq g_{l'}$ or $\mathbf{r}_l \neq \mathbf{r}_{l'}$; l' is not dominated by l if $g_l \not\leq g_{l'}$ or $\mathbf{r}_l \not\preceq \mathbf{r}_{l'}$.*

3 Resource Constrained Pathfinding with A*

Constrained pathfinding with A* involves a systematic search by exploring labels in best-first order, meaning that the search is guided by the partial path with the smallest *cost* estimate or f-value. Each iteration involves three main steps:
i) Extraction: remove one least-*cost* label from a priority queue $\mathcal{Q}$, also known as *Open* list.
ii) Dominance pruning: skip exploring labels that are weakly dominated by at least one other label (with the same vertex);
iii) Expansion: generate new descendant labels, check them for feasibility (against resource limits), and, if not pruned, store them in $\mathcal{Q}$ for further expansion.
The search in this framework generally terminates once an optimal solution is found, or there is no label in $\mathcal{Q}$ to explore, where the latter means there is no feasible solution. If there is only one resource, it is shown that dominance pruning can be done in constant time when performing best-first search [5, 6]. Nonetheless, this is not the case in RCSP with multiple resources and the dominance pruning remains a costly task.

RCSP with negative weights. Many real-world RCSP problems deal with attributes that are negative in nature, such as energy recuperation in electric vehicles, or attributes that may exhibit negative values in specific circumstances, such as negative reduced costs in column generation or pricing problems [19]. Although recent A*-based RCSP solutions, such as ERCA* and RCBDA*, are primarily designed and evaluated for problem instances with non-negative edge attributes, they can be adapted to be used for instances with negative weights (but no negative cycles). This can be achieved through a graph reformulation phase, where all edge weights are shifted to non-negative values, for example, using Johnson's reweighting technique [21]. However, as we will show in this section, our constrained search framework can be applied directly to graphs with negative edge weights, provided that its *cost* heuristic is consistent. With this introduction, we now describe our new RCSP algorithm.

Our approach. Algorithm 1 presents the pseudocode of NWRCA*, our A*-based RCSP method that employs *lazy* dominance checks and can handle problem instances with negative edge weights (but no negative cycles). The algorithm begins by initializing a priority queue $\mathcal{Q}$ and a global cost upper bound $\overline{f}$, which maintains the best-known cost of any feasible solution found during the search. For each vertex $v \in V$, it also initializes a list $\mathcal{X}(v)$ to store

Algorithm 1 NWRCA*.

Input: An RCSP Problem $(G, v_s, v_g, \mathbf{R})$
Output: A set of non-dominated *cost*-optimal solution labels

1 $\mathcal{Q} \leftarrow \emptyset$, $\overline{f} \leftarrow \infty$
2 $\mathcal{X}(v) \leftarrow \emptyset \ \ \forall v \in V$
3 $h_c(v) \leftarrow$ optimal *cost* from v to v_g $\forall v \in V$
4 $\mathbf{h}_r(v) \leftarrow$ lower bound on **resources** from v to v_g $\forall v \in V$
5 $l \leftarrow$ new label associated with v_s
6 $g_l \leftarrow 0$, $\mathbf{r}_l \leftarrow \mathbf{0}_d$, $f_l \leftarrow h_c(v_s)$, $p_l \leftarrow \varnothing$
7 add l to $\mathcal{Q}$
8 **while** $\mathcal{Q} \neq \emptyset$ **do**
9 extract from $\mathcal{Q}$ label l with the smallest f-value
10 **if** $f_l > \overline{f}$ **then break**
11 $v \leftarrow$ vertex associated with the extracted label l
12 *dominated* $\leftarrow$ false
13 **for each** $l' \in \mathcal{X}(v)$ **do**
14 **if** $\mathbf{r}_{l'} \preceq \mathbf{r}_l$ **then**
15 *dominated* $\leftarrow$ true
16 **break**
17 **if** *dominated* = true **then continue**
18 **for each** $l' \in \mathcal{X}(v)$ **do**
19 **if** $\mathbf{r}_l \preceq \mathbf{r}_{l'}$ **then** remove l' from $\mathcal{X}(v)$
20 add l to $\mathcal{X}(v)$
21 **if** $v = v_g$ **then**
22 $\overline{f} \leftarrow f_l$
23 **continue**
24 **for each** successor u of v **do**
25 $l' \leftarrow$ new label associated with u
26 $g_{l'} \leftarrow g_l + cost(v, u)$
27 $\mathbf{r}_{l'} \leftarrow \mathbf{r}_l +$ **resources**(v, u)
28 $f_{l'} \leftarrow g_{l'} + h_c(u)$
29 $p_{l'} \leftarrow l$
30 **if** $\mathbf{r}_{l'} + \mathbf{h}_r(u) \not\preceq \mathbf{R}$ **then continue**
31 add l' to $\mathcal{Q}$
32 **return** $\mathcal{X}(v_g)$

the non-dominated labels resulting from all prior expansions (i.e., closed labels) associated with vertex v. Like other A*-based RCSP algorithms, NWRCA* requires a *cost* heuristic function h_c to establish f-values. To enhance pruning of unpromising paths, we also compute lower bounds on **resources** using an admissible heuristic function $\mathbf{h}_r : V \to \mathbb{R}^d$. These cost and resource heuristics can be computed using $1 + d$ single-objective Dijkstra runs [11], with re-expansions allowed to accommodate negative weights [20], assuming there are no negative cycles on any path from v_s to v_g in any dimension (cost or resources). A similar strategy has been explored in [1] for multi-objective search. As we will demonstrate in the experimental section, this heuristic construction method, despite its exponential worst-case complexity, often outperforms alternatives such as Johnson's reweighting technique [21] in practice. NWRCA* then initializes a label with v_s and inserts it into $\mathcal{Q}$. Each iteration of the main search starts at line 8. Let $\mathcal{Q}$ be a non-empty queue. The algorithm extracts in each iteration a label l with the smallest f-value (line 9) and attempts the following steps.

Lazy dominance check. NWRCA* borrows a lazy label pruning technique from the multi-objective search literature and delays dominance check of newly generated labels until they are extracted from $\mathcal{Q}$, as in [18] and [1]. Let l be a label extracted from $\mathcal{Q}$ with f-value within the global upper bound. Also let v be the end vertex associated with l. Because the first dimension is always explored in sorted f order (guaranteed in A* with a consistent heuristic), we observe that l is already weakly dominated by previous expansions in terms of *cost*. Note that all expansions with v use $h_c(v)$ as lower bound. As a result, all prior expansions of v must have exhibited g-value no smaller than g_l. This means that the dominance test can only be done by comparing the resource vector of l, that is, $\mathbf{r}_l$, with that of (potentially all) previous expansions stored in $\mathcal{X}(v)$. This process can be further improved by prioritizing the most recent expansion of v as the first (potentially more informed) candidate for a quick dominance check, prior to a more rigorous dominance check against the labels of $\mathcal{X}(v)$ [1]. The expansion of label l can be skipped if it is found to be dominated by any previously expanded label (lines 13-16). If l is determined to be non-dominated, NWRCA* removes from the $\mathcal{X}(v)$ list any label l' whose resource vector is weakly dominated by $\mathbf{r}_l$ (lines 18-19), and then adds l to $\mathcal{X}(v)$ for future dominance checks at vertex v (line 20). This removal is correct because any future label that would be dominated by l' will also be dominated by l, making it unnecessary to keep the now-redundant label in $\mathcal{X}(v)$.

Capturing solutions. A tentative solution path is found if the search extracts a non-dominated label associated with the goal vertex v_g (line 21). Expansion of solution labels is not necessary. However, in the absence of tie-breaking in $\mathcal{Q}$, new solutions may offer better resources than previous ones. NWRCA* takes care of this situation by maintaining only non-dominated labels associated with the goal vertex in $\mathcal{X}(v_g)$.

Expansion. l's expansion involves generating new descendant labels through v's successors (adjacent vertices). Given $\mathbf{h}_r$ as an admissible heuristic function, descendant labels with estimated resource usage exceeding the given limit $\mathbf{R}$ cannot lead to a feasible solution and can be pruned (line 30). Note that in the case of negative resources, new labels whose usage of **resources** exceeds the budget (i.e., $\mathbf{r}_{l'} \not\preceq \mathbf{R}$) can be pruned, provided that all partial paths are also required to satisfy the resource constraints (which is not assumed in this work). As part of NWRCA*'s lazy dominance pruning rule, newly generated labels are not checked for dominance against previously expanded labels or those still in $\mathcal{Q}$; instead, they are only checked when extracted from the queue. Thus, each descendant label will be added to $\mathcal{Q}$ if its estimated usage of **resources** to v_g is within $\mathbf{R}$.

Termination. NWRCA* explores labels based on their f-value in ascending order. Given $\overline{f}$ as global *cost* upper bound, the search can terminate safely once it extracts a label with f-value greater than $\overline{f}$ (line 10). Although this upper bound is initially unknown, it will get updated as soon as a feasible solution is discovered (line 22). The algorithm returns $\mathcal{X}(v_g)$, a set containing *cost*-optimal labels with non-dominated **resources** (line 32). If there is no solution to a given instance (i.e., v_g is never reached), this set would remain empty.

Example. We further elaborate on the key steps of NWRCA* by solving a sample RCSP instance with three edge attributes (two resources), depicted in Figure 1. We assume resource limits are $(R_1, R_2) = (3, 3)$. The vertices v_s and v_g denote *start* and *goal*, respectively. The graph is free of negative cycles, and the values inside each vertex denote h_c and (h_{r1}, h_{r2}), calculated prior to the main search via three single-objective backward searches. We briefly

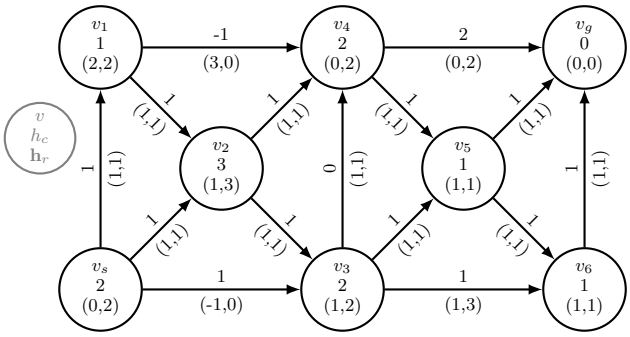


Figure 1 An RCSP instance with *cost* above and **resources** below edges ($d = 2$). Values inside the nodes denote vertex identifier, h_c and $\mathbf{h}_r$ of each vertex, listed from top to bottom.

explain all iterations (It.) of NWRCA* for the given instance, with the trace of $\mathcal{Q}$ and $\mathcal{X}$ sets provided in Table 1. Pruned labels of each iteration are shown in the third column. For label extractions, we adopt the Last-In, First-Out (LIFO) strategy in the event of ties in f-values of labels present in $\mathcal{Q}$. The queue is initialized with the first label l_1 associated with v_s.

- **It.1:** The search starts with expanding the initial label l_1. Since l_1 is the first expansion of v_s and thus non-dominated, it is stored in $\mathcal{X}(v_s)$. Expansion of l_1 generates three new labels: l_2, l_3, and l_4. l_2 and l_4 are added to $\mathcal{Q}$ but l_3 is pruned as its resource estimates are out of bounds, i.e., we have $\mathbf{r}_{l_3} + \mathbf{h}_r(v_2) \not\preceq \mathbf{R}$.
- **It.2:** There are two labels in $\mathcal{Q}$. l_2 has the smallest f-value and is expanded first. l_2 is non-dominated and its expansion generates l_5 and l_6 with v_2 and v_4, respectively. Both labels are pruned since their resource estimates are out of bounds.
- **It.3:** l_4, associated with v_3, is the only label in $\mathcal{Q}$ and is extracted. l_4 will be stored in $\mathcal{X}(v_3)$ as a non-dominated label. l_4's expansion generates three new labels: l_7, l_8, and l_9. However, only two of these are within the resource bounds and can be added to $\mathcal{Q}$. l_9 is out of bounds as we have $3 + 1 \not\leq 3$ for its second resource estimate.
- **It.4:** There are two labels in $\mathcal{Q}$, both with the same f-value. l_8 is extracted based on the LIFO strategy. l_8 is non-dominated, as this is the first expansion with v_4. Expansion of l_8 generates two new labels (l_{10} and l_{11}, both within the bounds) and adds them to $\mathcal{Q}$.
- **It.5:** l_{11} is extracted based on the LIFO strategy. l_{11} is non-dominated and its expansion generates two labels l_{12} and l_{13}, but only l_{12} is added into $\mathcal{Q}$ as the second resource estimate for l_{13} is not within the bound, i.e., we have $3 + 1 \not\leq 3$.
- **It.6:** There are three labels in $\mathcal{Q}$, all with the same f-value. The recent insertion l_{12} is extracted with v_g. This yields the first solution path, followed by updating $\overline{f} \leftarrow 3$. l_{12} does not get expanded.
- **It.7:** l_{10} is extracted, and it appears as a non-dominated solution. However, l_{10}'s resource vector $(0, 3)$ dominates that of the previous solution. Thus, l_{10} replaces l_{12} in $\mathcal{X}(v_g)$.
- **It.8:** l_7 is extracted. l_7 is a non-dominated label but dominates and replaces l_{11} in $\mathcal{X}(v_5)$. Expansion of l_7 generates two new labels (l_{14} and l_{15}). Both labels are within the resource bounds and thus added into $\mathcal{Q}$.
- **It.9:** l_{14} is extracted with v_g. l_{14} is non-dominated and is added to $\mathcal{X}(v_g)$. l_{14} does not dominate the previous solution.
- **It.10:** l_{15} is extracted. Given the *cost* upper bound updated in **It.6**, the search terminates due to $f_{l_{15}}$ surpassing the upper bound $\overline{f}$, i.e., we have $f_{l_{15}} = 4 > 3$.

Table 1 Tracking $\mathcal{Q}$ and $\mathcal{X}$ lists in each iteration (**It.**) of NWRCA* for Figure 1. The symbol $^{\uparrow}$ denotes the extracted label, and the last column shows pruned labels during the expansion phase.

It.	$\mathcal{Q}:\ [f_l, g_l, \mathbf{r}_l, v_l, p_l]$	$\mathcal{X}$	Generated but Pruned
1	$^{\uparrow}l_1 = [2, 0, (0,0), v_s, \varnothing]$	$\mathcal{X}(v_s) = \{l_1\}$	$l_3 = [4, 1, (1,1), v_2, l_1]$
2	$^{\uparrow}l_2 = [2, 1, (1,1), v_1, l_1]$ $l_4 = [3, 1, (-1,0), v_3, l_1]$	$\mathcal{X}(v_1) = \{l_2\}$	$l_5 = [5, 2, (2,2), v_2, l_2]$ $l_6 = [2, 0, (4,1), v_4, l_2]$
3	$^{\uparrow}l_4 = [3, 1, (-1,0), v_3, l_1]$	$\mathcal{X}(v_3) = \{l_4\}$	$l_9 = [3, 2, (0,3), v_6, l_4]$
4	$l_7 = [3, 2, (0,1), v_5, l_4]$ $^{\uparrow}l_8 = [3, 1, (0,1), v_4, l_4]$	$\mathcal{X}(v_4) = \{l_8\}$	
5	$l_7 = [3, 2, (0,1), v_5, l_4]$ $l_{10} = [3, 3, (0,3), v_g, l_8]$ $^{\uparrow}l_{11} = [3, 2, (1,2), v_5, l_8]$	$\mathcal{X}(v_5) = \{l_{11}\}$	
6	$l_7 = [3, 2, (0,1), v_5, l_4]$ $l_{10} = [3, 3, (0,3), v_g, l_8]$ $^{\uparrow}l_{12} = [3, 3, (2,3), v_g, l_{11}]$	$\mathcal{X}(v_g) = \{l_{12}\}$	
7	$l_7 = [3, 2, (0,1), v_5, l_4]$ $^{\uparrow}l_{10} = [3, 3, (0,3), v_g, l_8]$	$\mathcal{X}(v_g) = \{l_{10}\}$	
8	$^{\uparrow}l_7 = [3, 2, (0,1), v_5, l_4]$	$\mathcal{X}(v_5) = \{l_7\}$	
9	$^{\uparrow}l_{14} = [3, 3, (1,2), v_g, l_7]$ $l_{15} = [4, 3, (1,2), v_6, l_7]$	$\mathcal{X}(v_g) = \{l_{10}, l_{14}\}$	
10	$^{\uparrow}l_{15} = [4, 3, (2,3), v_6, l_{11}]$		

The example above shows how NWRCA* processes labels in the order of their f-value. As we observed in **It.6**, NWRCA* may capture a dominated solution in the absence of tie-breaking, but we discussed in **It.7** how the search refines the set $\mathcal{X}(v_g)$ in such circumstances to keep **resources** of solution paths non-dominated. We now briefly discuss the key differences of NWRCA* with two recent RCSP approaches, namely ERCA* and RCBDA*.

NWRCA* vs. ERCA*. Although both algorithms utilize best-first search to guide their constrained search, they differ in four key aspects regarding how labels are handled: (1) ERCA* checks labels for both dominance and resource usage once during expansion and again before expansion, whereas NWRCA* performs upper bound pruning only during expansion and dominance check only before expansion; (2) ERCA* explores labels in lexicographical order of their **resources** in case of ties between f-values in the queue (i.e., it compares labels based on resources in order, starting from the first resource), whereas NWRCA* does not perform tie-breaking; (3) ERCA* uses a specialized and relatively complex data structure (balanced binary search tree) to organize and dominance check labels during the search, whereas expanded labels in NWRCA* are stored in simple lists; (4) ERCA* immediately terminates upon finding a feasible solution, whereas NWRCA* terminates with all non-dominated solutions.

NWRCA* vs. RCBDA*. The algorithms differ in four key aspects: (1) RCBDA* conducts a bidirectional A* search, requiring a specialized and time-intensive procedure to handle frontier collisions (joining bidirectional labels), whereas NWRCA* performs a simple unidirectional search. (2) RCBDA* checks new labels for dominance against all previous expansions of the

vertex, whereas NWRCA* only checks against labels with non-dominated **resources**; (3) RCBDA* checks labels for dominance only during expansion, which can lead to the expansion of dominated labels, whereas NWRCA* performs dominance checks before expansion; (4) Similar to ERCA*, RCBDA* does not compute all non-dominated solutions.

4 Theoretical Results

This section provides a formal proof for the correctness of constrained search of NWRCA* and presents theoretical results on why the algorithm can solve RCSP instances with negative weights but no negative cycles. Throughout this section, we assume there is no negative cycle on any path from start vertex (v_s) to goal vertex (v_g) in any dimension, and thus consistent and admissible heuristic functions h_c and $\mathbf{h}_r$ can be computed for the problem instance.

▶ **Lemma 3.** *Assume the constrained A* search is guided by smallest (potentially negative) f-values. Let l_i and l_{i+1} be labels extracted from $\mathcal{Q}$ in two consecutive iterations of the search. We have $f_{l_i} \leq f_{l_{i+1}}$ if h_c is consistent.*

Proof. We distinguish two cases: i) if l_{i+1} was available in $\mathcal{Q}$ at the time l_i was extracted, the lemma is trivially true. ii) otherwise, l_{i+1} is the descendant label of l_i. For an edge (v, u) linking l_i to its descendant l_{i+1}, h_c's consistency ensures $h_c(v) \leq h_c(u) + cost(v, u)$. Adding the cost g_{l_i} to both sides of the inequality yields $f_{l_i} \leq f_{l_{i+1}}$. ◀

▶ **Corollary 4.** *Consider the sequence of labels $(l_1, l_2, ..., l_t)$ extracted from $\mathcal{Q}$. If h_c is consistent, then according to the conditions of Lemma 3, $i \leq j$ guarantees that $f_{l_i} \leq f_{l_j}$. In other words, the f-values of the extracted labels do not decrease throughout the process.*

▶ **Lemma 5.** *Suppose l' is extracted after l, both with the same vertex. l weakly dominates l' if $\mathbf{r}_l \preceq \mathbf{r}_{l'}$.*

Proof. Since l is extracted before l', we have $f_l \leq f_{l'}$ according to Corollary 4, and consequently $g_l \leq g_{l'}$. The other condition $\mathbf{r}_l \preceq \mathbf{r}_{l'}$ means that l' is no smaller than l in all resources, verifying l' is weakly dominated by l. ◀

▶ **Lemma 6.** *Let l' be a label weakly dominated by a previously explored label l, both associated with the same vertex. The expansion of l' is not necessary.*

Proof. We prove this by contradiction, assuming that l's expansion is necessary to obtain a *cost*-optimal and non-dominated solution path. As l weakly dominates l', it holds that $g_l \leq g_{l'}$ and $\mathbf{r}_l \preceq \mathbf{r}_{l'}$. In this scenario, the partial path represented by l' can be replaced with the one represented by l, resulting in a path that is better than or equal to the optimal solution path obtained via l'. If the resulting path is better, it contradicts the assumption of l''s expansion leading to a non-dominated or optimal solution. If both paths are identical in terms of *cost* and **resources**, it becomes clear that the expansion of l has been sufficient, and thus expanding l' is unnecessary, again contradicting the assumption. Therefore, we conclude that expanding the weakly dominated label l' is not necessary. ◀

▶ **Lemma 7.** *The expansion of label l is not necessary if $f_l \not\leq \overline{f}$ or $\mathbf{r}_l + \mathbf{h}_r \not\preceq \mathbf{R}$.*

Proof. Given that $\mathbf{h}_r$ is admissible, the second condition ensures that expanding l towards v_g cannot yield a solution within the resource limits. Similarly, since h_c is admissible, the first condition guarantees that l's expansion cannot produce a solution with a *cost* better than the best-known upper bound $\overline{f}$. Therefore, expanding l cannot contribute to any optimal solution path, and thus is not necessary. ◀

▶ **Theorem 8.** *NWRCA* produces all cost-optimal solution paths with non-dominated resources for the RCSP problem.*

Proof. The algorithm explores all feasible search labels from v_s towards v_g in best-first order, in search of all optimal solutions. Labels are checked for dominance (Lemma 5) before expansion, and weakly dominated ones can be safely pruned, as they offer no improvement over previously expanded labels (with the same vertex) in either *cost* or **resources** (Lemma 6). Labels violating the upper bounds can similarly be pruned safely (Lemma 7). The algorithm also retains only non-dominated labels in the $\mathcal{X}$ lists. This procedure is correct because if a recently extracted label l' weakly dominates a previously explored label l, then any future label l'' that would have been weakly dominated by l is already weakly dominated by l', rendering the storage of l redundant. Building on the above, we just need to show that the algorithm terminates with returning non-dominated labels. NWRCA* prunes all weakly dominated labels, but captures all non-dominated labels with v_g. However, since it does not process labels in any specific order of their **resources**, some tentative solutions may later appear dominated. Let l' be a new non-dominated solution extracted after solution l. We must have $f_l = f_{l'}$, otherwise the algorithm was terminated if $f_{l'} > \overline{f} = f_l$. In this case, the dominance check in lines 18-19 of Algorithm 1 removes l from $\mathcal{X}(v_g)$ if it is deemed to be weakly dominated by new solution l' (in terms of **resources**). Once the search exceeds $\overline{f}$, Corollary 4 ensures that expanding the remaining labels in $\mathcal{Q}$ cannot produce solutions with a primary cost better than $\overline{f}$. Thus, the termination criterion is correct, and NWRCA* returns $\mathcal{X}(v_g)$ as the set of all *cost*-optimal, non-dominated solution labels. ◀

5 Empirical Analysis

We evaluate the constrained search performance of NWRCA* against two recent A*-based RCSP approaches: the award-winning RCBDA* algorithm [31] and ERCA* [27]. We did not consider BiPulse [10] as prior work [27] has demonstrated that it is outperformed by ERCA*. The algorithms were evaluated on 600 instances across three maps (NY, BAY, and COL) from the 9th DIMACS Implementation Challenge[1], featuring road networks with 264K to 435K vertices and 733K to 1M edges. For each map, we generated 25 random (v_s,v_g) pairs to be evaluated on four levels of tightness, resulting in 100 test cases. We study each test case in scenarios with two and three resources ($d = 2, 3$). Following the RCSP literature, we define each resource limit R_k for $k \in \{1, \ldots, d\}$ based on the tightness of the constraint τ as:

$$\tau = \frac{R_k - h_{rk}}{ub_{rk} - h_{rk}} \quad \text{for } \tau \in \{20\%, 40\%, 60\%, 80\%\}$$

where ub_{rk} and h_{rk} represent the upper and lower bounds on $resource_k$ for paths from v_s to v_g, respectively. The upper bound is set by the (non-constrained) *cost*-optimal path. We choose the same τ for all resources.

To evaluate the algorithms on graphs with non-correlated but negative weights, following our previous work [1], we first enriched each map with Shuttle Radar Topography Mission[2] height data, and set the edge weights of each map with four new attributes as follows. The *cost* of each edge is the energy requirement of an electric vehicle with three passengers on board, using the energy model in [7]. The calculated energy can be negative in some downhill links due to energy recuperation. We deliberately did not impose a bound on

[1] http://www.diag.uniroma1.it/challenge9
[2] https://www2.jpl.nasa.gov/srtm/

battery capacity, enabling the use of plain constrained search for computing energy-efficient paths. We chose a penalized height function for $resource_1$. Given $height(v)$ as elevation of vertex v, the second attribute of link $(v,u) \in E$ is set to $2 \times (height(u) - height(v))$ for uphill links, and $height(u) - height(v)$ for downhill links (negative). For $resource_2$ and $resource_3$, we employed the (reversed) Johnson's technique to generate cycle-free but random negative weights in two steps: i) we first assigned a random integer in the [-100,0] range, known as potential, to each vertex; ii) we then set the third cost of each link $(v,u) \in E$ to be the potential difference between v and u, which was then added by a random integer in the [0, 10] range. The result is two sets of randomized negative weights in the [-100, 110] range with no negative cycle, which ensures the applicability of the selected algorithms over the instances. Note that the first and second dimensions cannot contain negative cycles due to the nature of the chosen metrics. The average percentage of negative weights observed across the maps for $[cost, resrource_1, resrource_2, resrource_3]$ is [10, 42, 45, 45]%, respectively.

Implementation. We used the publicly available C++ implementations of ERCA* and developed an improved version of RCBDA* in C++ based on its provided description. In doing so, we addressed a potential inefficiency in the design related to the label matching strategy of RCBDA*: rather than storing all expanded labels in a large pool and searching for complimentary labels (which adds unnecessary overhead in identifying the same matching vertex), we allocated a separate set for each vertex, optimizing the matching of partial paths during the bidirectional search. Additionally, the original description of RCBDA* does not include any dominance or infeasibility pruning rules. Nonetheless, our implementation incorporates these strategies, thereby improving its search efficiency. Both RCBDA* and ERCA* were provided with reweighted graphs (and budgets), ensuring that instances can be handled correctly in the presence of negative weights. We implemented our NWRCA* algorithm in C++, providing two variants that differ slightly in their label ordering mechanisms:

- NWRCA*$_{v1}$: (i) labels in $\mathcal{X}$ lists are maintained in lexicographical order based on their **r**-value, (ii) unexplored labels are ordered in a bucket-based queue and extracted using a LIFO strategy in the event of ties in their f-values (as in WCA* [6]).
- NWRCA*$_{v2}$: (i) labels in $\mathcal{X}$ lists are not maintained in any specific order, (ii) unexplored labels are ordered in a binary heap queue and extracted in lexicographical order of their resource estimates (i.e., $\mathbf{r} + \mathbf{h}_r$) in the event of ties in their f-values.

These variants allow us to investigate the impact of lexicographical label ordering in both $\mathcal{X}$ and $\mathcal{Q}$ lists on search performance. Note that the correctness of NWRCA* does not depend on labels being ordered lexicographically by *cost* and **resources** in $\mathcal{Q}$. Keeping non-dominated labels in lexicographical order within each $\mathcal{X}$ list in the first variant allows for partial traversal over labels of the list for both dominance checking and removal of dominated labels, but it incurs an additional sorting overhead (see [1] for more details). In contrast, the second variant provides faster insertion/removal of labels to/from the list (with minimal ordering overhead), but requires two linear scans over all labels in $\mathcal{X}$, one to ensure that a newly extracted label is non-dominated and another to remove any dominated labels from the list.

We compiled all C++ code using the GCC7.5 compiler and optimization level O3. The experiments were conducted on a single core of an Intel Xeon Platinum 8175 processor, clocked at 2.5GHz, with 16GB of RAM and a one-hour time limit per run. It is worth mentioning that RCBDA* and both variants of NWRCA* were implemented within the same framework, using the same data structures to manage labels. Therefore, the performance comparisons between the algorithms are head-to-head. Our code is publicly available[3].

[3] `https://bitbucket.org/s-ahmadi/multiobj`

Table 2 Average reweighting and lower-bounding time in each map with three resources ($d = 3$).

	Reweighting time (s)			Lower bounding (s)		
Approach	NY	BAY	COL	NY	BAY	COL
Graph Reformulation (Johnson)	1.64	2.92	4.02	0.41	0.50	0.54
Modified Dijkstra	-	-	-	0.52	1.00	1.02

Table 3 The runtime statistics of the algorithms (in seconds) for $d = 2, 3$. For unsolved instances, the runtime is considered to be one hour. $|\mathcal{P}|$ denotes the number of instances solved within the timeout, and ϕ is the average slowdown factor of the mutually solved instances w.r.t. NWRCA*$_{v1}$.

			Search time (s)					Search time (s)			
Map	Algorithm	$\lvert\mathcal{P}\rvert$	Mean_A	Mean_G	Max.	ϕ	$\lvert\mathcal{P}\rvert$	Mean_A	Mean_G	Max.	ϕ
NY	NWRCA*$_{v1}$	100	**1.01**	**0.24**	**11.0**	-	100	**33.29**	**2.89**	**461.0**	-
	NWRCA*$_{v2}$	100	3.24	0.59	38.1	2.5	100	108.21	7.73	1607.3	2.5
	ERCA*	100	66.04	11.46	765.5	50.5	67	1513.00	217.35	3600.0	93.7
	RCBDA*	100	18.91	1.74	31.1	12.1	86	803.41	44.62	3600.0	44.8
BAY	NWRCA*$_{v1}$	100	**3.31**	**0.54**	**31.1**	-	100	**125.21**	**8.65**	**1803.5**	-
	NWRCA*$_{v2}$	100	10.20	1.31	124.6	2.4	96	335.69	22.84	3600.0	2.5
	ERCA*	95	306.88	27.28	3600.0	50.0	54	1922.49	479.12	3600.0	98.9
	RCBDA*	97	208.72	6.41	3600.0	28.8	73	1320.69	136.40	3600.0	122.0
COL	NWRCA*$_{v1}$	100	**2.14**	**0.50**	**21.9**	-	100	**67.92**	**7.70**	**720.7**	-
	NWRCA*$_{v2}$	100	6.18	1.22	61.3	2.4	100	201.75	19.63	1797.5	2.4
	ERCA*	97	204.77	26.46	3600.0	52.3	60	1779.38	468.34	3600.0	96.0
	RCBDA*	96	297.38	9.68	3600.0	379.0	72	1346.47	169.16	3600.0	112.4
		Instances with two resources $d = 2$					Instances with three resources $d = 3$				

Computation of heuristics. Our first analysis explores two approaches for computing heuristic functions for instances with three resources: (i) graph reformulation using Johnson's technique (as in ERCA*), and (ii) a modified version of Dijkstra's algorithm that allows for re-expansions. For the first approach, we used the improved Bellman-Ford algorithm [23, 8, 14] to reweight edge costs, followed by Dijkstra's to compute lower bounds. In case of RCBDA*, two rounds of lower-bound computation are required (one for the forward search and another for the backward search). Table 2 compares the average computation time for both approaches, assuming unidirectional heuristics are needed (as in ERCA* and NWRCA*). We observe that, the approach (i) requires a considerable amount of time to preprocess the graphs but enables faster computation of lower bounds, whereas the second approach, using modified Dijkstra's algorithm, delivers 3.5 to 4 times faster *overall* processing time without the need for reweighting in our (sparse) road networks. While approach (ii) can be advantageous in scenarios where the graph configuration changes between queries, it may be up to twice as slow compared to approach (i) when the search graph is static and already reweighted, though the difference is less than 0.5 seconds. Nonetheless, computing heuristic functions for NWRCA* using approach (ii) offers several benefits: i) it provides a simple yet efficient setup for constrained A* search with negative weights; ii) it eliminates the need for a time-intensive graph reformulation step, which is advantageous in dynamic scenarios; iii) it performs comparably to conventional methods when weights are non-negative (would be equivalent to standard Dijkstra).

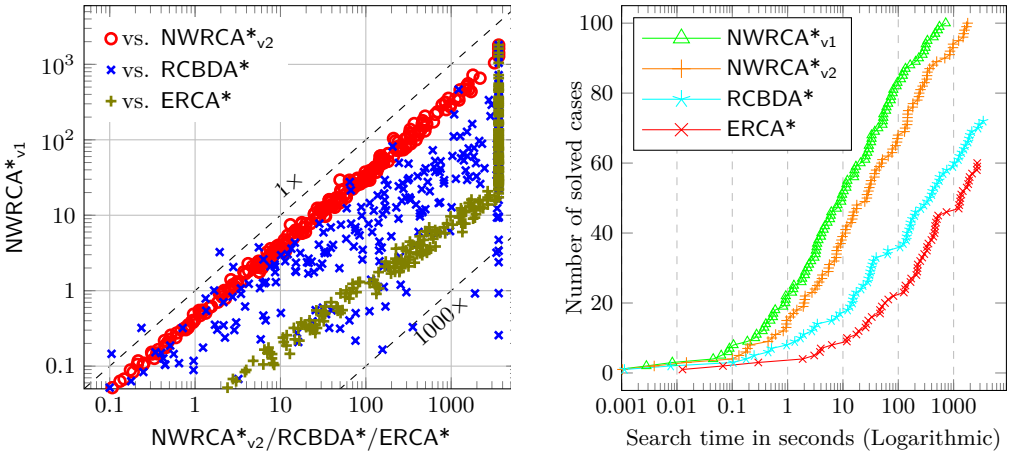


Figure 2 (Left) Runtime distribution of NWRCA*$_{v1}$ versus NWRCA*$_{v2}$ and the two other algorithms over all instances with $d = 3$. (Right) Algorithms' performance over the instances of the COL map with $d = 3$ (solved cases sorted based on runtime).

Algorithmic performance. Table 3 presents the experimental results for both scenarios ($d = 2, 3$). It displays the number of test cases successfully solved ($|\mathcal{P}|$), the arithmetic and geometric mean (Mean_A and Mean_G) and maximum constrained runtime observed (in seconds) by each algorithm. To enable a fair comparison of actual search performance across algorithms, and since all algorithms use same heuristic functions, we exclude preprocessing times (graph reformulation and lower bounding) from our runtime analyses. Therefore, the runtimes reported in this table refer solely to resource-constrained search time. All algorithms solved their easiest instances in less than 0.1 seconds. For unsolved cases, the search time is recorded as the timeout. It can be observed that both variants of NWRCA* have successfully solved almost all instances for each map in both scenarios, except for the BAY map where NWRCA*$_{v2}$ leaves four instances unsolved. The outperformance is even more pronounced in the BAY and COL maps with $d = 3$, where ERCA* and RCBDA* could solve less than 75% of cases. In terms of computation time, both variants of NWRCA* deliver significantly better performance than ERCA* and RCBDA* in all metrics. The detailed results reveal that NWRCA*$_{v1}$ consistently outperforms other algorithms, including NWRCA*$_{v2}$, across nearly all instances, successfully solving its most challenging case in under 31 minutes.

The parameter ϕ in the last column of Table 3 presents the (arithmetic) average slowdown factor for each algorithm relative to the top-performing algorithm, NWRCA*$_{v1}$. This factor is calculated by comparing the search time of each mutually solved instance with the corresponding search time of NWRCA*$_{v1}$, highlighting the relative slowdown. We also show in Figure 2 (Left) the runtime distribution of NWRCA*$_{v1}$ against other algorithms over instances of all maps with three resources ($d = 3$). Based on the results, NWRCA*$_{v2}$ performs nearly 2.5 times slower than NWRCA*$_{v1}$. This difference in performance is primarily due to NWRCA*$_{v2}$'s costly queuing process, which requires lexicographical ordering of labels in $\mathcal{Q}$ (tie-breaking with **resources**), and NWRCA*$_{v1}$'s more efficient label pruning mechanism. This improved pruning efficiency, in particular, stems from the partial traversal of the $\mathcal{X}$ lists, where non-dominated labels are maintained in lexicographical order of their **resources**.

Larger slowdown factors are observed for ERCA* and RCBDA*, with both algorithms being outperformed by up to two orders of magnitude on average. The more than one order of magnitude performance gap between ERCA* and NWRCA* is primarily due to NWRCA*'s more efficient lazy pruning strategy and its simpler yet effective data structure for dominance

checks, both contributing to significantly faster search times. In contrast, ERCA* suffers from a more rigid dominance pruning strategy that attempts to eliminate dominated and infeasible labels both at generation time and again upon extraction from the queue. While early dominance checks can help reduce queue size, they add computational overhead when applied to non-dominated labels. Furthermore, rechecking resource feasibility after extraction is redundant, as resource limits remain unchanged throughout the search, resulting in wasted computation and reduced efficiency.

Figure 2 (Right) also shows the number of solved cases for each algorithm, sorted by runtime, on the COL map with three resources. We observe that both variants of NWRCA* solve 60% of instances within 50 seconds, whereas ERCA* and RCBDA* require above 17 minutes to achieve the same success rate. In summary, both variants of NWRCA* demonstrate significantly faster performance than the state of the art in terms of computation time, with NWRCA*$_{v1}$ being approximately twice as fast as NWRCA*$_{v2}$.

ERCA* vs. RCBDA*. No performance comparison was provided between ERCA* and RCBDA* by the authors of ERCA*. Our results show that, while the improved RCBDA* algorithm solves more instances and achieves better average runtimes than ERCA* on certain maps, it does not consistently outperform ERCA*. The strong performance of RCBDA* in many instances is due to its bidirectional framework, which combines perimeter and alternating search strategies. In terms of worst-case runtime performance (Figure 2 (Left)), ERCA* generally performs better than RCBDA* but tends to be consistently around two orders of magnitude slower than NWRCA*$_{v1}$, whereas RCBDA* can be over three orders of magnitude slower than NWRCA*$_{v1}$ in some cases.

6 Conclusion

This paper introduced a novel, fast heuristic-guided label-setting approach with effective pruning mechanisms for efficiently solving resource-constrained shortest path (RCSP) problems in large-scale graphs. Unlike existing methods that aim to find a feasible solution, our framework identifies all non-dominated optimal solutions for a given RCSP instance. It also naturally handles instances with negative costs and resources, provided the cost heuristic is consistent and the search graph contains no negative cycles. We explored different configurations of the framework, distinguished by how heuristics are derived and how search labels are stored and explored during the search process. Experimental results highlight the importance of efficient ordering of labels in constrained search over large graphs and demonstrate the superiority of the proposed framework in solving challenging RCSP instances within tight time limits, outperforming state-of-the-art methods, including an enhanced version of the award-winning RCBDA* algorithm, by up to two orders of magnitude on average.

References

1 Saman Ahmadi, Nathan R. Sturtevant, Daniel Harabor, and Mahdi Jalili. Exact multi-objective path finding with negative weights. In Sara Bernardini and Christian Muise, editors, *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, pages 11–19. AAAI Press, 2024. doi:10.1609/ICAPS.V34I1.31455.

2 Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. Bi-objective search with bi-directional A*. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 3:1–3:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.3.

3 Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. A fast exact algorithm for the resource constrained shortest path problem. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Event, February 2-9, 2021*, pages 12217–12224. AAAI Press, 2021. doi:10.1609/AAAI.V35I14.17450.

4 Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. Vehicle dynamics in pickup-and-delivery problems using electric vehicles. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CP.2021.11.

5 Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. Weight constrained path finding with bidirectional A*. In Lukás Chrpa and Alessandro Saetti, editors, *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, pages 2–10. AAAI Press, 2022. doi:10.1609/SOCS.V15I1.21746.

6 Saman Ahmadi, Guido Tack, Daniel Harabor, Philip Kilby, and Mahdi Jalili. Enhanced methods for the weight constrained shortest path problem. *Networks*, 84(1):3–30, 2024. doi:10.1002/net.22210.

7 Saman Ahmadi, Guido Tack, Daniel Harabor, Philip Kilby, and Mahdi Jalili. Real-time energy-optimal path planning for electric vehicles. *arXiv preprint arXiv:2411.12964*, 2024. doi:10.48550/arXiv.2411.12964.

8 Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

9 Manuel A. Bolívar, Leonardo Lozano, and Andrés L. Medaglia. Acceleration strategies for the weight constrained shortest path problem with replenishment. *Optim. Lett.*, 8(8):2155–2172, 2014. doi:10.1007/s11590-014-0742-x.

10 Nicolás Cabrera, Andrés L. Medaglia, Leonardo Lozano, and Daniel Duque. An exact bidirectional pulse algorithm for the constrained shortest path. *Networks*, 76(2):128–146, 2020. doi:10.1002/net.21960.

11 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.

12 Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004. doi:10.1002/NET.20033.

13 Daniele Ferone, Paola Festa, Serena Fugaro, and Tommaso Pastore. On the shortest path problems with edge constraints. In *22nd International Conference on Transparent Optical Networks, ICTON 2020, Bari, Italy, July 19-23, 2020*, pages 1–4. IEEE, 2020. doi:10.1109/ICTON51198.2020.9203378.

14 Lester R Ford Jr. Network flow theory. Technical report, Rand Corp Santa Monica Ca, 1956.

15 Bruce L. Golden and Douglas R. Shier. 2019–2020 glover-klingman prize winners. *Networks*, 2021. doi:10.1002/net.22072.

16 Gabriel Y. Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980. doi:10.1002/net.3230100403.

17 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.

18 Carlos Hernández, William Yeoh, Jorge A Baier, Ariel Felner, Oren Salzman, Han Zhang, Shao-Hung Chan, and Sven Koenig. Multi-objective search via lazy and efficient dominance checks. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 7223–7230, 2023.

19 Gerhard Hiermann, Jakob Puchinger, Stefan Ropke, and Richard F. Hartl. The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *Eur. J. Oper. Res.*, 252(3):995–1018, 2016. doi:10.1016/J.EJOR.2016.01.038.

20 Donald B Johnson. A note on dijkstra's shortest path algorithm. *Journal of the ACM (JACM)*, 20(3):385–388, 1973. doi:10.1145/321765.321768.

21 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.

22 Leonardo Lozano and Andrés L. Medaglia. On an exact method for the constrained shortest path problem. *Comput. Oper. Res.*, 40(1):378–384, 2013. doi:10.1016/j.cor.2012.07.008.

23 Edward F. Moore. The shortest path through a maze. In *Proc. of the International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.

24 Keisuke Otaki, Tomosuke Maeda, Takayoshi Yoshimura, and Hiroyuki Sakai. Roaming navigation: Diverse constrained paths using heuristic search. *IEEE Access*, 11:75617–75627, 2023. doi:10.1109/ACCESS.2023.3295830.

25 Luigi Di Puglia Pugliese and Francesca Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013. doi:10.1002/net.21511.

26 Francisco Javier Pulido, Lawrence Mandow, and José-Luis Pérez-de-la-Cruz. Dimensionality reduction in multiobjective shortest path search. *Comput. Oper. Res.*, 64:60–70, 2015. doi:10.1016/j.cor.2015.05.007.

27 Zhongqiang Ren, Zachary B Rubinstein, Stephen F Smith, Sivakumar Rathinam, and Howie Choset. ERCA*: A new approach for the resource constrained shortest path problem. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

28 Zhongqiang Ren, Richard Zhan, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Enhanced multi-objective a* using balanced binary search trees. In Lukás Chrpa and Alessandro Saetti, editors, *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, pages 162–170. AAAI Press, 2022. doi:10.1609/SOCS.V15I1.21764.

29 Giovanni Righini and Matteo Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discret. Optim.*, 3(3):255–273, 2006. doi:10.1016/j.disopt.2006.05.007.

30 Antonio Sedeño-Noda and Sergio Alonso-Rodríguez. An enhanced K-SP algorithm with pruning strategies to solve the constrained shortest path problem. *Appl. Math. Comput.*, 265:602–618, 2015. doi:10.1016/j.amc.2015.05.109.

31 Barrett W. Thomas, Tobia Calogiuri, and Mike Hewitt. An exact bidirectional A* approach for solving resource-constrained shortest path problems. *Networks*, 73(2):187–205, 2019. doi:10.1002/net.21856.

32 Carlos Hernández Ulloa, William Yeoh, Jorge A. Baier, Han Zhang, Luis Suazo, and Sven Koenig. A simple and fast bi-objective search algorithm. In J. Christopher Beck, Olivier Buffet, Jörg Hoffmann, Erez Karpas, and Shirin Sohrabi, editors, *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 143–151. AAAI Press, 2020. URL: https://aaai.org/ojs/index.php/ICAPS/article/view/6655.

33 Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *Eur. J. Oper. Res.*, 209(1):1–10, 2011. doi:10.1016/J.EJOR.2010.03.045.

Fine-Grained Classification of Detecting Dominating Patterns

Jonathan Dransfeld
Karlsruhe Institute of Technology, Germany

Marvin Künnemann
Karlsruhe Institute of Technology, Germany

Mirza Redzic
Karlsruhe Institute of Technology, Germany

Abstract

We consider the following generalization of dominating sets: Let G be a host graph and P be a pattern graph P. A dominating P-pattern in G is a subset S of vertices in G that (1) forms a dominating set in G *and* (2) induces a subgraph isomorphic to P. The graph theory literature studies the properties of dominating P-patterns for various patterns P, including cliques, matchings, independent sets, cycles and paths. Previous work (Kunnemann, Redzic 2024) obtains algorithms and conditional lower bounds for detecting dominating P-patterns particularly for P being a k-clique, a k-independent set and a k-matching. Their results give conditionally tight lower bounds if k is sufficiently large (where the bound depends the matrix multiplication exponent ω). We ask: Can we obtain a classification of the fine-grained complexity for *all* patterns P?

Indeed, we define a graph parameter $\rho(P)$ such that if $\omega = 2$, then

$$\left(n^{\rho(P)} m^{\frac{|V(P)|-\rho(P)}{2}}\right)^{1\pm o(1)}$$

is the optimal running time assuming the Orthogonal Vectors Hypothesis, for all patterns P except the triangle K_3. Here, the host graph G has n vertices and $m = \Theta(n^\alpha)$ edges, where $1 \leq \alpha \leq 2$.

The parameter $\rho(P)$ is closely related (but sometimes different) to a parameter $\delta(P) = \max_{S \subseteq V(P)} |S| - |N(S)|$ studied in (Alon 1981) to tightly quantify the maximum number of occurrences of induced subgraphs isomorphic to P. Our results stand in contrast to the lack of a full fine-grained classification of detecting an arbitrary (not necessarily *dominating*) induced P-pattern.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases fine-grained complexity theory, domination in graphs, subgraph isomorphism, classification theorem, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.98

Funding Research supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 462679611.

1 Introduction

Among the most intensively investigated graph problems is the dominating set problem: Given a graph $G = (V, E)$, find a (small) vertex subset $S \subseteq V$ that *dominates* all vertices, i.e., for each $v \in V$, we have $v \in S$ or there is some $s \in S$ with $\{s, v\} \in E$. In many scenarios, one might not merely want to find *any* dominating set, but rather a dominating set S satisfying some additional requirements. Possibilities include S forming a connected subgraph, admitting a perfect matching, or more generally being connected by a prescribed topology (e.g., rings or cliques) or even being fully disconnected (i.e., forming an independent


© Jonathan Dransfeld, Marvin Künnemann, and Mirza Redzic;
licensed under Creative Commons License CC-BY 4.0



set). In this work, we study a general form of such problems: For any pattern graph H, the (Non-induced) Dominating P-Pattern problem asks to determine, given a graph G, whether there exists a (non-)induced copy of P in G that dominates all vertices of G.

Indeed, for various patterns P, the structural properties of Dominating P-Patterns have been well investigated, e.g., Dominating Cliques [5, 7, 12, 15, 26, 27, 28], Dominating Independent Sets[1] [8, 9, 10, 29, 34], Dominating (Induced) Matchings [22, 37, 38], Dominating Cycles [17, 18] or Dominating Paths [19, 20, 42]. Recently, the algorithmic complexity of detecting Dominating P-Patterns in general graphs has been performed by Künnemann and Redzic, focusing on cliques, independent sets and perfect matchings [30] (see below for further details). In this work, we set out to understand the fine-grained complexity of this problem for *all* patterns P.

Note that the Dominating P-Pattern problem is the natural combination of two classic problems which are heavily studied in isolation: Dominating Set and P-Pattern Detection.

- *k-Dominating Set*: Dominating set is a notoriously difficult problem. When parameterized by the solution size k (i.e., $|S|$), it is the arguably most natural W[2]-complete problem and hard even to approximate within $f(k)$ factors [6, 24]. The best known algorithm is based on fast matrix multiplication and solves it in time $n^{k+o(1)}$ for all sufficiently large k [16]. This is tight in the sense that an $\mathcal{O}(n^{k-\epsilon})$-time algorithm for any $k \geq 3$ and $\epsilon > 0$ would refute the k-Orthogonal Vectors Hypothesis (k-OVH) and thus the Strong Exponential Time Hypothesis [36] (see Section 2 for details). Taking the graph sparsity (i.e., the number m of edges) into account as well, a recent result [21] gives upper and conditional lower bounds establishing a tight time complexity of $mn^{k-2\pm o(1)}$ if the matrix multiplication exponent ω is equal to 2.
- *P-Pattern Detection (aka Induced P-Subgraph Isomorphism):* The complexity of P-Pattern Detection for general P is sensitive to the considered pattern P. The probably most notable special case is k-Clique Detection. It is the most natural $W[1]$-complete problem and also resists good approximations [6, 31, 23]. Its best known algorithm solves it in time essentially $\mathcal{O}(n^{\frac{\omega}{3}k})$ [33][2] and is conjectured to be optimal, see, e.g., [1]. For other (classes of) patterns P, however, only partial results are known: While for any k-node pattern P, the problem can be reduced to k-Clique Detection, this approach is not necessarily optimal. E.g., all 4-node patterns except clique and independent set can be detected polynomially faster than the conjectured time for clique and independent set, see in particular [25, 16, 39, 13]. Despite significant effort (see [25, 16, 39, 14, 13] for a selection), the task of finding matching upper and conditional lower bounds appears far from completed. Let us remark that also for Non-induced P-Pattern Detection a fine-grained classification of all patterns P remains open (see, e.g., [32, 13]).

We ask: *How does the time complexity of Dominating P-Pattern relate to the complexity of P-Pattern Detection versus to the complexity of k-Dominating Set?* Can we, despite the lack of a complete classification for P-Pattern Detection classify the fine-grained complexity of **all** *dominating* patterns?

Previous work appears to indicate that the complexity should be governed by the domination aspect: By adapting the conditional lower bound of Patrascu and Williams for k-Dominating Set [36] (see also [21, 30]) it is not too difficult to obtain a conditional lower

[1] We remark that a Dominating Independent Set is equivalent to the well-studied notion of a Maximal Independent Set.
[2] If k is divisible by 3.

bound of $n^{k-o(1)}$ in dense graphs for *any* pattern P, based on k-OVH. However, the situation becomes much more interesting for *sparse* graphs. Here, it has already been observed that the fine-grained time complexity is highly sensitive to the specific pattern P. Specifically, Künnemann and Redzic [30] give some curious insights into selected patterns: (Here, to simplify the presentation, we assume that $\omega = 2$ and that k-OVH holds.)

- If P is the k-star (a tree with a root and $k-1$ leaves), the tight time complexity is $mn^{k-2\pm o(1)}$, i.e., even in very sparse graphs with $m = \Theta(n)$, we only save a linear factor in n compared to the $n^{k\pm o(1)}$ complexity in dense graphs.
- In contrast, if P is the clique with $k \geq 5$ vertices or the independent set with $k \geq 3$ vertices, we obtain a particularly simple case with tight time complexity of $(m^{\frac{k+1}{2}}/n)^{1\pm o(1)}$. In very sparse graphs with $m = \mathcal{O}(n)$, the resulting running time of $n^{\frac{k-1}{2}\pm o(1)}$ is less than the square root of the running time of $n^{k\pm o(1)}$ in dense graphs.
- Finally, the substantially different pattern of a perfect matching on $k \geq 4$ vertices achieves the only slightly worse time complexity of $m^{\frac{k}{2}\pm o(1)}$.

We remark that the above results suggest that in sparse graphs, Dominating P-Pattern shares an additional flavor with the P-Pattern *Enumeration* problem: It is not too difficult to obtain an algorithm whose running time is roughly bounded by the time required to list all occurrences of the pattern P. However, it turns out that this number alone cannot fully explain the time complexity: While for the case of perfect matchings, the time complexity coincides precisely with the maximum number of occurrences of the pattern, for others (e.g., independent sets or cliques), the time complexity is polynomially less than the maximum number of occurrences of the pattern. This begs the question: which other parameter of the pattern P captures the time complexity of Dominating P-Pattern?

Our results. To state the time complexity for any pattern P, we introduce the following graph parameter $\rho(P)$. Here for any graph P and $S \subseteq V(P)$, we let $N(S)$ denote the (open) neighborhood of the vertices S, where $N(\emptyset) = \emptyset$.[3]

▶ **Definition 1.1.** *Let P be an arbitrary pattern graph, and denote by $I(P) \subseteq V(P)$ the set of isolated nodes of P. Choose an* independent *set $S \subseteq V(P) \setminus I(P)$ maximizing $|S| - |N(S)|$; if S is not uniquely defined, take any choice maximizing $|S|$. We define*

$$\rho(P) := \begin{cases} |S| - |N(S)| & \text{if } S \neq \emptyset \\ -1 & \text{if } S = \emptyset. \end{cases}$$

and set

$$t_P(n,m) := n^{\rho(P)} \cdot m^{\frac{|V(P)|-\rho(P)}{2}}.$$

These quantities turn out similar (but sometimes different) to the maximum number of occurrences of the pattern graph P for patterns *without isolated nodes* (i.e., $I(P) = \emptyset$). Specifically, if $I(P) = \emptyset$ and $S \neq \emptyset$, the maximum number of induced copies of P in an n-vertex m-edge graph G is $\Theta(t_P(n,m))$. Indeed, for connected graphs G and patterns P, Alon [3] defines the parameter $\delta(P) := \max_{S \subseteq V(P)} |S| - |N(S)|$ and establishes $\Theta(n^{\delta(P)} m^{\frac{|V(P)|-\delta(P)}{2}})$ as the maximum number of induced copies of P in G whenever G has $m = \Theta(n)$ edges

[3] For any graph G, we will always denote by $V(G)$ and $E(G)$ its set of vertices and edges, respectively.

(which generalizes to arbitrary $m \geq n$).[4] Note that $\rho(P)$ and $\delta(P)$ may only differ for some patterns P with $\delta(P) = 0$, in which case $\rho(P) \in \{0, -1\}$. In contrast, if P contains isolated nodes (e.g., the case of Dominating Independent Set), the number of occurrences of P and $t_P(n, m)$ may differ vastly (e.g., between $\Theta(n^k)$ and $\Theta(m^{k/2})$).

Our results determine that for all patterns (possibly except the triangle K_3), $t_P(n, m)^{1\pm o(1)}$ is the conditionally tight time complexity of detecting a Dominating P-Pattern if $\omega = 2$. Put differently, for any pattern P (except K_3), we can easily determine the conditionally optimal running time (assuming $\omega = 2$)! Specifically, we obtain the following algorithmic result:

▶ **Theorem 1.2** (Upper Bound). *For any pattern graph P with at least 16 vertices, there is an algorithm solving Dominating P-Pattern problem in time $t_P(n, m)^{1+o(1)}$. Further, if $\omega = 2$, this algorithm exists for all patterns P except K_3.*

We remark that our algorithms have a running time close to $t_P(n, m)^{1\pm o(1)}$ even under current values of ω and small pattern sizes – the small polynomial overhead depends on ω and vanishes if $\omega = 2$ (except for the triangle K_3). Further, we complement our algorithmic result by a corresponding conditional lower bound of $t_P(n, m)^{1-o(1)}$ based on the k-Orthogonal Vectors Hypothesis (and thus the Strong Exponential Time Hypothesis).

▶ **Theorem 1.3** (Conditional Lower Bound). *For any pattern graph P with at least 2 vertices, there is no algorithm solving the P-Dominating Set problem in time $\mathcal{O}(t_P(n, m)^{1-\varepsilon})$ for any $\varepsilon > 0$ unless the k-Orthogonal Vectors Hypothesis fails.*

The only pattern P for which we do not obtain an algorithm matching the lower bound of Theorem 1.3 is the triangle K_3. For this pattern, our best algorithm only leaves a time overhead of $m^{1/3+o(1)}$ if $\omega = 2$, yielding a bound of $(t_{K_3}(n, m) \cdot m^{\frac{1}{3}})^{1+o(1)} = (m^{\frac{7}{3}}/n)^{1+o(1)}$ in this case.

▶ **Theorem 1.4.** *We can solve Dominating Triangle in time $\left(m^{1+\frac{2\omega}{\omega+1}}/n\right)^{1+o(1)}$.*

Technical Outline and Overview. In Section 3, we obtain our algorithmic results: We first study sufficiently large connected patterns P (different from K_3 and K_4), which we call *basic patterns*. We argue that there are only $\mathcal{O}(t_P(n, m))$ sets that might form a Dominating P-Pattern (using similar arguments to Alon [3]). Moreover, we show how to enumerate all such candidate sets in time $t_P(n, m)^{1+o(1)}$ in such a way that allows us to reduce to fast matrix multiplication to perform a dominance check as in Eisenbrand and Grandoni [16]. Notably, this approach introduces a polynomial overhead for K_3 and K_4, as the created matrices are too large compared to the lower bound. We handle these cases separately: Using a careful combination of ideas from sparse Triangle Counting [4] and sparse 2-Dominating Set [21], we are able to improve the algorithms for K_4 and K_3; if $\omega = 2$, this completely eliminates the overhead for K_4 and reduces the overhead for K_3 to $m^{1/3+o(1)}$. Finally, we extend our algorithm beyond connected patterns by showing how to handle isolated nodes:

[4] To clarify the correspondence, we remark that already Alon (see [3, proof of Lemma 8]) observed that in the definition of $\delta(P)$, one may without loss of generality let S range only over independent sets: for any set S maximizing $|S| - |N(S)|$, the set $S' = S \setminus (S \cap N(S))$ is an independent set with $|S'| - |N(S')| \geq |S| - |N(S)|$. Thus, whenever a nonempty set S maximizes $|S| - |N(S)|$, we have $\rho(P) = \delta(P)$. Alon proves that a connected m-edge graph G has a maximum number of $\Theta(m^{\frac{|V(P)|+\delta(P)}{2}})$ induced copies of P, where P is any connected pattern. The generalization to an arbitrary trade-off between n and m is implicit in our proofs.

We can no longer enumerate all candidate sets, as this number becomes too large. Instead, we apply a recursive approach similar to the algorithm for Dominating Independent Sets given in [30]. Here, particularly the case of a single isolated node requires great care.

Due to space reasons, our conditional lower bounds had to be deferred to the full version of the paper. Our conditional lower bound construction generalizes the ones in [30] (which in turn are based on [36, 21]): Specifically, we reduce from a version of the k-OV problem with carefully chosen set sizes. Intuitively, the parameter $\rho(P)$ spells out how to choose these sizes: (1) for every vertex $v \in S$, we have a set of n vectors, (2) for every vertex $v \in N(S)$, we have a set of m/n vectors, and (3) for every vertex $v \in V(P) \setminus (S \cup N(S))$, we have a set of $\sqrt{m}$ vectors. The precise construction requires care; in particular if $S = \emptyset$, we need to use an alternative choice of a single set of m/n vectors and $k-1$ sets of $\sqrt{m}$ vectors.

We additionally study variants of Dominating P-Pattern where we are given a set $\mathcal{Q}$ of patterns rather than a single pattern. The corresponding problem Dominating $\mathcal{Q}$-Pattern asks to detect a dominating set S that induces a subgraph isomorphic to *some* $P \in \mathcal{Q}$. A notable special case is the Non-Induced Dominating P-Pattern problem in which the task is to detect a dominating set S such that $G[S]$ contains P as a subgraph. Combining the following theorem with Theorems 1.2 and 1.3, our results settle the fine-grained complexity of Non-Induced Dominating P-Pattern for any pattern P except K_3 (assuming $\omega = 2$).

▶ **Theorem 1.5** (Dominating $\mathcal{Q}$-Pattern, Informal version). *Let Q be a finite set of patterns of identical order. The fine-grained complexity of Dominating $\mathcal{Q}$-Pattern is dominated by the pattern $P \in \mathcal{Q}$ with the highest time complexity.*

Due to the space restrictions, the proof of this theorem is provided only in the full version of the paper.

2 Preliminaries

For a $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. For a set X, its power set is denoted by 2^X and the set of all subsets of size k by $\binom{X}{k}$. For two sets X, Y, by $X \times Y$ we denote a set of unordered pairs, i.e. $X \times Y := \{\{x, y\} \mid x \in X, y \in Y\}$. For a pattern graph P and a host graph G, we use k and n, respectively, to denote their order, i.e., number of vertices. As a shorthand for $\{u, v\}$, the notation uv denotes an edge between u and v. The set $N(v) = \{u \mid uv \in E\}$ is the *neighborhood* of v. The *closed neighborhood* of v is $N[v] = N(v) \cup \{v\}$. For a set of vertices $S \subseteq V$, by $N(S)$ (resp. $N[S]$), we denote the set $\bigcup_{v \in S} N(v)$ (resp. $\bigcup_{v \in S} N[v]$). Moreover, the subgraph of G *induced* by S is denoted by $G[S]$. For a vertex $v \in V$, the graph $G - v$ denotes the graph G with the vertex v deleted. Likewise, for a set $X \subseteq V$, we use $G - X := G[V \setminus X]$. A set of vertices $D \subseteq V$ dominates the graph G if every vertex of v is either in D or has a neighbor in D.

The matrix multiplication exponent $\omega \geq 2$ is the smallest constant such that there is an algorithm multiplying any two $n \times n$ matrices in $n^{\omega + o(1)}$ time. For two rectangular matrices of size $n_1 = n \times n_2$ and $n_2 \times n_3$, $\mathrm{MM}(n_1, n_2, n_3)$ denotes the time complexity of multiplying these matrices. Similarly, for matrix of size $n^a \times n^b$ and $n^b \times n^c$, $n^{\omega(a,b,c)+o(1)}$ denotes the time complexity of multiplying them. Further, $\alpha \leq 1$ is the largest constant such that $\omega(1, \alpha, 1) = 2$. The best known bounds for α and ω are $\omega \leq 2.3713$ [2], and $\alpha \geq 0.3213$ [41].

3 Algorithm for Pattern Dominating Set

In this section we develop the algorithms for Dominating P-Pattern problem for different choices of patterns P and prove the following main theorem as stated in the Introduction.

▶ **Theorem 1.2** (Upper Bound). *For any pattern graph P with at least 16 vertices, there is an algorithm solving Dominating P-Pattern problem in time $t_P(n,m)^{1+o(1)}$. Further, if $\omega = 2$, this algorithm exists for all patterns P except K_3.*

In order to prove this theorem, we proceed in three steps. We first develop algorithms for a class of patterns P we call *basic*: all patterns that contain no isolated vertices and are not isomorphic to a K_3 or K_4. We then handle patterns with isolated vertices and the remaining small cliques K_3, K_4 separately. An important ingredient that we use to speed up the dominance check is an approach via fast matrix multiplication due to Eisenbrand and Grandoni [16].

▶ **Lemma 3.1.** *Let X, Y be the sets of vertices and $\phi : 2^X \to \{0,1\}$ be a predicate such that for any $D \subseteq X$ we can check $\phi(D)$ in constant time. Let $\mathcal{V}_A, \mathcal{V}_B \subseteq 2^X$ be sets of subsets of X such that any subset $D \subseteq X$ that satisfies ϕ can be written as a union of two sets $A \in \mathcal{V}_A$ and $B \in \mathcal{V}_B$. Then in time $\mathrm{MM}(|\mathcal{V}_A|, |Y|, |\mathcal{V}_B|)$ we can enumerate all subsets of X that satisfy ϕ and dominate Y.*

We remark that in the k-Dominating Set algorithm [16], the predicate $\phi(D)$ from the previous lemma is true if $|D| = k$ and false otherwise. In our case, for a fixed pattern P, the predicate $\phi(D)$ will be true if the subgraph of G induced by D is isomorphic to P (i.e. $G[D] \cong P$).[5] Recall also our definition of the parameter $\rho(P)$, which will be relevant in this section.

▶ **Definition 1.1.** *Let P be an arbitrary pattern graph, and denote by $I(P) \subseteq V(P)$ the set of isolated nodes of P. Choose an* independent *set $S \subseteq V(P) \setminus I(P)$ maximizing $|S| - |N(S)|$; if S is not uniquely defined, take any choice maximizing $|S|$. We define*

$$\rho(P) \coloneqq \begin{cases} |S| - |N(S)| & \text{if } S \neq \emptyset \\ -1 & \text{if } S = \emptyset. \end{cases}$$

and set

$$t_P(n,m) \coloneqq n^{\rho(P)} \cdot m^{\frac{|V(P)|-\rho(P)}{2}}.$$

Let S be a (possibly empty) independent set of non-isolated vertices in P that maximizes the value $|S| - |N(S)|$ as in Definition 1.1. We define the *remainder* set R to be $V(P) \setminus N[S]$. The following observation follows directly from the fact that S is an independent set.

▶ **Observation 3.2.** *Let P be a pattern on k vertices and $S, N(S), R$ be as defined above. Then $S, N(S), R$ form a partition of $V(P)$.*

Before proceeding with the algorithm, we state another observation, that gives us a nice way to think about the value of $t_P(n,m)$ depending on whether set S is empty or not.

▶ **Observation 3.3.** *Let P be a pattern on k vertices and $S, N(S), R$ be the partition of $V(P)$ as defined above.*

- *If $S = \emptyset$, then $t_P(n,m) = \frac{m^{(k+1)/2+o(1)}}{n}$.*
- *If $S \neq \emptyset$, then $t_P(n,m) = t_{P[N[S]]}(n,m) \cdot m^{|R|/2}$.*

If we consider a partition into sets $S, N(S)$ and R similar as above, intuitively we have some structure on the parts S (independent set in P) and $N(S)$ (there exists a maximal matching that matches each vertex in $N(S)$ to a vertex in S), but we have very little structure on how the remainder set looks like. In the following lemma we show that we can decompose the remainder set R into much simpler subgraphs (disjoint edges and cycles).

[5] Note that verifying ϕ takes $f(|P|) = \mathcal{O}(1)$ time, for constant size patterns.

▶ **Lemma 3.4.** *Let S, $N(S)$ and R be a partition of $V(P)$ as defined above and assume that the induced subgraph $P[R]$ contains no isolated vertices. Then the following holds.*

- *There exists a spanning subgraph of $P[R]$ that is isomorphic to a disjoint union of edges and odd cycles.*
- *For any vertex $x \in R$, there exists a spanning subgraph of $P[R-x]$ that is isomorphic to a disjoint union of edges and odd cycles.*

The proof follows from a structural theorem proved in [11] together with the choice of the set S in Definition 1.1. For details we refer the reader to the full version of the paper. We are now equipped with all the tools we need to construct the algorithm for the first family of patterns P that we call *basic patterns*.

3.1 Basic Patterns

In this section we construct an algorithm that solves the P-Dominating Set problem in the (conditionally) optimal time for most of the patterns. More precisely, we say a pattern graph P is *basic* if: 1) it has at least three vertices; 2) it has no isolated vertices; 3) it is neither isomorphic to a K_3 nor a K_4. We prove the following theorem for basic patterns.

▶ **Theorem 3.5.** *For any basic pattern P on at least 16 vertices, there exists an algorithm solving the Dominating P-Pattern problem in time $t_P(n,m)^{1+o(1)}$. Moreover, if $\omega = 2$, this time complexity can be achieved for all basic patterns.*

The high level idea is to first decompose the pattern into (not necessarily induced) odd cycles, edges and isolated vertices, and then enumerate all possible *valid* choices for each of those parts efficiently, and then use Lemma 3.1 to check if the union of guessed parts induces a subgraph isomorphic to P, and if it dominates G. We first show that we can enumerate all subgraphs isomorphic to the remainder set R efficiently. Let $k := |V(P)|$ and $n := |V(G)|$. We say a vertex $v \in V(G)$ is *heavy* if $\deg(v) \geq \frac{n}{k} - 1$. We prove the following lemma in the full version of the paper.

▶ **Lemma 3.6.** *Let P be a basic pattern. Let $S, N(S), R$ be a partition of $V(P)$ as defined in Definition 1.1. Then the following holds.*

1. *We can enumerate all subgraphs of G that are isomorphic to $P[R]$ in time $\mathcal{O}(m^{|R|/2})$.*
2. *We can enumerate all subgraphs of G that are isomorphic to $P[R]$ and contain a heavy vertex in time $\mathcal{O}(\frac{m^{(|R|+1)/2}}{n})$.*

We now use the lemma above to handle the case when the decomposition into S, $N(S)$ and R as in Definition 1.1 yields $S = \emptyset$. Due to space constraints, we only sketch the proof here. The details can be found in the full version.

▶ **Lemma 3.7.** *Let P be a basic pattern and $S, N(S), R$ be a partition of $V(P)$ as defined in Definition 1.1, such that $S = \emptyset$. If the basic pattern P has at least 16 vertices, then there exists an algorithm solving Dominating P-Pattern in time $\left(\frac{m^{\frac{k-1}{2}}}{n}\right)^{1+o(1)} = t_P(n,m)^{1+o(1)}$. If $\omega = 2$, this time complexity can be achieved for all basic patterns P.*

Proof (sketch). If $S = \emptyset$, then $V(P) = R$ and Lemma 3.6 applies to all of P. Recall that any dominating set of size k contains a heavy vertex v_h (a vertex of degree at least $n/k - 1$), hence we can first guess this vertex in time $\mathcal{O}(\frac{m}{n})$. By Lemma 3.4, there is a spanning subgraph of P that is isomorphic to a disjoint union of edges and odd cycles and we can find such a spanning subgraph in $f(k) = \mathcal{O}(1)$ time. Assume that this decomposition yields α odd cycles and β disjoint edges. If $\beta > 0$, we show that we can decompose $P - v_h$ further into two graphs B_1, B_2 such that they satisfy the following conditions:

- Each B_i is a disjoint union of independent P_3's (paths of length two) and K_2's (independent edges), such that each copy of P_3 will correspond to a subgraph of G that is isomorphic to P_3 and whose all vertices either have degree at most $\sqrt{m}$, or one of the endpoints has degree at least $\sqrt{m}$. [6]
- $B_1 \cup B_2$ is a spanning subgraph of $P - v_h$.
- (Without loss of generality) $0 \leq |V(B_1)| - |V(B_2)| \leq 2$.
- We can enumerate all copies of G isomorphic to B_i that satisfy the degree constraints of P_3 in time $\mathcal{O}(m^{|B_i|/2})$.

In particular, for $k \geq 16$, we have $|V(B_1)| + |V(B_2)| = k - 1 \geq 15$, and combining this with the inequality above, we can conclude that $|V(B_1)| \geq |V(B_2)| \geq 7$. Using that the rectangular matrix multiplication constant $\alpha \geq 0.321$, and that $m \geq n$, we for the exponent

$$\omega(1, \frac{2}{|V(B_1')|}, 1) = \omega(1, \frac{2}{|V(B_2')|}, 1) = 2, \tag{1}$$

and hence we obtain for all $k \geq 16$:

$$\begin{aligned}
\mathcal{O}\left(\frac{m}{n}\,\mathrm{MM}\left(m^{\frac{|V(B_1')|}{2}}, n, m^{\frac{|V(B_2')|}{2}}\right)\right) &\leq \mathcal{O}\left(\frac{m}{n}\,\mathrm{MM}\left(m^{\frac{|V(B_1')|}{2}}, m, m^{\frac{|V(B_2')|}{2}}\right)\right) && (m \geq n)\\
&\leq \mathcal{O}\left(\frac{m}{n} \cdot m^{\frac{|V(B_1')|+|V(B_2')|}{2}+o(1)}\right) && (\text{Eq. 1})\\
&= \mathcal{O}\left(\frac{m}{n} \cdot m^{\frac{k-1}{2}+o(1)}\right) = \left(\frac{m^{\frac{k+1}{2}}}{n}\right)^{1+o(1)}.
\end{aligned}$$

If on the other hand $\beta = 0$ (i.e. the decomposition contains only copies of odd cycles), we proceed similarly, but the best bound on the size difference between B_1 and B_2 is $0 \leq |V(B_1)| - |V(B_2)| \leq 3$, which is not good enough in general. To get around this issue, we distinguish between two cases. If there exists a vertex in $P - v_h$ that is mapped to a vertex of degree $\geq \sqrt{m}$ in G, we can make sure that there exists a copy of P_3 whose one endpoint will be mapped to a vertex in G that has degree at least $\sqrt{m}$. We can decompose this P_3 further into an edge and a vertex of degree $\geq \sqrt{m}$, and intuitively use these components to "re-balance" the sets $V(B_1)$ and $V(B_2)$ as above. If on the other hand all vertices in $P - v_h$ are mapped to vertices in G that have degree at most $\sqrt{m}$, we then know that if $|Y| := |G - N_G[v_h]| \geq (k-1)\sqrt{m} + 1$, there is no valid solution. If on the other hand $|Y| \leq (k-1)\sqrt{m}$, by Lemma 3.1, we can find any such solution in time bounded by $\mathcal{O}\left(\frac{m}{n}\,\mathrm{MM}\left(m^{\frac{|V(B_1)|}{2}}, \sqrt{m}, m^{\frac{|V(B_2)|}{2}}\right)\right)$. Similarly as in previous case, for $k \geq 16$, we get that $|V(B_1)| \geq |V(B_1)| \geq 6$ and again since $\alpha \geq 0.321$, we obtain $\omega(1, \frac{1}{|V(B_1)|}, 1) = \omega(1, \frac{1}{|V(B_2)|}, 1) = 2$. Hence, we can bound the time complexity as

$$\begin{aligned}
\mathcal{O}\left(\frac{m}{n}\,\mathrm{MM}\left(m^{\frac{|V(B_1)|}{2}}, \sqrt{m}, m^{\frac{|V(B_2)|}{2}}\right)\right) &\leq \mathcal{O}\left(\frac{m}{n} \cdot \left(\sqrt{m}\right)^{|V(B_1)|+|V(B_2)|+o(1)}\right)\\
&= \mathcal{O}\left(\frac{m}{n} \cdot \left(\sqrt{m}\right)^{k-1+o(1)}\right) = \left(\frac{m^{\frac{k+1}{2}}}{n}\right)^{1+o(1)} \quad \blacktriangleleft
\end{aligned}$$

The second case for basic patterns is that the decomposition into S, $N(S)$ and R as in Definition 1.1 yields $S \neq \emptyset$. Intuitively this case is simpler than the one above, since we can use the sets $S, N(S)$ to simulate $\beta > 0$, even if $\beta = 0$ and "balance" the sets $V(B_1)$ and $V(B_2)$. We only sketch the proof, as the details are quite similar to the ones above.

[6] Hence in G we will be able to enumerate all subgraphs isomorphic to P_3 that fulfill this degree requirement in time $\mathcal{O}(m^{3/2})$.

▶ **Lemma 3.8.** *Let P be a basic pattern and $S, N(S), R$ be a partition of $V(P)$ as defined in Definition 1.1, such that $S \neq \emptyset$. If the basic pattern P has at least 16 vertices, then there exists an algorithm solving Dominating P-Pattern in time $\left(n^{|S|-|N(S)|}m^{|N(S)|+\frac{|R|}{2}}\right)^{1+o(1)} = t_P(n,m)^{1+o(1)}$. If $\omega = 2$, this time complexity can be achieved for all basic patterns P.*

Proof (sketch). Similarly as above, construct the sets B_1 and B_2 by finding a spanning subgraph of $P[R]$ that consists of union of independent P_3's and K_2's and distributing the connected components as equally as possible. Clearly, we have that $|V(B_1)| + |V(B_2)| = |R|$, and (without loss of generality) $0 \leq |V(B_1)| - |V(B_2)| \leq 3$. Since $S \neq \emptyset$, also $N(S) \neq \emptyset$ (by definition S contains no isolated vertices), hence we can distribute the edges from a perfect matching between the set $N(S)$ and some subset of S of size $|N(S)|$ (such perfect matching always exists by Hall's marriage theorem) between the sets B_1 and B_2 similarly as in previous lemma to obtain graphs B_1', B_2'. Clearly, it holds that $|V(B_1')| + |V(B_2')| = 2|N(S)| + |R|$ and (without loss of generality) $0 \leq |V(B_1')| - |V(B_2')| \leq 2$. We then define the graphs Q_1, Q_2 that consist of $\lfloor \frac{|S|-|N(S)|}{2} \rfloor$ and $\lceil \frac{|S|-|N(S)|}{2} \rceil$ isolated vertices respectively. Similarly as above, we can enumerate all subgraphs of G isomorphic to $B_i' \cup Q_i$ in time $\mathcal{O}\left(n^{|Q_i|}m^{\frac{|B_i'|}{2}}\right)$. Also, by similar arguments as above, we have that for $k \geq 16$, the matrices are "thin enough" that rectangular matrix multiplication techniques can be used to obtain the tight running time of

$$\begin{aligned} \mathrm{MM}\left(n^{|Q_1|}m^{\frac{|B_1'|}{2}}, n, n^{|Q_2|}m^{\frac{|B_2'|}{2}}\right) &\leq \left(n^{|Q_1|+|Q_2|}m^{(|B_1'|+|B_2'|)/2}\right)^{1+o(1)} \qquad (k \geq 16) \\ &\leq \left(n^{|S|-|N(S)|}m^{\frac{2|N(S)|+|R|}{2}}\right)^{1+o(1)} \end{aligned}$$

◀

3.2 Small Cliques

In this section, we deal with the remaining small patterns, i.e., the cliques of size at most 4. We note that in [30] the K_2 was already settled, so it remains to settle the cliques of size 3 and 4. In particular, this section is dedicated to proving the following two main theorems.

▶ **Theorem 3.9.** *For a Dominating Triangle, there exists a deterministic $\left(\frac{m^{1+\frac{2\omega}{\omega+1}}}{n}\right)^{1+o(1)}$-time algorithm and a randomized $m^{\frac{\omega+1}{2}+o(1)}$-time algorithm.*

▶ **Theorem 3.10.** *There is a randomized algorithm solving K_4-Dominating Set in time $\left(\frac{m^{(\omega+3)/2}}{n}\right)^{1+o(1)}$. If $\omega = 2$, this is $\left(\frac{m^{5/2}}{n}\right)^{1+o(1)} = t_{K_4}(n,m)^{1+o(1)}$.*

We start with the simple deterministic algorithm for Dominating Triangle problem that uses a reduction to the *All-Edges-Triangle-Counting* problem. This problem asks for a given tripartite graph $G = (V_1 \cup V_2 \cup V_3, E)$ to determine, for every edge e in $(V_1 \times V_2) \cap E$, how many triangles in G contain e. It is well known that this problem can be solved in time $m^{\frac{2\omega}{\omega+1}+o(1)}$ (see e.g. [4]). We remark that any $\mathcal{O}(m^{\frac{4}{3}-\varepsilon})$-time algorithm would refute the 3-SUM and the APSP Hypothesis [35, 40], giving matching lower bounds if $\omega = 2$.

▶ **Lemma 3.11** (All-Edge Triangle Counting, see [4])**.** *There is an algorithm solving All-Edges-Triangle-Counting in time $m^{\frac{2\omega}{\omega+1}+o(1)}$.*

By using this lemma, we are able to solve the problem in $\mathcal{O}\left(t_{K_3}(n,m)\right) \cdot m^{\frac{\omega-1}{\omega+1}+o(1)}$, which evaluates to $\mathcal{O}\left(t_{K_3}(n,m)\right) \cdot m^{\frac{1}{3}+o(1)}$ if $\omega = 2$. By a more careful approach, we are able to construct a randomized algorithm that uses a bloom-filter inspired approach similar to the

one in [21], to achieve $\mathcal{O}\left(t_{K_3}(n,m)\right) \cdot \frac{n}{\sqrt{m}}$, which is better whenever $m \in \Omega(n^{1.10283})$ with current value of ω and $m \in \Omega(n^{1.2})$ if $\omega = 2$, giving us the second part of the Theorem 3.9. We remark that although this value can be slightly improved by using a more careful analysis, it seems that in order to match the lower bound, a new technique will be required. We now sketch the approach of the deterministic algorithm and give the details to both algorithms in the full version of the paper.

▶ **Proposition 3.12.** *There exists a deterministic algorithm solving Dominating Triangle in time* $\left(\frac{m^{1+\frac{2\omega}{\omega+1}}}{n}\right)^{1+o(1)}$. *If* $\omega = 2$, *this equals* $\left(\frac{m^{7/3}}{n}\right)^{1+o(1)}$.

Proof (sketch). We first guess a heavy vertex w (a vertex satisfying $\deg(w) \geq n/3 - 1$). We then run a breadth-first-search from w and let $(X_0 = \{w\}, X_1, \ldots, X_n)$ be the obtained BFS layering. Note that for any $\ell \geq 3$, if X_ℓ is non-empty, we can conclude that there is no dominating clique that contains w and we can proceed with the next choice of the heavy vertex. Otherwise, finding any dominating triangle in G that contains w is equivalent to finding an edge u, v in X_1 that dominates all vertices in X_2. For any vertex u, we write $N_2(u) := N(u) \cap X_2$. Using the principle of inclusion-exclusion, we can write $|N_2(u) \cup N_2(v)| = |N_2(u)| + |N_2(v)| + |N_2(u) \cap N_2(v)|$. We note that for any pair of vertices $u, v \in X_1$ the value $|N_2(u) \cap N_2(v)|$ corresponds to the number of triangles containing the edge uv. Hence, by employing the algorithm from Lemma 3.11, we obtain the desired time. ◀

On the other hand, by coming up with a more involved argument, we are able to construct a randomized algorithm that matches the lower bound (up to resolving the matrix multiplication exponent ω) for the remaining small pattern, namely Dominating 4-Clique. The algorithm starts similarly as above. That is, we first guess a heavy vertex w in time $\mathcal{O}\left(\frac{m}{n}\right)$, and then we partition the vertices into two sets X_1 and X_2 based on the distance from w and thus reduce to detecting a triangle in X_1 that dominates X_2. We then consider two cases based on the size of X_2. If $|X_2| \leq 3\sqrt{m}$, this turns out to be quite easy, and boils down to using standard matrix multiplication algorithms (with a bit more careful analysis). If on the other hand $|X_2| > 3\sqrt{m}$, we then either reduce to counting cliques (triangles and K_4's) in a certain restricted setting, or use the randomized bloom-filter based approach of [21] (with a finer-grained analysis) to achieve the desired running time. This approach gives us the proof of Theorem 3.10 and we provide the detailed proof in the full version of the paper.

3.3 Handling Isolated Vertices

In the last two subsections we considered the graphs that have no isolated vertices. It remains to prove that we can obtain a tight classification even for patterns with isolated vertices. In particular, we prove the following theorem.

▶ **Theorem 3.13.** *Let P be any pattern graph with k vertices and $1 \leq r \leq k$ isolated vertices. There is a randomized algorithm that enumerates all dominating P-patterns in time $t_P(n,m) \cdot m^{\frac{\omega-2}{2}+o(1)}$ with high probability. If $\omega = 2$, this is $t_P(n,m)^{1+o(1)}$.*

We again consider two cases separately, namely when $r \geq 2$ and $r = 1$. In the first (simpler) case of $r \geq 2$, we aim to reduce to the Maximal r-Independent Set problem, which is known to have efficient algorithm in sparse graphs (see [30]). We enumerate all subgraphs of G that correspond to the induced subgraph of P that does not contain isolated vertices, and then after removing the closed neighborhood of the enumerated subgraph, we run the Maximal

r-Independent Set algorithm on the remaining part. By a careful analysis, we obtain the desired time. We provide a sketch of the proof here, for the full proof, see the full version of the paper.

▶ **Lemma 3.14.** *Let P be a pattern with $2 \leq r \leq k$ isolated vertices. Then there exists a randomized algorithm solving P-Dominating Set in time $\left(t_P(n,m) \cdot m^{(\omega-2)/2}\right)^{1+o(1)}$ with high probability. If $\omega = 2$, this is equal to $(t_P(n,m))^{1+o(1)}$.*

Proof (sketch). If $r = k$, then this problem is precisely the Maximal k-Independent Set problem, and by [30] can be solved in the time bounded by $\left(\frac{m^{(k-1)/2+\omega/2}}{n}\right)^{1+o(1)}$, as desired. Hence, we may assume that $2 \leq r \leq k-2$ (if P contains at least one edge, then at least two vertices have degree ≥ 1). Let $X := \{x \in V(P) : \deg_P(x) \geq 1\}$ and $Y := V(P) \setminus X$ be the set of non-isolated and isolated vertices in P respectively. Let $S, N(S), R$ be the partition of $V(P)$ as in Definition 1.1. We can enumerate all induced subgraphs of G isomorphic to $P[X]$ in some time $\mathcal{O}\left(t^{(1)}_{P[X]}(n,m)\right)$ (the ratio of the size of this set to $t_{P[X]}(n,m)$ depends only on whether the set S is empty or not). For each enumerated subgraph H, we appeal to the randomized Dominating r-Independent Set algorithm from [30] on the graph $G - N[H]$ that runs in $\mathcal{O}\left(t^{(2)}_r(n,m)\right)$ (depends on which vertex from P is mapped to a heavy vertex). It is straightforward to see that this algorithm is correct. We conclude the proof by showing that the product of $t^{(1)}_{P[X]}(n,m)$ and $t^{(2)}_r(n,m)$ is in all cases bounded by $t_P(n,m)$ (see the full version of the paper for details). ◀

Things get slightly more complicated when dealing with patterns that only have one isolated vertex. In particular, the preprocessing part of the algorithm above takes in the worst case $\Theta\left(t_{P[X]}(n,m) \cdot m\right)$, while for pattern graphs with only one isolated vertex, we have $t_P(n,m) = \mathcal{O}\left(t_{P[X]}(n,m) \cdot \sqrt{m}\right)$. To circumvent this overhead, we employ a more careful analysis and use a hashing-based approach similar to that in Section 3.2. We first provide an algorithm for two special cases of some of the smallest patterns that exhibit this structure and then the idea is to reduce any pattern to one of these special cases.

▶ **Lemma 3.15.** *Let G be a graph with n vertices and m edges, and let $Y \subseteq V(G)$ be a subset of vertices of G. Then we can construct an algorithm that w.h.p. enumerates all choices of three vertices u, v, w in time $m^{(\omega+1)/2+o(1)}$, such that the following is satisfied.*

- *The vertex w is contained in Y and satisfies $\deg(w) < \sqrt{m}$.*
- *Vertices u and v are adjacent in G and both of them are non-adjacent to w in G (i.e. the induced subgraph $G[\{u, v, w\}]$ is isomorphic to an edge and an isolated vertex).*
- *$Y \subseteq N[u] \cup N[v] \cup N[w]$.*

Proof. Without loss of generality, assume that $\deg(u) \geq \deg(v)$. Let t be such that $t \leq \deg(u) \leq 2t$. Assume first that $t \leq \sqrt{m}$. Then each vertex u, v, w has degree bounded by $\mathcal{O}(\sqrt{m})$ and so, if there exists a valid triple that dominates Y, the size of Y must also be bounded by $\mathcal{O}(\sqrt{m})$. Hence by Lemma 3.1 we can enumerate all valid triples in time $\mathrm{MM}(m, \sqrt{m}, \sqrt{m}) \leq m^{(\omega+1)/2+o(1)}$, as desired.[7] For the rest of this proof we assume that $t \geq \sqrt{m}$. We now proceed to enumerate all valid triples u, v, w such that $\deg(v) \leq \sqrt{m}$. To this end, we run the following routine. For each of the $\mathcal{O}(m/t)$ valid choices for the vertex u, construct the tripartite graph $H_u = (V^{(1)}_u \cup V^{(2)}_u \cup V^{(3)}_u, E_u)$, where $V^{(1)}_u$ corresponds to a

[7] Since vertices u, v are adjacent, there are at most m choices for this pair, and w is by assumption in Y, and $|Y| \leq \mathcal{O}(\sqrt{m})$

copy of $N(u)$, while $V_u^{(2)}$ and $V_u^{(3)}$ each correspond to a copy of the set $Y - N(u)$. The set of edges E_u is constructed naturally, that is, for each pair $x \in V_u^{(i)}, y \in V_u^{(j)}$ for $i \neq j$ we add an edge xy if and only if the edge between the corresponding vertices is present in G (we assume that each vertex is adjacent to itself in G). It is easy to see that for any fixed u, any valid choice of v, w corresponds precisely to a choice of $v \in V_u^{(1)}$ and $w \in V_u^{(2)}$ such that the following two conditions are satisfied: 1) $V_u^{(3)} \subseteq N(w) \cup N(v)$ and 2) $vw \notin E(H_u)$. Now, since we are only enumerating triples u, v, w where both v, w have degrees at most $\sqrt{m}$, if for a fixed u the set $V_u^{(3)}$ contains more than $2\sqrt{m}$ vertices, we can conclude that no pair v, w satisfies both conditions and proceed with the next choice of u. On the other hand, if $|V_u^{(2)}| = |V_u^{(3)}| \leq 2\sqrt{m}$, by Lemma 3.1, for any fixed u, we can enumerate all the valid pairs of non-adjacent vertices v, w that dominate $Y - N(u)$ in time

$$\begin{aligned} T_u(n,m) &\leq \mathrm{MM}\left(V_u^{(1)}, V_u^{(3)}, V_u^{(2)}\right) \\ &\leq \mathrm{MM}\left(t, \sqrt{m}, \sqrt{m}\right) \\ &\leq \mathcal{O}\left(\frac{t}{\sqrt{m}}\right) \mathrm{MM}\left(\sqrt{m}, \sqrt{m}, \sqrt{m}\right) && (t \geq \sqrt{m}) \\ &\leq t \cdot m^{\frac{\omega-1}{2}+o(1)}. \end{aligned}$$

Repeating this for each of the possible $\mathcal{O}(m/t)$ many choices of vertex u yields the desired time. It remains to enumerate the triples of vertices u, v, w where $\deg(v) \geq \sqrt{m}$. Recall also that $2t \geq \deg(u) \geq \deg(v)$ and $t \geq \sqrt{m}$. Hence, since by the first condition of the Lemma, we are only interested in the triples u, v, w with $\deg(w) \leq \sqrt{m} \leq t$, if the set Y contains more than $5t$ vertices, we can conclude that no triple u, v, w satisfies $Y \subseteq N[u] \cup N[v] \cup N[w]$. Hence, we can assume that $Y \leq 5t$. Let B be a matrix whose rows correspond to adjacent pairs of vertices u, v with $t \leq \deg(u) \leq 2t$ and $\deg(v) \geq \sqrt{m}$ and whose columns correspond to the set Y. We set the entry $B[(u,v), y]$ to 0 if either u or v are adjacent to y and to 1 otherwise. Similarly, we construct a matrix C whose columns correspond to the set of vertices $w \in Y$ such that $\deg(w) \leq \sqrt{m}$ and the rows correspond to Y. Set the entry $B[y, w]$ to 1 if w and y are adjacent (we understand that each vertex is adjacent to itself) and 0 otherwise. Notice that u, v, w dominate Y if and only if $(B \cdot C)[(u,v), w] = \sum_k B[w,k]$ (each vertex that is not dominated by the pair (u,v) is dominated by w). Also notice that the matrix B is an $\mathcal{O}\left(\frac{m\sqrt{m}}{t}\right) \times \mathcal{O}(t)$ and C is an $\mathcal{O}(t) \times \mathcal{O}(t)$ matrix with at most m non-zeros. Hence by a Lemma from [21] (see full version for details), we can enumerate with high probability all valid triples u, v, w in time $\sqrt{m} \cdot m^{\omega/2+o(1)} = m^{\frac{\omega+1}{2}+o(1)}$, as desired. We remark that we enumerate all valid triples for any fixed t in the claimed time. To enumerate all the valid triples, simply run the previous algorithm $\mathcal{O}(\log(n))$ times, once for each $\ell \in 1, \ldots, \lfloor \log(n) \rfloor$, setting $t = 2^\ell$. ◀

An almost identical argument as in previous lemma can also be used to show that we can efficiently enumerate the solutions that are isomorphic to $C_{2r+1} + K_1$.

▶ **Lemma 3.16.** *Let G be a graph with n vertices and m edges, and let $Y \subseteq V(G)$ be a subset of vertices of G. Then we can construct an algorithm that w.h.p. enumerates all choices of $2r+2$ vertices $x_1, \ldots, x_{2r+1}, w$ in time $m^{r+\omega/2+o(1)}$, such that the following is satisfied.*

- *The vertex w is contained in Y and satisfies $\deg(w) < \sqrt{m}$.*
- *Vertices $x_1, \ldots, x_{2r+1}$ form a cycle in G (not necessarily induced) and w is not adjacent to any of the vertices x_i in G.*
- *$Y \subseteq N[w] \cup N[x_1] \cup \cdots \cup N[x_{2r+1}]$.*

The idea for any pattern with precisely one isolated vertex v is to decompose the pattern into sets $S, N(S), R$ as in Definition 1.1, and then depending on whether the set S is empty or not, we either directly reduce to the setting of Lemma 3.15, or first apply Lemma 3.4 to decompose R into edges and odd cycles, and then reduce to either the setting of Lemma 3.15, or that of Lemma 3.16. For the lack of space, we give the detailed proof in the the full version of the paper.

▶ **Proposition 3.17.** *Let P be a pattern with one isolated vertex. Then there exists a randomized algorithm solving P-Dominating Set in time $\mathcal{O}(t_P(n,m)) \cdot m^{\frac{\omega-2}{2}+o(1)}$.*

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.

2 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.

3 Noga Alon. On the number of subgraphs of prescribed type of graphs with a given number of edges. *Israel Journal of Mathematics*, 38(1):116–130, March 1981. doi:10.1007/BF02761855.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

5 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed maximal matching and maximal independent set on hypergraphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2632–2676. SIAM, 2023. doi:10.1137/1.9781611977554.CH100.

6 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 743–754. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.74.

7 Victor Chepoi. A note on r-dominating cliques. *Discrete mathematics*, 183(1-3):47–60, 1998. doi:10.1016/S0012-365X(97)00076-9.

8 Eun-Kyung Cho, Ilkyoo Choi, Hyemin Kwon, and Boram Park. A tight bound for independent domination of cubic graphs without 4-cycles. *Journal of Graph Theory*, 104(2):372–386, 2023. doi:10.1002/JGT.22968.

9 Eun-Kyung Cho, Ilkyoo Choi, and Boram Park. On independent domination of regular graphs. *Journal of Graph Theory*, 103(1):159–170, 2023. doi:10.1002/JGT.22912.

10 Eun-Kyung Cho, Jinha Kim, Minki Kim, and Sang-il Oum. Independent domination of graphs with bounded maximum degree. *Journal of Combinatorial Theory, Series B*, 158:341–352, 2023. doi:10.1016/J.JCTB.2022.10.004.

11 Gérard Cornuéjols and William R. Pulleyblank. Critical graphs, matching and tours or a hierarchy of relaxations for the traveling salesman problem. *Comb.*, 3(1):35–52, 1983. doi:10.1007/BF02579340.

12 Margaret B Cozzens and Laura L Kelleher. Dominating cliques in graphs. *Discrete Mathematics*, 86(1-3):101–116, 1990. doi:10.1016/0012-365X(90)90353-J.

13 Mina Dalirrooyfard and Virginia Vassilevska Williams. Induced cycles and paths are harder than you think. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 531–542. IEEE, 2022. doi:10.1109/FOCS54457.2022.00057.

14 Mina Dalirrooyfard, Thuy-Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster noninduced cycles. *SIAM J. Comput.*, 50(5):1627–1662, 2021. doi:10.1137/20M1335054.

15 Feodor F Dragan and Andreas Brandstädt. Dominating cliques in graphs with hypertree structure. In *STACS 94: 11th Annual Symposium on Theoretical Aspects of Computer Science Caen, France, February 24–26, 1994 Proceedings 11*, pages 735–746. Springer, 1994. doi:10.1007/3-540-57785-8_186.

16 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004. doi:10.1016/J.TCS.2004.05.009.

17 Yibin Fang and Liming Xiong. Circumference of a graph and its distance dominating longest cycles. *Discrete Mathematics*, 344(2):112196, 2021. doi:10.1016/j.disc.2020.112196.

18 Yibin Fang and Liming Xiong. On the dominating (induced) cycles of iterated line graphs. *Discrete Applied Mathematics*, 325:43–51, 2023. doi:10.1016/j.dam.2022.09.025.

19 Jill R. Faudree, Ralph J. Faudree, Ronald J. Gould, Paul Horn, and Michael S. Jacobson. Degree sum and vertex dominating paths. *J. Graph Theory*, 89(3):250–265, 2018. doi:10.1002/JGT.22249.

20 Ralph J. Faudree, Ronald J. Gould, Michael S. Jacobson, and Douglas B. West. Minimum degree and dominating paths. *J. Graph Theory*, 84(2):202–213, 2017. doi:10.1002/JGT.22021.

21 Nick Fischer, Marvin Künnemann, and Mirza Redzic. The effect of sparsity on k-dominating set and related first-order graph properties. In David P. Woodruff, editor, *Proc. 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 4704–4727. SIAM, 2024. doi:10.1137/1.9781611977912.168.

22 Teresa W Haynes and Peter J Slater. Paired-domination in graphs. *Networks: An International Journal*, 32(3):199–206, 1998. doi:10.1002/(SICI)1097-0037(199810)32:3\%3C199::AID-NET4\%3E3.0.CO;2-F.

23 Karthik C. S. and Subhash Khot. Almost polynomial factor inapproximability for parameterized k-clique. In Shachar Lovett, editor, *37th Computational Complexity Conference, CCC 2022, July 20-23, 2022, Philadelphia, PA, USA*, volume 234 of *LIPIcs*, pages 6:1–6:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CCC.2022.6.

24 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *J. ACM*, 66(5):33:1–33:38, 2019. doi:10.1145/3325116.

25 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000. doi:10.1016/S0020-0190(00)00047-8.

26 Ekkehard Köhler. Connected domination and dominating clique in trapezoid graphs. *Discret. Appl. Math.*, 99(1-3):91–110, 2000. doi:10.1016/S0166-218X(99)00127-4.

27 Dieter Kratsch. Finding dominating cliques efficiently, in strongly chordal graphs and undirected path graphs. *Discrete mathematics*, 86(1-3):225–238, 1990. doi:10.1016/0012-365X(90)90363-M.

28 Dieter Kratsch, Peter Damaschke, and Anna Lubiw. Dominating cliques in chordal graphs. *Discrete Mathematics*, 128(1-3):269–275, 1994. doi:10.1016/0012-365X(94)90118-X.

29 Kirsti Kuenzel and Douglas F Rall. On independent domination in direct products. *Graphs and Combinatorics*, 39(1):7, 2023. doi:10.1007/S00373-022-02600-0.

30 Marvin Künnemann and Mirza Redzic. Fine-grained complexity of multiple domination and dominating patterns in sparse graphs. In Édouard Bonnet and Pawel Rzazewski, editors, *Proc. 19th International Symposium on Parameterized and Exact Computation (IPEC 2024)*, volume 321 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.IPEC.2024.9.

31 Bingkai Lin, Xuandi Ren, Yican Sun, and Xiuhan Wang. Improved hardness of approximating k-clique under ETH. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 285–306. IEEE, 2023. doi:10.1109/FOCS57990.2023.00025.

32 Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6(1):85–112, 2010. doi:10.4086/TOC.2010.V006A005.

33 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

34 Shiwei Pan, Yiming Ma, Yiyuan Wang, Zhiguo Zhou, Jinchao Ji, Minghao Yin, and Shuli Hu. An improved master-apprentice evolutionary algorithm for minimum independent dominating set problem. *Frontiers of Computer Science*, 17(4):174326, 2023. doi:10.1007/S11704-022-2023-7.

35 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.

36 Mihai Pătraşcu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 1065–1075. SIAM, 2010. doi:10.1137/1.9781611973075.86.

37 Justin Southey and Michael A Henning. A characterization of graphs with disjoint dominating and paired-dominating sets. *Journal of combinatorial optimization*, 22(2):217–234, 2011. doi:10.1007/S10878-009-9274-1.

38 Daniel S Studer, Teresa W Haynes, and Linda M Lawson. Induced-paired domination in graphs. *Ars Combinatoria*, 57:111–128, 2000.

39 Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1671–1680. SIAM, 2015. doi:10.1137/1.9781611973730.111.

40 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and APSP. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 786–797. IEEE, 2020. doi:10.1109/FOCS46700.2020.00078.

41 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3792–3835. SIAM, 2024. doi:10.1137/1.9781611977912.134.

42 Henk Jan Veldman. Existence of dominating cycles and paths. *Discret. Math.*, 43(2-3):281–296, 1983. doi:10.1016/0012-365X(83)90165-6.

Engineering Minimal k-Perfect Hash Functions

Stefan Hermann
Karlsruhe Institute of Technology, Germany

Sebastian Kirmayer
Karlsruhe Institute of Technology, Germany

Hans-Peter Lehmann
Karlsruhe Institute of Technology, Germany

Peter Sanders
Karlsruhe Institute of Technology, Germany

Stefan Walzer
Karlsruhe Institute of Technology, Germany

Abstract

Given a set S of n keys, a k-perfect hash function (kPHF) is a data structure that maps the keys to the first m integers, where each output integer can be hit by at most k input keys. When $m = \lceil n/k \rceil$, the resulting function is called a *minimal* k-perfect hash function (MkPHF). Applications of kPHFs can be found in external memory data structures or to create efficient 1-perfect hash functions, which in turn have a wide range of applications from databases to bioinformatics.

Several papers from the 1980s look at external memory data structures with small internal memory indexes. However, actual k-perfect hash functions are surprisingly rare, and the area has not seen a lot of research recently. At the same time, recent research in 1-perfect hashing shows that there is a lack of efficient kPHFs. In this paper, we revive the area of k-perfect hashing, presenting four new constructions. Our implementations simultaneously dominate older approaches in space consumption, construction time, and query time. We see this paper as a possible starting point of an active line of research, similar to the area of 1-perfect hashing.

2012 ACM Subject Classification Theory of computation → Data compression; Theory of computation → Bloom filters and hashing; Information systems → Point lookups

Keywords and phrases Compressed Data Structures, Perfect Hashing

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.99

Related Version *Extended Version*: `https://arxiv.org/abs/2504.20001` [18]

Supplementary Material *Software*: `https://github.com/stefanfred/engineering-k-perfect-hashing` [19], archived at `swh:1:dir:e6756f018691a80adc68d839fd3617d8f5e2d0b0`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500). This work was also supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF).

Acknowledgements This paper is based on and has text overlaps with the bachelor's thesis of Sebastian Kirmayer [22].

1 Introduction

Given a set S of n keys, a k-perfect hash function (kPHF) is a data structure that maps the keys to the first m non-negative integers $[m]$, called *bins* in the following. Each output bin can be hit by at most k input keys. The output for any key not in S is undefined, so a k-perfect hash function does not need to store the input set. For $k = 1$, there is a wide

© Stefan Hermann, Sebastian Kirmayer, Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer; licensed under Creative Commons License CC-BY 4.0

range of constructions covering different trade-offs between space consumption, query time, and construction time. The area is an active field of research with several publications each year [27, 28, 30, 31, 9, 20, 16, 3, 29, 17, 37]. However, there has been little prior work on k-perfect hashing. While several papers from the 1980s look at internal memory indexes for external hash tables, these are not full k-perfect hash functions. We refer to Section 3 for details. In this paper, we present four new k-perfect hash function constructions. Each covers a different trade-off between the three main performance metrics – space consumption, construction time, and query time.

Minimality. When $m = \lceil n/k \rceil$, we obtain a *minimal* k-perfect hash function (MkPHF) [26]. When using the MkPHF to partition 1-PHFs [30, 31], it might be beneficial to look at a stricter definition where all output bins contain exactly k keys, except possibly the last. The approaches presented in this paper fulfill the stricter definition. An alternative definition of non-minimal kPHFs could be that we could fill each output bin with either exactly k keys or 0 keys. For $k = 1$, this results in an ordinary non-minimal PHF and achieves a significant reduction in space consumption. However, for $k > 1$, the resulting space consumption is the same as the one of MkPHFs [33].

Applications. k-perfect hashing has a wide range of applications. In external memory hash tables, kPHFs can return the page on which a key is stored. External memory is usually accessed with page granularity anyway, so having to scan the page for the key only causes negligible overhead. Compared to a 1-PHF, however, this can significantly reduce the space needed. For internal memory hash tables it can be beneficial to store more than one key in each table cell [12]. Using an MkPHF, we can completely fill such a hash table. A k-perfect hash function can also be used to make the construction of 1-perfect hash functions more uniform by dividing the input to same-size partitions [30, 31].

Our Contributions. In this paper, we present four new or significantly improved k-perfect hash function constructions. Threshold-based bumping was previously only briefly described as an ad-hoc solution to aid 1-perfect hashing [30, 31]. In Section 4, we extend it through optimal thresholds and CONSENSUS-coded [31] thresholds. Perfect hashing through bucket placement [37, 20, 2] generalizes to k-perfect hashing in an obvious way. In Section 5, we introduce an optimal bucket assignment function. RecSplit [9] is a known 1-PHF, which we extend to a k-PHF in Section 6. Finally, we introduce a variant of the external memory hash table PaCHash [23] that can be used as an MkPHF in Section 7. To the best of our knowledge, we then give the first experimental evaluation of different k-perfect hash functions in Section 8.

2 Preliminaries

In this section we show the space lower bound for minimal k-perfect hash functions. We then explain preliminary data structures that we use throughout this paper.

Space Lower Bounds. The space lower bound for representing a minimal k-perfect hash function is rather easy to prove [32, 2, 23]. Let the n input keys be a subset of a universe of size u. There are $\binom{u}{n}$ possible input sets. Each behavior of an MkPHF can cover at most $\binom{u/(n/k)}{k}^{n/k}$ input sets. Therefore, the number of bits we need is at least

Table 1 Space lower bound for different values of k.

k	1	2	4	10	100	1000
Bits per key	1.443	0.943	0.589	0.300	0.046	0.006

$$\log_2 \left(\frac{\binom{u}{n}}{\binom{u/(n/k)}{k}^{n/k}} \right) \overset{n^2 \in o(u)}{\approx} n \cdot \left(\log_2(e) - \log_2(k^k/k!)/k \right) = n \cdot \left(\frac{\log_2(2\pi k)}{2k} + \Theta\left(\frac{1}{k^2}\right) \right).$$

In Table 1, we give a range of example values. The space lower bounds become more complex for non-minimal k-perfect hashing, and no tight bounds are known yet [2].

Retrieval Data Structures. A retrieval data structure stores a static function $f : S \to \{0,1\}^r$ with r-bit output values. It can return arbitrary results for any key not in S. This makes it possible to represent it using just rn bits. Bumped Ribbon Retrieval (BuRR) [7] is based on solving a system of linear equations and has a space consumption of $1.01rn$ bits in practice.

Elias-Fano Coding. Let $(a_i)_{i\in[n]}$ be a monotonically increasing sequence of integers. To store this sequence with Elias-Fano coding [8, 11], we split each integer into two parts. We store the lower $L = \lceil \log_2(a_{n-1}/n) \rceil$ bits directly in a packed array. Then we store the remaining upper bits $(h_i)_{i\in[n]}$ as a bit vector where the bits at indices $h_i + i$ are 1, and all other bits are 0. On this bit vector, we have $h_i = \text{select}_1(i) - i$. Together with a lookup in the packed array, we get constant access time. In total, storing the sequence needs $n \cdot (2 + \lceil \log_2(a_{n-1}/n) \rceil) + o(n)$ bits of space [8, 11].

Golomb-Rice Coding. For a parameter L, Golomb-Rice coding [14, 38] divides an integer x into the two parts $h = \lfloor x/2^L \rfloor$ and $l = x \bmod 2^L$. It encodes l using L bits in binary coding, and h in unary coding. Golomb-Rice coding is optimal for geometrically distributed x with $p = 2^{-L}$. We can store a sequence of integers with constant time access by using a common array for the lower bits, concatenating the upper bits, and using a select data structure.

3 Related Work

We now explain k-perfect hash functions and similar external memory hashing data structures from the literature. Note that Alon et al. [1] and Berman et al. [4] use the variable k for the range of the output values in 1-perfect hashing, which is now usually called m. In an online setting when building hash tables, Frei and Wehner [13] call a family of hash functions k-*ideal* if there is, for any input set of size n, some hash function that is k-perfect.

3.1 k-Perfect Hash Functions

Brute-Force Search. Similar to what can be done for 1-perfect hashing [34], we can use brute-force search to construct k-perfect hash functions. We use a hash function that can be parameterized by a seed value, changing its behavior completely. We greedily start trying seed values until we come across one seed where the hash function is k-perfect on our given input set. Then storing the index of that seed in binary coding is enough to represent the k-perfect hash function. Unfortunately, like for 1-perfect hashing, we need an exponential

number of hash function evaluations, making the approach unfeasible for large input sets. Mairson [32] shows that the probability of a random function being an MkPHF can be described by the multinomial coefficient $(n/k)^{-n}\binom{n}{\underbrace{k, k, \ldots, k}_{n/k \text{ times}}}$.

Bucket Placement. Perfect hashing through bucket placement [37, 20, 2] is a class of 1-perfect hash functions. The idea is to hash the input keys to small buckets of expected size λ, where λ is a tuning parameter usually in the range 2–6 keys. For each bucket, we then greedily search for a hash function seed such that the keys in that bucket do not collide with previous keys. After placing all buckets, we compress the list of seeds while retaining fast access. The first buckets are easier to place because all bins are empty. It is therefore helpful to insert the buckets in decreasing order of size. In CHD [2], the keys are hashed uniformly to the buckets. Despite sorting, the last buckets are still much harder to place than the first. Therefore, PHOBIC [20] introduces an optimal *bucket assignment function* $\beta : [0, 1] \to [0, 1]$. A key with a normalized hash value $x \in [0, 1]$ is then assigned to bucket $\lceil \beta(x)n/\lambda \rceil$. The bucket assignment function makes the first buckets much larger to allow even smaller buckets towards the end, resulting in roughly the same success probability of a seed in all buckets.

Bucket placement can be used as a k-perfect hash function by allowing up to k collisions for each output bin. The implementation [6] of CHD supports only non-minimal k-perfect hashing. In Section 5, we extend the idea through an optimized bucket assignment function.

Predecessor Dictionary. Pagh [36] suggests hashing all input keys with a hash function of range $[n^3]$ and sorting them by hash value. Now map the keys linearly to the output bins and store the smallest hash value in each bin. For a given hash value x, a *predecessor query* can give the largest value y in the list such that $y \leq x$. The index of that value is the k-perfect output bin. We implement a variant in Section 7.

Threshold-Based Bumping. Threshold-based bumping [30] is an ad-hoc solution to k-perfect hashing introduced for simplifying a 1-perfect hash function. The idea is to first hash all keys to $\lceil 0.9n/k \rceil$ bins. The constant of 0.9 is selected arbitrarily in the paper, ensuring that there are few bins that receive fewer than k keys. An additional hash function uniformly assigns a fingerprint value $\in (0, 1]$ to each key. For each bin, threshold-based bumping then stores a threshold such that at most k keys in the bin have fingerprints below or equal to the threshold. It *bumps* keys with fingerprints larger than the threshold to a second layer of the same data structure. To reduce the space consumption and to enable fast access, we restrict ourselves to a small list of possible thresholds and only store the index into the list. Threshold-based bumping has similarities with perfect hashing through fingerprinting [35] (which essentially uses 1-bit thresholds) and separator hashing [15, 25] (see Section 3.2). The idea of bumping is also used in BuRR [7], which solves a system of linear equations and bumps conflicting rows.

To make the resulting function *minimal* k-perfect, keys bumped in the second layer have to be placed into the empty slots left in the bins. Threshold-based bumping does that by storing an Elias-Fano coded list of bins that still have slots left. Using a minimal 1-PHF on the bumped keys, it can map each key to one of the empty slots. Refer to Figure 1 for an illustration. Queries hash the key to a bin. Most keys can immediately return the bin index because their fingerprint is below the threshold. Some have to look at the next layer and few have to query the fallback 1-PHF. In Section 4, we extend the approach through an optimal selection of threshold values, as well as more efficient overloading to minimize the number of bins with less than k keys.

3.2 External Memory Hash Tables

In external memory hash tables, a goal is to find objects while only accessing a small number of external memory pages. For this, different internal memory index data structures have been proposed. These index data structures are very similar to k-perfect hash functions. The main difference is that they either need to inspect more than one external memory page or that they do not fully utilize the external memory, meaning that they are not *minimal* k-perfect. Still, these data structures use similar ideas as MkPHFs or can even be adapted to MkPHFs.

Separator Hashing. Separator hashing [15, 25, 24] is a static, external memory hash table that guarantees a single disk access for each query. Each key has a *probe sequence* of output bins and a *fingerprint sequence*, both determined through a sequence of hash functions. For each external memory page, it stores a threshold value for the fingerprint, also called *separator*. When querying a key, we look at the bins along its probe sequence. If the key's fingerprint for that bin is smaller than the separator, we found the corresponding page. Otherwise, we continue scanning. A problem with the technique, especially for high load factors, is that bumped keys map back into the same range of bins. This causes a series of dependent memory accesses during queries, and complicates the construction.

External Robin-Hood Hashing. Instead of displacing keys based on their fingerprint, *external robin-hood hashing* [5] displaces the keys that are at the smallest index of their probe sequence. The internal memory then stores, for each page, the smallest distance of a contained key to its home address. Queries iterate over the distance array starting from their home address until the distance is at least what is stored in the array. If multiple keys have the same home address, more than one external memory page has to be searched.

Extendible Hashing. The idea of *extendible hashing* [10] is to store an internal memory table holding pointers. Each row points to an external memory page holding the keys hashed to that table row. If the keys hashed to several (adjacent) table rows fit onto an external memory page together, extendible hashing stores the same pointers multiple times. The approach achieves a load factor of about 69% [36].

PaCHash. PaCHash [23] is designed as a static external memory hash table with variable-length keys. However, we only explain the case that each external memory page can hold k fixed-length keys. Then PaCHash uses a hash function of range $[a \cdot n/k]$, where a is a tuning parameter. This divides the keys into different buckets.[1] For each external memory page, we store the index of the first bucket overlapping it. Using Elias-Fano coding (see Section 2), this takes $n/k(2 + \log(a))$ bits. To locate a key, we perform a predecessor query on that sequence to determine which external memory page should be loaded. If a bucket overlaps the page boundaries, the index cannot determine whether the key is on the page or on the previous page. In that case, which happens for roughly $1/a$ of the queries, PaCHash loads both adjacent pages. In Section 7, we explain how we can adapt PaCHash to an MkPHF.

[1] For consistency within this paper, we interchange the terms *bin* and *bucket* of the PaCHash paper.

Listing 1 Query algorithm of k-perfect hashing trough threshold-based bumping. Hash function h_X uniformly maps into the set X.

```
Function query(key ∈ S)
  numBins := ⌈(n/k)/γ⌉
  bin := h_[numBins](key)
  thresholdIndex := thresholds[bin]
  fingerprint := h_[0,1](key)
  if fingerprint ≤ T_thresholdIndex
    return bin
  else // bumped to next layer
    return numBins + query next layer
```

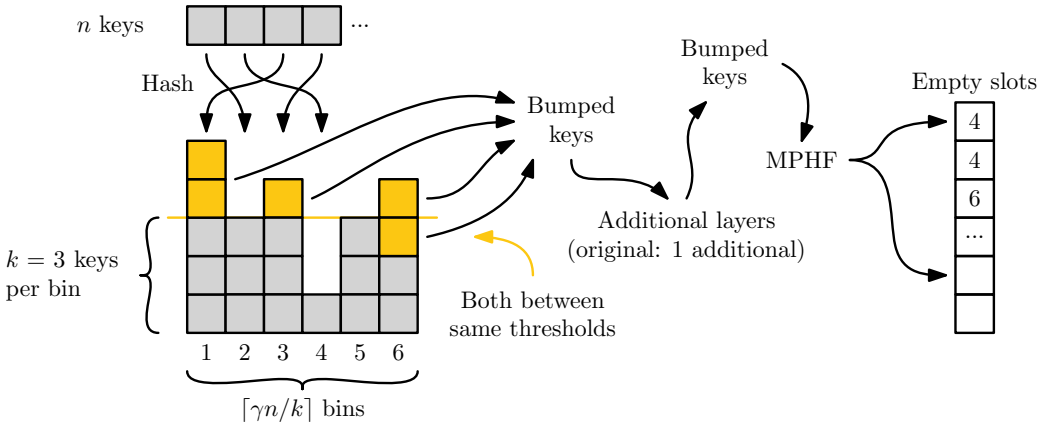


Figure 1 Threshold-based k-perfect hashing. Figure adapted from [30].

4 k-Perfect Hashing Through Threshold-Based Bumping

Threshold-based bumping is introduced in ShockHash-Flat [30] as an ad-hoc implementation to aid 1-perfect hashing. In this section, we enhance it significantly. Remember from Section 3.1 that threshold-based bumping hashes each key directly to a bin. Additionally, we hash each key to determine a uniform fingerprint value $\in (0, 1]$. Each bin stores a threshold such that at most k keys with fingerprints smaller than or equal to the threshold remain. The other keys are *bumped* to another layer of the same data structure. Figure 1 gives an illustration.

4.1 Overloading

To make it more likely that each bin receives at least k keys, we *overload* the bins to contain more than k keys in expectation. While the ad-hoc implementation [30] already overloads the first layer, we now take this to its logical conclusion and overload all layers. More concretely, we start with $\lceil \frac{n}{k\gamma} \rceil$ bins where $\gamma > 1$ is the overloading factor. We determine the thresholds to store, which leaves us with n' bumped keys. From the remaining output bins, we then use $\lceil \frac{n'}{k\gamma} \rceil$ bins for the next layer. We continue this as long as the total number of bins used is still smaller than $\lceil \frac{n}{k} \rceil$. Only for the remaining keys, we have to use 1-perfect hashing to puzzle them back into the under-filled bins. Even with overloading, we maintain expected constant query times and expected linear construction times. Preliminary experiments

with $k = 1000$ show that overloading all layers reduces the number of keys bumped in the last layer from 0.2% (with the original two layers) to 0.02% (with 5 layers). Because the fallback 1-PHF needs non-negligible space, this reduces the overall space consumption. In our implementation, we use the 1-PHF FiPS [27] due to its small constant space overhead when using it for small sets.

4.2 Optimizing the Thresholds

For fast access we use a fixed number of bits to represent the threshold of each bin, which gives us t possible threshold values. In the following we consider the value of those t thresholds. For each bin we then only store the index of the threshold. We model the fingerprint of a key as a uniformly distributed hash value $x \in (0, 1]$. If x is greater than the threshold of the key's bin, the key is bumped. It is less likely that we need threshold values close to 0 because the expected number of keys in each bin is $k\gamma$. The ad-hoc implementation [30] therefore heuristically uses uniformly spaced thresholds in $[\frac{2}{3}, 1]$. In this section, we derive an optimal selection of thresholds.

Our goal is to find the t thresholds $T_1 < \cdots < T_t \in [0, 1]$ such that they minimize the expected number of empty slots. As a fallback option, we use $T_1 = 0$, bumping all keys. We also use $T_t = 1$. A value of $T_t < 1$ would mean that keys in the range $(T_t, 1]$ always get bumped, which would counteract our increased load factor. Therefore, we need to determine the values of the remaining $t - 2$ thresholds. The number of keys in a bin is binomially distributed. To simplify analysis, we approximate the number of keys using a Poisson distribution with γk keys in expectation. We consider the sorted fingerprints $0 < x_1 < x_2 < \dots$ of the keys. The distance between any two consecutive keys $(x_{i+1} - x_i)$ and between 0 and x_1 is exponentially distributed with an expected distance of $1/(\gamma k)$. Hence, it is equivalent to obtain the fingerprints by sampling consecutive fingerprints using this exponential distribution until we pass 1. However, for notational convenience we sample an infinite number of fingerprints in this way and allow them to exceed 1. This does not change the distribution of the fingerprints smaller 1.

To minimize the number of empty slots we always choose the highest possible threshold that is still smaller than x_{k+1}, since we need to ensure that at most k keys remain in the bin. Hence, threshold T_i is chosen if $T_i < x_{k+1} \leq T_{i+1}$ for $i \in [t]$ and we formally set $T_{t+1} := \infty$ to simplify notation. A fingerprint x_j results in an empty slot if it is bumped ($x_j > T_i$) *and* it is one of the k keys which we want to keep in the bin ($x_j < x_{k+1}$). If we know x_{k+1}, then the k smaller fingerprints are uniformly distributed in the range $(0, x_{k+1})$. Hence, the expected number of empty slots is $k\frac{x_{k+1}-T_i}{x_{k+1}}$. The fingerprint x_{k+1} follows a gamma distribution with shape $k + 1$ and rate γk. Let $\phi(x)$ be the density function of that gamma distribution. We can calculate the expected number of empty slots of the bin as

$$\begin{aligned} E := \mathbb{E}[\text{empty slots}] &= \int_0^\infty \mathbb{E}[\text{empty slots} \mid x_{k+1} = s]\phi(s)\mathrm{d}s \\ &= \sum_{i=1}^{t} \int_{T_i}^{T_{i+1}} \mathbb{E}[\text{empty slots} \mid x_{k+1} = s]\phi(s)\mathrm{d}s \\ &= \sum_{i=1}^{t} \int_{T_i}^{T_{i+1}} k\frac{s - T_i}{s}\phi(s)\mathrm{d}s \end{aligned} \tag{1}$$

For E to be minimal, its derivative with respect to $T_2, \ldots, T_{t-1}$ must be zero. We have:

$$\begin{aligned} 0 \stackrel{!}{=} \frac{\mathrm{d}E}{\mathrm{d}T_i} &= \frac{\mathrm{d}}{\mathrm{d}T_i}\int_{T_{i-1}}^{T_i} k\frac{s-T_{i-1}}{s}\phi(s)\mathrm{d}s + \frac{\mathrm{d}}{\mathrm{d}T_i}\int_{T_i}^{T_{i+1}} k\frac{s-T_i}{s}\phi(s)\mathrm{d}s \\ &= k\left(\frac{T_i - T_{i-1}}{T_i}\phi(T_i) - \int_{T_i}^{T_{i+1}} \frac{1}{s}\phi(s)\mathrm{d}s\right) \\ &\iff T_{i-1} = T_i - \frac{T_i}{\phi(T_i)}\int_{T_i}^{T_{i+1}} \frac{1}{s}\phi(s)\mathrm{d}s \end{aligned} \tag{2}$$

If we knew T_t and T_{t-1} we could inductively obtain the remaining thresholds. However, we only know that $T_t = 1$ and $T_1 = 0$. Our implementation therefore fixes $T_t = 1$ and uses binary search of T_{t-1} until T_1 is close to 0. This minimizes the number of empty slots under the necessary conditions $T_t = 1$ and $T_1 = 0$. For better intuition, we show in the full version of this paper [18] that the density of thresholds follows a gamma distribution for $t \to \infty$. The query algorithm of our data structure is shown in Listing 1.

4.3 Tighter Packing With Retrieval

Sometimes there is more than one key between the relevant thresholds. If we take the smaller threshold, we keep empty slots that we later need to repair. If we take the larger threshold, we would have more than k keys in the bin. In this case, we extend the data structure by querying a 1-bit retrieval data structure (see Section 2) on all keys that have fingerprints between these two thresholds. The bit tells us whether this key should be kept in the bin or if it should be bumped. This costs us 1 bit per key for all keys between thresholds, even the ones we do *not* bump. However, overall, this reduces space because each bumped key leaving an empty slot costs us several bits of space later to repair. The technique trades some query time for reduced space consumption.

4.4 CONSENSUS-Coded Thresholds

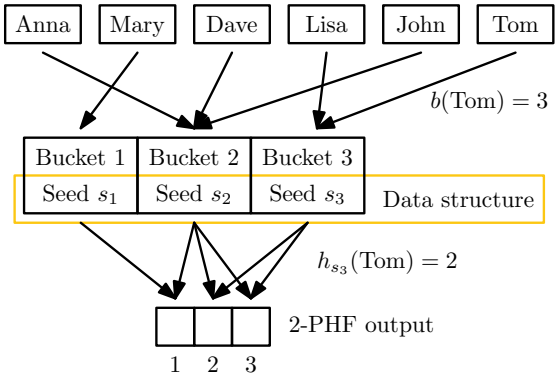
CONSENSUS [31] is a technique for space-efficiently storing seeds for randomized data structures. It basically uses a fixed number of bits to store the seed of each subtask. This means that for some subtasks, there might be multiple successful seeds that can be represented with these bits, while for other tasks, there might be none. The novel idea of CONSENSUS is to concatenate the current seed with previous seeds. This means that if we change one seed, we completely change the meaning of all following seeds. That way, if one task has no successful seeds, we can backtrack to another successful seed in a previous task and get a completely new chance for success in the current task.

While the thresholds in threshold-based k-perfect hashing are not classical seeds, we can still apply the CONSENSUS idea. The similarity is that for some bins, we might have more than one threshold that gives a sufficiently small number of empty slots, while for other bins, there might be none. We now use the thresholds of previous bins as a seed value when determining the fingerprints of the next bin. Selecting a different threshold in a previous bin therefore gives us a new chance to reduce the number of empty slots.

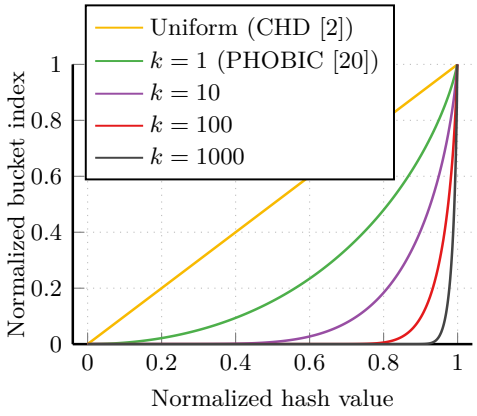
A problem with this approach is that there might be some bins that are much larger than γk caused by uniformly hashing keys to bins. Even if we try another hash function seed, the bins are biased towards having too many keys. Therefore, finding a seed such that exactly k keys are below one of the thresholds is unlikely. However, we still have the option to bump too many keys and repair the empty slots later. We need this option anyway because there

Listing 2 Query algorithm of k-perfect hashing trough bucket placement. Hash function h_X uniformly maps into the set X.

```
Function query(key ∈ S)
  bucket := ⌊(n/λ) · β_k(h_[0,1](key))⌋
  seed := seeds[bucket]
  return h_[⌈n/k⌉](key, seed)
```



(a) Illustration of the data structure using $k = 2$.



(b) Bucket assignment functions for different values of k.

Figure 2 Minimal k-perfect hashing through bucket placement.

might be bins that are not full enough. Therefore, for each actual size of a bin, we determine how many keys we have to accept bumping if we want to have at least one valid threshold in expectation. We then accept a seed if the number of empty slots is less than this limit.

5 k-Perfect Hashing Through Bucket Placement

Remember from Section 3.1 that perfect hashing through bucket placement [37, 20, 2] first hashes n keys to $\frac{n}{\lambda}$ buckets. For each bucket, it then searches for a hash function seed that can place the keys without collisions. The *bucket assignment function* $\beta : [0,1] \to [0,1]$ varies the expected size of each bucket such that placing each bucket has the same success probability. Figure 2a gives an illustration of the idea and Listing 2 describes the query algorithm. The idea generalizes trivially to k-perfect hashing, where we now place a bucket if all output bins stay below k collisions. Because k-perfect hashing has different success probabilities than 1-perfect hashing, we need a different bucket assignment function β_k. In this section we show how to choose β_k such that each bucket has approximately the same success probability.

We first need some general observations about bucket assignment functions. For key x, the expected size of its bucket with normalized index $\beta_k(x)$ is $\lambda/\beta_k'(x)$.[2] Let us assume that an x-fraction of keys is already mapped to bins. Then let $p_k(x)$ be the probability that a

[2] In the close neighborhood of x and for infinitesimal δ, a δ-fraction of the hash range (used by δn keys in expectation) is shared by a $(\beta_k'(x) \cdot \delta)$-fraction of the $\frac{n}{\lambda}$ buckets. The quotient is $\delta n/(\beta_k'(x)\delta\frac{n}{\lambda}) = \lambda/\beta_k'(x)$ [20].

random bin is not full at this point in the construction process. When placing a bucket of size s, a seed is successful if all keys map to bins that are not full yet which has a probability of $p_k(x)^s$ (ignoring self collisions). The optimal solution for $k = 1$ satisfies that $p_k(x)^{\lambda/\beta'_k(x)}$ is some constant [20]. Because λ is a constant, the equation simplifies to $p_k(x)^{1/\beta'_k(x)} = C_k$. Rearranging gives

$$\beta'_k(x) = \ln(p_k(x))/\ln(C_k). \tag{3}$$

For $k = 1$, a bin is full if there is a single key mapped to it. We therefore have $p_1(x) = 1 - x$ and $\beta'_1(x) = \ln(1 - x)/\ln(C_1)$. The constant C_k has to be chosen such that all keys are placed after the last bucket, i.e. $\beta_k(1) = 1$. After integration and proper choice of C_1 we have $\beta_1(x) = x + (1 - x)\ln(1 - x)$, the optimal bucket assignment function for $k = 1$ [20]. The difficulty for $k > 1$ is finding $p_k(x)$. Once we have $p_k(x)$, we use Equation (3) to obtain the bucket assignment function. Note that we assume without proof that using Equation (3) is still optimal for $k > 1$.

Model. To determine $p_k(x)$, we model the following process where we insert bk keys into b bins of capacity k. We initialize a counter $c_i = 0$ for each bin $i \in [b]$. The keys are handled sequentially as described in the following. For each key we uniformly sample bins. Sampling a bin $s \in [b]$ has two possible outcomes:

- $c_s < k$ (*Success*): We increment the counter c_s by one and continue with the next key.
- $c_s \geq k$ (*Failure*): In this case, we also increment the counter c_s by one. Note that this does *not* change anything in terms of success probabilities, since the bin is already full anyway. We continue with sampling a new bin for the same key.

Determining $p_k(x)$ Numerically. The reason for incrementing the counter even if the bin is full is that the counter of each bin now follows a Poisson distribution (assuming that the number of bins b is large). We show how $p_k(x)$ can be expressed as the solution of an integro-differential equation in the full version of this paper [18]. However, solving this equation is numerically unstable. In the following we give a numerically stable solution for $p_k(x)$. We repeatedly choose values μ. We use each μ as the expected value of the Poisson distribution that describes the distribution of the counters. For each μ we calculate the probability p that a key is inserted successfully, as well as the fraction of inserted keys x. Hence, for each μ we obtain p and x which we use to numerically determine $p_k(x)$. We choose the values μ such that the distance of consecutive x values is sufficiently small. The success probability p for a given μ is the probability that $c_s < k$, which we calculate using the cumulative distribution function of the Poisson distribution. To determine the fraction of inserted keys x for a given expected value μ we consider the counter of the bins as described in the model. For a bin where the counter is higher than k, we charge k successfully inserted keys, because all other incrementations are from failures. We then divide by k, since we are interested in the relative number of inserted keys. Let X follow a Poisson distribution with mean μ then $x = \frac{1}{k}(\mathbb{E}[X \mid X \leq k] + k\mathbb{P}[X > k]) = \frac{1}{k}\mathbb{E}[X \mid X \leq k] + \mathbb{P}[X > k]$.

The Bucket Assignment Function. The final bucket assignment function can be obtained by numerically evaluating the integral of Equation (3) for $\beta_k(x)$. The constant C_k can be ignored by normalizing the integral as a last step such that the boundary condition $\beta_k(1) = 1$ is met. Refer to Figure 2b for an illustration of the bucket assignment functions with different values of k. Larger k make the function more aggressive, further increasing the size of the first buckets. For example, for $k = 100$, 82% of the keys are hashed to 1% of buckets. Our implementation tabulates the bucket assignment function for specific k and uses linear interpolation to maintain a fast query time.

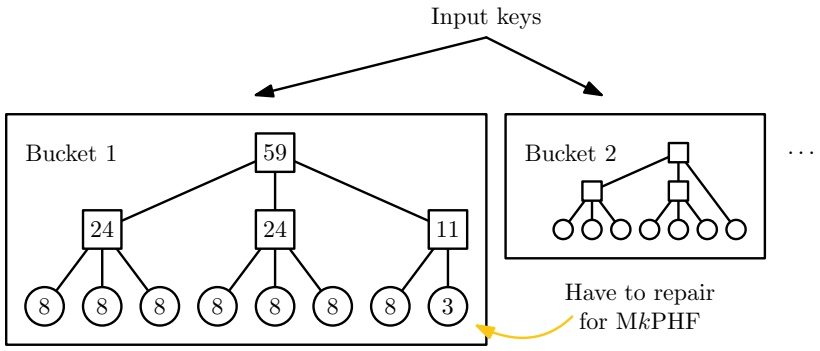


Figure 3 Illustration of RecSplit with $k = 8$ and lowest-layer fanout $\ell = 3$.

6 k-Perfect RecSplit

RecSplit [9] is a very space-efficient 1-perfect hash function. We first explain RecSplit and then show how we adapt it to a k-perfect hash function.

RecSplit. RecSplit [9] first hashes each key to a bucket of expected size b, where b is a tuning parameter with default value $b = 2000$ in the implementation. Within each bucket, RecSplit then uses brute-force to search for a hash function seed that divides the keys into two (or more) subsets. The number of these subsets (*fanout*) depends on the *splitting strategy*. This continues recursively until we are left with small input sets of $\leq \ell$ keys in the leaf nodes. These subsets are small enough that it is feasible to search for a perfect hash function using brute-force. RecSplit stores the hash function seeds determined at each node using Golomb-Rice coding (see Section 2) in DFS order. An Elias-Fano coded sequence (see Section 2) stores where the encoding of each bucket starts, as well as the total number of keys before it. A query traverses the splitting tree of the corresponding bucket, evaluating the hash function in each node. When descending into a subtree, it sums up the number of keys in sibling nodes left to it. Therefore, the final hash function value is given by the number of keys before the bucket, the number of keys to the left in the tree, and the final brute-force hash function value in the leaf. The RecSplit splitting strategy ensures that all subtrees except possibly the last are a complete tree where each leaf node receives exactly ℓ keys. Therefore, only the very last leaf in any tree can have a size smaller than ℓ. We illustrate RecSplit in Figure 3. Through the use of the buckets, RecSplit achieves linear time construction and constant time queries.

Main Idea. Our k-perfect adaptation of RecSplit is (almost) straight-forward. We perform splittings until we are left with leaves of size k. These then do not need to store seeds. We now use the parameter ℓ to select the fanout of the layer above. The main difficulty is that the last leaf in each bucket can have size $< k$. This would make the result a non-minimal kPHF. In Section 6.1, we explain how we can merge the leaves of adjacent buckets in order to get output bins of size k. As an alternative, in Section 6.2, we explain *nested* k-perfect hashing. This can avoid the problem by ensuring that each bucket's size is a multiple of k.

6.1 Merging Leaves of Size $< k$

From the prefix sum of bucket sizes, we know the number of keys x before each bucket. We start placing the output of the bucket's tree at the offset $\lfloor x/k \rfloor$. This means that we do not leave space for the previous bucket's leaves of size $< k$. Only if the total number of keys in

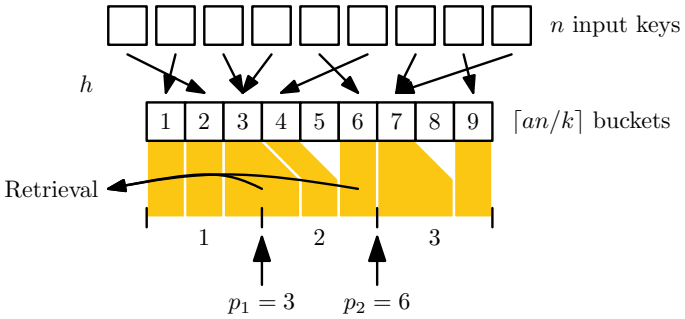


Figure 4 Illustration of PaCHash for k-perfect hashing with $a = k = 3$. Queries for keys hashed to bucket 3 cannot directly determine whether their output bin is 1 or 2 by looking at the cut-points p_i. Therefore, they have to query a retrieval data structure. For the single key in bin 6, the placement would actually be unambiguous. However, from looking at p_2, we cannot see that all keys in bin 6 have to map to the left of the threshold. Therefore, we still need to query the retrieval data structure. For p_2, can store either 6 or 7. We select the bin with fewer keys to reduce the number of keys in the retrieval data structure.

non-full leaves exceeds k, we get an additional output bin. For this, we calculate a splitting hash function on the last leaf that splits away the keys needed to fill the leaf. We store the splitting seed as an additional hash function seed in the Golomb-Rice coded seeds.

During queries, if we arrive in a leaf of size $< k$, there are two cases. (1) If the remainder of the bucket position does not cross k, we scan forward the following buckets until one does. The output bin is then the last bin of that bucket. (2) If the current bucket is the one letting the remainder cross k, we evaluate the splitting hash function. Then we either return the last bin of the current bucket, or we scan forward like in the first case.

Note that this means that we have to scan forward the bucket sizes when querying a leaf of size $< k$. However, because the number of keys in a bucket is random and much larger than k, we can assume we get a uniform random remainder. Therefore, in expectation, we only have to scan two buckets.

6.2 Avoiding Leaves of Size $< k$ Through Nesting

Another technique is to completely avoid leaves of size $< k$ in the first place. We can use a minimal b-perfect hash function (where k divides b) to partition the keys. In general, we can nest different k-perfect hash functions to interpolate their trade-offs. One function (maybe fast and less space-efficient) can map the input set to buckets of k' keys. Then, in each bucket, we can construct a k-perfect hash function. We could use the idea to implement a k-perfect version of CONSENSUS-RecSplit. However, nesting different kPHFs would introduce another dimension of experiments and make this paper harder to follow. We therefore do not go into detail about nesting or k-perfect CONSENSUS-RecSplit.

7 k-Perfect PaCHash

As we explain in Section 3.2, PaCHash hashes each key to one of $\lceil an/k \rceil$ buckets. The variable a is a tuning parameter. For each output bin i, it stores the first bucket p_i overlapping it using Elias-Fano coding (see Section 2). A predecessor query on that list then maps the bucket to the respective output bin. Figure 4 illustrates the data structure. During queries, if a searched bucket is contained in the sequence, it is not clear whether the key is supposed

Listing 3 Query algorithm of k-perfect PaCHash. If the range returned by the internal memory data structure is ambiguous, we have to query the retrieval data structure. Hash function h_X uniformly maps into the set X.

```
Function query(key ∈ S)
    b := h_{⌈an/k⌉}(x)
    find i such that p_{i-1} < b ≤ p_i                    // predecessor query
    if p_i = b
      i := i − 1                                          // b may start in previous block
    find first j such that p_j > b                        // predecessor query or scan
    if i = (j − 1)
      return i
    else
      return i + retrievalQuery(key)                      // (j-1-i)-bit retrieval
```

to be in the bin or in the bin before. In fact, it might be possible that d bins store the same bucket index. We get a similar problem if a bucket starts exactly at a bin boundary. Then the index cannot tell whether the bucket overlaps or not. The external memory implementation then loads all $d + 1$ candidate bins. However, for k-perfect hashing, we need a different strategy.

***k*-Perfect Adaptation.** To decide for one of the $d + 1$ bins using only the index data structure, we use a $\lceil \log_2(d+1) \rceil$ bit retrieval data structure (see Section 2). The stored value tells us which of the bins to return. Since we rarely need more than one bit, we simply store the numbers bitwise in a single 1-bit retrieval data structure. Listing 3 gives the pseudocode of the query algorithm. PaCHash expects $1/a$ of the queries to load one external memory page too much [23]. In the context of k-perfect hashing, this means that for $1/a$ of the queries, the output bin is ambiguous. This means that the retrieval data structure essentially stores n/a bits. By selecting $a = k$, we get a total space consumption of $n/k(3 + \log(k))$, which already gets close to the lower bound for large k (see Section 2). An advantage of the technique is that its construction through (integer) sorting and a single scan is very simple. Like the other approaches it takes linear time. Queries take expected constant time due to the distribution of values in the Elias-Fano data structure [23]. This adaptation is briefly mentioned in previous papers [27, 31] but without an implementation. The CONSENSUS paper [31] gives a simple analysis for the resulting space consumption.

8 Evaluation

In this section, we compare our approaches with CHD [2], as well as the ad-hoc implementation of threshold-based bumping [31] from the literature. While CHD is not *minimal* k-perfect, we still include it in our comparison with a load factor of 97% to give a feeling for the performance of our approaches. With this paper, we build a foundation for future work on k-perfect hashing, contributing implementations for future papers to compare against.

We run our experiments on an Intel i7 11700 processor with 8 cores and a base clock speed of 2.5 GHz. The machine runs Rocky Linux 9.5 with Linux 5.14.0. We use the GNU C++ compiler version 11.2.0 with optimization flags `-O3 -march=native`. Following the practice in 1-perfect hashing [20, 27, 28], our input set consists of 100 million random strings of uniform random length $\in [10, 50]$. Because all approaches hash the input keys anyway, the choice of the input set is not very important. Our source code is public under the General Public License [19].

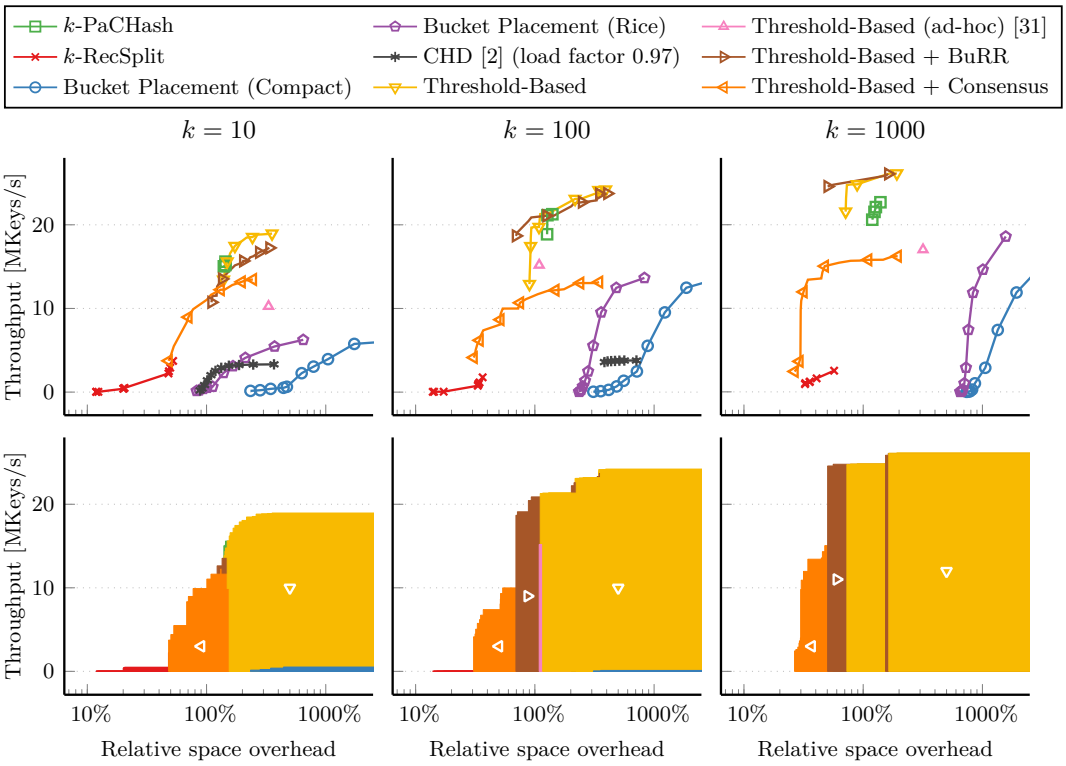
Because each approach has configuration parameters, it can cover a range of trade-offs between space consumption, construction time, and query time. To give an intuitive overview, we give visual plots showing different parameters of each approach. Figure 5a gives the trade-off between construction time and space consumption for three different values of k. Figure 5b gives the trade-off between query time and space consumption. We also give a *dominance map* [27, 7, 28] of the trade-off. For each point having a specific trade-off between construction time and space consumption, the dominance map in Figure 5a shows the approach achieving the best query time. As such, it can be seen as a "front view" of the 3-dimensional Pareto space. The dominance map in Figure 5b shows the method with the fastest construction for a given trade-off between space consumption and query performance. We show a selection of representative configurations from the Pareto front in the full version of this paper [18]. In the following, we discuss the different approaches in detail.

Threshold-Based Bumping. Compared to the ad-hoc implementation [31], our implementation of threshold-based bumping offers up to 3 times smaller space overhead due to the optimized thresholds. Only for $k = 100$, the linearly spaced ad-hoc thresholds happen to be close to optimal, so we only achieve small space improvements. For growing k, threshold-based bumping gets closer to the space lower bound, while the space overhead of most other approaches increases. Figure 5a also shows that our implementation is up to 50% faster to construct than the next best technique. This is mainly because the ad-hoc implementation fully sorts the keys by their bin and fingerprint, while we only partition by the bin and then use quickselect [21] to find the k-th key. By packing using retrieval, we can achieve smaller space consumption. This has only a small influence on construction and query times. Finally, using CONSENSUS-coded thresholds, we achieve smaller space consumption but slower construction and queries.

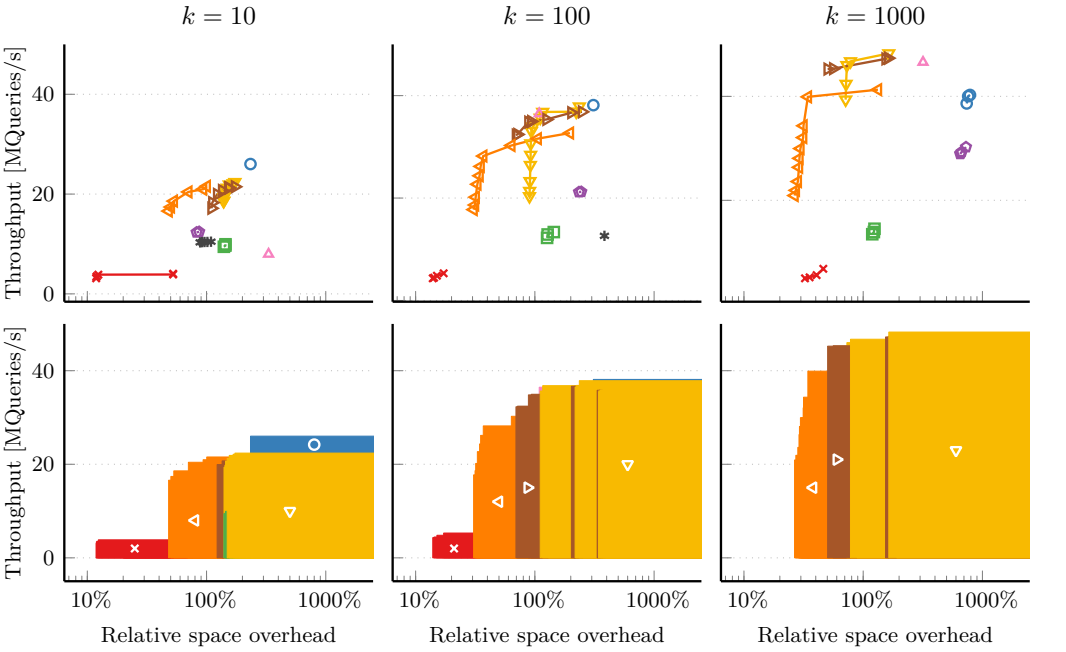
Hash and Displace. For the hash and displace technique, Rice coding (see Section 2) achieves lower space consumption than compact coding at the same construction time. This comes at the cost of slower queries, as can be seen in Figure 5b, due to the select queries. Compared to CHD [2], an implementation from the literature, our construction is much faster. Even though we use CHD with a load factor of just 97%, our construction gets close to its space consumption. The CHD implementation from the literature crashes for $k = 1000$, while our implementation supports large k as well. Our measurements show that the hash and displace technique with compact coding is fast to query, especially for smaller k. This is in contrast to the implementation from the literature, which has much slower queries. In general, the hash and displace technique seems not competitive for k-perfect hashing, especially for large k.

***k*-RecSplit.** Our k-perfect adaptation of RecSplit achieves the smallest space consumption. With just 10% overhead over the lower bound, it has a large margin to the next smallest competitor. This is not surprising, since the RecSplit paper [9] shows that the recursive splitting idea converges towards the space lower bound. However, its construction is significantly slower than the other approaches in Figure 5a. Because splittings become even more expensive for large k, we do not include the configuration with 10% space overhead for $k = 1000$. The queries are slow compared to other approaches.

***k*-PaCHash.** Our k-perfect adaptation of PaCHash is very simple to describe and fast to construct through sorting followed by a linear scan. However, its queries are slow due to the predecessor queries in the Elias-Fano coded sequence of bucket indexes.



(a) Space consumption versus construction time. The bottom row shows dominance maps indicating the approach with the fastest queries, given a specific trade-off between space and construction time.



(b) Space consumption versus query time. The bottom row shows dominance maps indicating the approach with the fastest construction, given a specific trade-off between space and query time.

Figure 5 Comparison of minimal k-perfect hash functions. A relative space overhead of 100% means that the data structure needs twice the lower bound. Refer to Table 1 for a table with lower bounds in bits per key.

9 Conclusion

To our knowledge, we present the first paper fully dedicated to presenting k-perfect hash functions. The area of k-perfect hashing has been known for a long time, but has not received a lot of attention. With the advent of very space-efficient 1-perfect hash functions, however, the area is becoming more and more relevant. In this paper, we kickstart the development of kPHFs by introducing four new or significantly improved constructions. Our constructions cover various trade-offs between space consumption, construction time, and query time.

With threshold-based bumping, we significantly improve a k-perfect hash function that was previously only described briefly as an ad-hoc solution in 1-perfect hashing papers [30, 31]. We achieve a significant reduction in bumped keys by combining it with retrieval, and we optimize the threshold function. By combining the idea with CONSENSUS coded seeds, we achieve a significant reduction in space consumption while retaining its fast queries. We also extend the idea of perfect hashing through bucket placement by optimizing the bucket assignment function for k-perfect hashing. As a very space-efficient alternative, we adapt RecSplit [9] to construct k-perfect hash functions. The main difficulty is to merge leaf nodes that have size $< k$. Finally, we adapt PaCHash [23] to a k-perfect hash function by combining it with a retrieval data structure.

Our extensive evaluation compares our new techniques with competitors from the literature. Our constructions are simultaneously faster to construct, faster to query, and more space-efficient than previous approaches. The most promising techniques are threshold-based bumping and k-RecSplit. While k-RecSplit achieves the best space consumption, it is slow to construct and query. Threshold-based bumping is very fast to construct and query while still giving competitive space consumption.

Future Work. This paper revives the research area of k-perfect hashing. In the future, we expect a wide range of new techniques, like there is in 1-perfect hashing. The approaches presented in this paper give a starting point outlining the trade-off. Especially our improvements to threshold-based bumping can now be used to improve the space efficiency of CONSENSUS-RecSplit [31]. An open theoretical problem is the space lower bound for non-minimal k-perfect hashing.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Djamal Belazzougui, Fabiano C. Botelho, and Martin Dietzfelbinger. Hash, displace, and compress. In *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 682–693. Springer, 2009. doi:10.1007/978-3-642-04128-0_61.

3 Piotr Beling. Fingerprinting-based minimal perfect hashing revisited. *ACM J. Exp. Algorithmics*, 28:1.4:1–1.4:16, 2023. doi:10.1145/3596453.

4 Francine Berman, Mary Ellen Bock, Eric Dittert, Michael J. O'Donnell, and Darrell Plank. Collections of functions for perfect hashing. *SIAM J. Comput.*, 15(2):604–618, 1986. doi:10.1137/0215044.

5 Pedro Celia. External robin hood hashing. Technical report, Computer Science Department, Indiana University. TR246, 1988.

6 Davi de Castro Reis, Djamel Belazzougui, Fabiano Cupertino Botelho, and Nivio Ziviani. CMPH – C minimal perfect hashing library. http://cmph.sourceforge.net/, 2012.

7 Peter C. Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer. Fast succinct retrieval and approximate membership using ribbon. In *SEA*, volume 233 of *LIPIcs*, pages 4:1–4:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SEA.2022.4.

8 Peter Elias. Efficient storage and retrieval by content and address of static files. *J. ACM*, 21(2):246–260, 1974. doi:10.1145/321812.321820.

9 Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. RecSplit: Minimal perfect hashing via recursive splitting. In *ALENEX*, pages 175–185. SIAM, 2020. doi:10.1137/1.9781611976007.14.

10 Ronald Fagin, Jürg Nievergelt, Nicholas Pippenger, and H. Raymond Strong. Extendible hashing - A fast access method for dynamic files. *ACM Trans. Database Syst.*, 4(3):315–344, 1979. doi:10.1145/320083.320092.

11 Robert Mario Fano. On the number of bits required to implement an associative memory. Technical report, MIT, Computer Structures Group, 1971. Project MAC, Memorandum 61".

12 Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst.*, 38(2):229–248, 2005. doi:10.1007/S00224-004-1195-X.

13 Fabian Frei and David Wehner. Bounds for c-ideal hashing. In *FCT*, volume 14292 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 2023. doi:10.1007/978-3-031-43587-4_15.

14 Solomon W. Golomb. Run-length encodings. *IEEE Trans. Inf. Theory*, 12(3):399–401, 1966. doi:10.1109/TIT.1966.1053907.

15 Gaston H. Gonnet and Per-Åke Larson. External hashing with limited internal storage. *J. ACM*, 35(1):161–184, 1988. doi:10.1145/42267.42274.

16 Ragnar Groot Koerkamp. Ptrhash: Minimal perfect hashing at RAM throughput. *CoRR*, abs/2502.15539, 2025. doi:10.48550/arXiv.2502.15539.

17 Stefan Hermann. MorphisHash: Improving space efficiency of shockhash for minimal perfect hashing. *CoRR*, abs/2503.10161, 2025. doi:10.48550/arXiv.2503.10161.

18 Stefan Hermann, Sebastian Kirmayer, Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. Engineering minimal k-perfect hash functions. *CoRR*, abs/2504.20001, 2025. doi:10.48550/arXiv.2504.20001.

19 Stefan Hermann, Sebastian Kirmayer, Hans-Peter Lehmann, and Stefan Walzer. Engineering Minimal k-Perfect Hash Functions. Software, swhId: swh:1:dir:e6756f018691a80adc68d839fd3617d8f5e2d0b0 (visited on 2025-09-08). URL: https://github.com/stefanfred/engineering-k-perfect-hashing, doi:10.4230/artifacts.24698.

20 Stefan Hermann, Hans-Peter Lehmann, Giulio Ermanno Pibiri, Peter Sanders, and Stefan Walzer. PHOBIC: perfect hashing with optimized bucket sizes and interleaved coding. In *ESA*, volume 308 of *LIPIcs*, pages 69:1–69:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.69.

21 C. A. R. Hoare. Algorithm 65: find. *Commun. ACM*, 4(7):321–322, July 1961. doi:10.1145/366622.366647.

22 Sebastian Kirmayer. Engineering k-perfect hashing. Abschlussarbeit - Bachelor, Karlsruher Institut für Technologie (KIT), 2024. doi:10.5445/IR/1000179026.

23 Florian Kurpicz, Hans-Peter Lehmann, and Peter Sanders. PaCHash: Packed and compressed hash tables. In *ALENEX*, pages 162–175. SIAM, 2023. doi:10.1137/1.9781611977561.CH14.

24 Per-Åke Larson. Linear hashing with separators - A dynamic hashing scheme achieving one-access retrieval. *ACM Trans. Database Syst.*, 13(3):366–388, 1988. doi:10.1145/44498.44500.

25 Per-Åke Larson and Ajay Kajla. File organization: Implementation of a method guaranteeing retrieval in one access. *Commun. ACM*, 27(7):670–677, 1984. doi:10.1145/358105.358193.

26 Per-Åke Larson and M. V. Ramakrishna. External perfect hashing. In *SIGMOD Conference*, pages 190–200. ACM Press, 1985. doi:10.1145/318898.318916.

27 Hans-Peter Lehmann. *Fast and Space-Efficient Perfect Hashing*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2024. doi:10.5445/IR/1000176432.

28 Hans-Peter Lehmann, Thomas Mueller, Rasmus Pagh, Giulio Ermanno Pibiri, Peter Sanders, Sebastiano Vigna, and Stefan Walzer. Modern minimal perfect hashing: A survey. *CoRR*, abs/2506.06536, 2025. doi:10.48550/arXiv.2506.06536.

29 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. SicHash – small irregular cuckoo tables for perfect hashing. In *ALENEX*, pages 176–189. SIAM, 2023. doi:10.1137/1.9781611977561.CH15.

30 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. ShockHash: Near optimal-space minimal perfect hashing beyond brute-force. *CoRR*, abs/2310.14959, 2024. doi:10.48550/arXiv.2310.14959.

31 Hans-Peter Lehmann, Peter Sanders, Stefan Walzer, and Jonatan Ziegler. Combined search and encoding for seeds, with an application to minimal perfect hashing. In *33nd Annual European Symposium on Algorithms (ESA 2025)*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.110.

32 Harry G. Mairson. The program complexity of searching a table. In *FOCS*, pages 40–47. IEEE Computer Society, 1983. doi:10.1109/SFCS.1983.76.

33 Harry G Mairson. The effect of table expansion on the program complexity of perfect hash functions. *BIT Numerical Mathematics*, 32:430–440, 1992. doi:10.1007/BF02074879.

34 Kurt Mehlhorn. Data structures and algorithms, vol. 1: Sorting and searching. *EATCS Monographs on Theoretical Computer Science, Springer-Verlag*, 1984.

35 Ingo Müller, Peter Sanders, Robert Schulze, and Wei Zhou. Retrieval and perfect hashing using fingerprinting. In *SEA*, volume 8504 of *Lecture Notes in Computer Science*, pages 138–149. Springer, 2014. doi:10.1007/978-3-319-07959-2_12.

36 Rasmus Pagh. Basic external memory data structures. In *Algorithms for Memory Hierarchies*, volume 2625 of *Lecture Notes in Computer Science*, pages 14–35. Springer, 2002. doi:10.1007/3-540-36574-5_2.

37 Giulio Ermanno Pibiri and Roberto Trani. PTHash: Revisiting FCH minimal perfect hashing. In *SIGIR*, pages 1339–1348. ACM, 2021. doi:10.1145/3404835.3462849.

38 Robert F. Rice. Some practical universal noiseless coding techniques. *Jet Propulsion Laboratory, JPL Publication*, 1979.

Cut-Query Algorithms with Few Rounds

Yotam Kenneth-Mordoch
Weizmann Institute of Science, Rehovot, Israel

Robert Krauthgamer
The Harry Weinrebe Professorial Chair of Computer Science, Weizmann Institute of Science, Rehovot, Israel

Abstract

In the cut-query model, the algorithm can access the input graph $G = (V, E)$ only via cut queries that report, given a set $S \subseteq V$, the total weight of edges crossing the cut between S and $V \setminus S$. This model was introduced by Rubinstein, Schramm and Weinberg [ITCS'18] and its investigation has so far focused on the number of queries needed to solve optimization problems, such as global minimum cut. We turn attention to the round complexity of cut-query algorithms, and show that several classical problems can be solved in this model with only a constant number of rounds.

Our main results are algorithms for finding a minimum cut in a graph, that offer different tradeoffs between round complexity and query complexity, where $n = |V|$ and $\delta(G)$ denotes the minimum degree of G: (i) $\tilde{O}(n^{4/3})$ cut queries in two rounds in unweighted graphs; (ii) $\tilde{O}(rn^{1+1/r}/\delta(G)^{1/r})$ queries in $2r+1$ rounds for any integer $r \geq 1$ again in unweighted graphs; and (iii) $\tilde{O}(rn^{1+(1+\log_n W)/r})$ queries in $4r+3$ rounds for any $r \geq 1$ in weighted graphs. We also provide algorithms that find a minimum (s,t)-cut and approximate the maximum cut in a few rounds.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Submodular optimization and polymatroids

Keywords and phrases Cut Queries, Round Complexity, Submodular Optimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.100

Related Version *A full version is available at*: `arXiv:2506.20412`

Funding *Robert Krauthgamer*: Work partially supported by the Israel Science Foundation grant #1336/23, by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center, and by a research grant from the Estate of Harry Schutzman.

1 Introduction

Graph cuts are a fundamental object in computer science with numerous applications, in part because they are a basic example of submodular functions. Recall that a set function $f : 2^V \to \mathbb{R}$ is *submodular* if it satisfies the diminishing-returns property

$$\forall A \subset B \subset V, \forall v \notin B, \quad f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B).$$

From the viewpoint of submodular optimization, it is natural to study the complexity of graph algorithms in the *cut-query model*, which correspond to value queries to the submodular function. Here, the input graph $G = (V, E, w)$ can be accessed only via oracle queries to its cut function, namely to $\mathrm{cut}_G : 2^V \to \mathbb{R}$, given by

$$\forall S \subseteq V, \qquad \mathrm{cut}_G(S) := \sum_{e \in E:\ |e \cap S| = 1} w(e).$$

Here and throughout $w(e) > 0$ is the weight of the edge $e \in E$, precluding edges of zero weight as they cannot be detected by the algorithm. As usual, an unweighted graph models unit weights, i.e., $w(e) = 1$ for all e.


© Yotam Kenneth-Mordoch and Robert Krauthgamer;
licensed under Creative Commons License CC-BY 4.0



The *query complexity* of an algorithm is the number of queries it makes in the worst-case. While this is often the primary performance measure, it is highly desirable that algorithms can be parallelized, as such algorithms can utilize better the available computing resources. The literature on the parallelization of submodular optimization [9, 24, 25, 18], relies on a measure called *round complexity*, which count the number of sequential rounds of queries the algorithm makes.[1]

▶ **Definition 1.1** (Round Complexity). *An algorithm has* round complexity r *if it performs r rounds of queries, meaning that queries in round $t \in [r]$ can depend only on the answers to queries in previous rounds $1, \ldots, t-1$. An algorithm is called* non-adaptive *if it uses a single round of queries.*

We study the cut-query model, from the perspective of round complexity focusing on the problem of finding a global minimum cut. Previous work [33, 31, 3, 2] shows that a minimum cut of a graph can be found using $O(n)$ queries in unweighted graphs and $\tilde{O}(n)$ queries in weighted graphs. However, it paid no attention to the algorithms' round complexity, and we fill this gap by studying the tradeoff between round complexity and query complexity.

An analysis of the round complexity of existing algorithms, with some simple modifications, shows that it is possible to find a global minimum cut of a graph using $\tilde{O}(n)$ queries with round complexity $\tilde{O}(\log^2 n)$ [33, 31]. It should be noted that other algorithms do require round complexity $\Omega(n)$ [3, 2]. As another baseline, it is possible to recover the entire graph using $O(n^2)$ deterministic non-adaptive queries [14],[2] which clearly suffices to find a minimum cut. This motivates our main question

▶ **Question 1.2.** *What is the tradeoff between the round complexity and the query complexity of finding a global minimum cut in the cut-query model?*

1.1 Main Results

We show that the polylogarithmic round complexity of previous work is not necessary for finding a global minimum cut. We begin by presenting two results for unweighted graphs. The first one shows that using even two rounds, we can substantially improve on the $O(n^2)$ query complexity of the naive algorithm. All of our algorithms return a vertex set $S \subseteq V$ and the value of the minimum cut. In the case of unweighted graphs the algorithms can also report the edges of the cut using one additional round of queries and with the same asymptotic query complexity.

▶ **Theorem 1.3** (Unweighted Minimum Cut with 2 rounds). *Given an unweighted graph G on n vertices, it is possible to find a minimum cut of G using $\tilde{O}(n^{4/3})$ cut queries in 2 rounds. The algorithm is randomized and succeeds with probability $1 - n^{-2}$.*

We further provide a smooth tradeoff between the number of queries and the number of rounds, namely $\tilde{O}(n^{1+1/r}/\delta(G)^{1/r})$ cut queries with in $2r$ rounds, where $\delta(G) \geq 1$ is the minimum degree of G.[3]

[1] The aforementioned references use the term *adaptivity*, which is the number of rounds minus one.

[2] $\tilde{O}(|E|)$ queries suffice, however without some bound on the size of the graph the algorithm must make $O(n^2)$ queries in the worst case.

[3] We can assume that the graph is connected since connectivity can be checked using $\tilde{O}(n)$ cut queries in 1 round (non-adaptive) [4].

▶ **Theorem 1.4** (Unweighted Minimum Cut with $O(r)$ rounds). *Given an unweighted graph G on n vertices and a parameter $r \in \{1, 2, \ldots, \log n\}$, it is possible to find a minimum cut of G using $\tilde{O}(rn^{1+1/r}\delta(G)^{-1/r})$ cut queries in $2r+1$ rounds. The algorithm is randomized and succeeds with probability $1 - n^{-1}$.*

We also obtain a similar tradeoff also for weighted graphs.

▶ **Theorem 1.5** (Weighted Minimum Cut Graphs with $O(r)$ rounds). *Given a weighted graph G on n vertices with integer edge weights bounded by W, and a parameter $r \in \{1, 2, \ldots, \log n\}$, it is possible to find a minimum cut of G using $\tilde{O}(rn^{1+(1+\log_n W)/r})$ cut queries in $4r+3$ rounds. The algorithm is randomized and succeeds with probability $1 - n^{-2}$.*

Finally, we note that our techniques can also be applied to approximating the maximum cut and finding a minimum (s,t)-cut with low adaptivity. Furthermore, our result for weighted minimum cuts can also be applied to finding a minimum cut in dynamic streams, providing a smooth tradeoff between the number of passes and the storage complexity of the algorithm. This holds since each cut query in a given round can be computed using $O(\log n)$ storage in a single pass by simply counting the edges in the cut. Therefore, the round complexity of our algorithm translates immediately to the number of passes required by a streaming algorithm. These results are detailed in the full version of the paper.

1.2 Related Work

Algorithms in the cut query model. In recent years there have been several works on cut problems in the cut-query model, most of which focused on query complexity and not round complexity. We will discuss the known results for several problems in the cut-query model.

Beginning with finding a minimum cut, $O(n)$ randomized queries suffice for simple graphs [33, 3]. Meanwhile, finding a minimum cut in a weighted graph requires $\tilde{O}(n)$ randomized queries [31]. Finally, in a recent work it was shown that non-trivial deterministic algorithms exist for finding the minimum cut of a simple graph using $\tilde{O}(n^{5/3})$ queries [2]. For the easier problem of determining whether a graph is connected, it is known that $O(n)$ randomized queries suffice [19, 30]. Additionally, there exists a one round randomized algorithm for connectivity that uses $\tilde{O}(n)$ queries [4].

Moving to the problem of finding a minimum (s,t)-cut. In [33] it was shown that it is possible to compute the minimum (s,t)-cut of a simple graph using $O(n^{5/3})$ queries, by showing that using a cut sparsifier it is possible to find a small set of edges that contains the minimum (s,t)-cut. This query complexity was matched in the deterministic setting by [2].

We also mention the problem of finding a $1-\epsilon$ approximation of the maximum cut. One way to achieve this is by constructing a cut sparsifier, this was show in [33] for simple graphs and in [32] for weighted graphs, both using $\tilde{O}(\epsilon^{-2}n)$ cut queries. Furthermore, an algorithm for 1/2-approximation using $O(\log n)$ cut queries in one round and a lower bound of $\tilde{\Omega}(n)$ on the query complexity needed to find a $1-\epsilon$ approximation of the maximum cut are also known [32].

Finally, there have been several works on graph problems in different query models, such as additive queries [22, 14, 15], quantum cut queries [29, 6, 3], matrix multiplication queries [34, 6, 3], Bipartite Independent Set queries [11, 4, 1] and more.

Adaptivity in submodular optimization. In recent years, there has been a growing interest in the adaptivity of algorithms for submodular optimization. This line of work was initiated by the work of [9], which showed that constrained submodular maximization can be solved

within a constant approximation factor in $O(\log n)$ rounds with a polynomial number of queries, the work also provides a matching lower bound. A string of later works improved on the query complexity and the approximation guarantee, culminating in an algorithm with an optimal $1 - 1/e$ approximation using $\tilde{O}(n)$ queries [9, 7, 8, 24, 25, 21]. The round complexity of unconstrained submodular maximization was also studied in [20].

The adaptivity of algorithms for submodular minimization was also studied in several works. It was shown that solving the minimum (s, t)-cut problem in weighted graphs using $O(\log n)$ rounds of queries requires $\tilde{\Omega}(n^2)$ queries [5]. For more general submodular functions, a bound of $\Omega(n/\log n)$ rounds for algorithms with a polynomial number of queries is known [16, 10, 17]. Note that all these hard instances are non symmetric, and hence do not apply to the minimum cut problem. It remains open to study the round complexity of symmetric submodular minimization problems in future work.

2 Technical Overview

Each of our three algorithms for minimum cut is based on a different insight. Our two-round algorithm uses contractions in the style of [28] to create a smaller graph that preserves the minimum cut. Similar contractions have been used in several minimum cut algorithm [28, 26, 3], however the existing algorithms aim to reduce the number of vertices in the graph, while our variant, called τ-star contraction, reduces the number of edges.

Our $(2r)$-round algorithm for unweighted graphs employs a known approach of applying a contraction procedure and then packing (in the contracted graph) edge-disjoint forests. The main challenge to ensure that forests are edge disjoint when packing them in parallel, and we overcome it by leveraging an elegant connection to cut sparsification, which shows that amplifying the sampling probabilities of the sparsifier construction by a factor of k ensures finding k edge-disjoint forests.

Our algorithm for weighted minimum cut follows the approach of [31], which constructs a $(1 \pm \epsilon)$-cut-sparsifier and then solves a monotone-matrix problem that is easily parallelizable. Prior work in the cut-query model provided cut sparsifiers [33, 32], but their core step of weight-proportional edge sampling requires $O(\log n)$ rounds. We eliminate this bottleneck with our two-round weight-proportional edge sampling primitive (Lemma 2.2), and thereby build a $(1 \pm \epsilon)$-cut-sparsifier in $3r$ rounds, and complete the entire algorithm in $O(r)$ rounds.

A common thread to all our algorithms is the use of edge sampling, both directly as part of the algorithms and to construct complex primitives such as forest packings and cut sparsifiers. In edge sampling, the input is a source vertex s and a target vertex set $T \subseteq V \setminus \{s\}$, and the goal is to sample an edge from $E(s, T) \coloneqq E \cap (\{s\} \times T)$. We need two types of edge sampling, both returning a sampled edge $e \in E(s, T)$ and its weight $w(e)$. The first one is *uniform edge sampling*, which picks each $e \in E(s, T)$ with equal probability $1/|E(v, T)|$ regardless of its weight, similarly to l_0 sampling; the second one is *weight-proportional edge sampling*, which picks each $e \in E(s, T)$ with probability $w(e)/w(E(v, T))$, similarly to l_1 sampling. In unweighted graphs these two definitions clearly coincide.

Weight-proportional edge sampling was used extensively in previous algorithms for the cut-query model [33, 31, 3, 32], although it had a naive implementation that requires $O(\log n)$ rounds. This naive implementation, which was proposed in [33], follows a binary search approach. Each step partitions the set T at random into two sets T_1, T_2 of equal cardinality, and finds the total weight of the edges in $E(s, T_1)$ and $E(s, T_2)$. The algorithm then decides whether to continue with T_1 or T_2 randomly in proportion to their weights. This approach clearly requires $\Omega(\log n)$ rounds, which exceeds the $O(1)$ round complexity we aim for, and furthermore yields only weight-proportional sampling, while some of our algorithms require uniform sampling in weighted graphs.

2.1 Edge Sampling Primitives

Our first technical contribution is a procedure for non-adaptive uniform edge sampling using $O(\log^3 n/\log\log n)$ queries that returns the weight of the sampled edge. An algorithm for uniform edge sampling using $O(\log^3 n)$ cut queries in one round (i.e. non-adaptively) was proposed in [4].[4] However, their algorithm is based on group testing principles using BIS queries[5] and returns only the endpoints of the edge, but not its weight, though it is probably not difficult to adapt their algorithm to return the weight as well. Our algorithm slightly improves on the query complexity of their algorithm, while also returning the edge's weight in the same round.

▶ **Lemma 2.1** (Uniform Edge Sampling). *Given a (possibly weighted) graph G on n vertices, a source vertex $s \in V$, and a target vertex set $T \subseteq V \setminus \{s\}$, one can return an edge e uniformly sampled from $E(s,T)$, along with its weight $w(e)$, using $O(\log^3 n/\log\log n)$ cut queries in one round, i.e, non-adaptively. The algorithm succeeds with probability $1-n^{-4}$.*

The algorithm works by subsampling the vertices in T into sets $T = T_0 \supseteq T_1 \supseteq \ldots \supseteq T_{\log n}$, and finds a level i such that $0 < |E(s,T_i)| \leq O(\log n)$. It then leverages sparse recovery techniques to recover all the edges in $E(s,T_i)$ and return one of them at random. To perform this algorithm in one round the algorithm recovers $w(E(s,T_i))$ and performs the queries needed for sparse recovery for all levels in parallel. It then applies the sparse recovery algorithm to the last level with non-zero number of edges, i.e. the last level with $w(E(s,T_i)) > 0$. We note that it is easy to decrease the query complexity of Lemma 2.1 by a factor of $O(\log n)$ by adding a round of queries and performing the sparse recovery only on the relevant level.

Our second procedure performs weight-proportional edge sampling in two rounds.

▶ **Lemma 2.2** (Weight-Proportional Edge Sampling). *Given a graph G on n vertices with integer edge weights bounded by W, a source vertex $s \in V$, and a target vertex set $T \subseteq V \setminus \{s\}$, one can return an edge e from $E(s,T)$, along with its weight $w(e)$, sampled with probability $w(e)/w(E(s,T))$ using $O(\log^2 n \log^2(nW))$ queries in two rounds. The algorithm succeeds with probability $1-n^{-4}$.*

The algorithm follows the same basic approach as the uniform edge sampling primitive, i.e. creating nestled subsets $T = T_0 \supseteq T_1, \supseteq \ldots \supseteq T_{\log(nW)}$ by subsampling. However, instead of recovering only the edges that were sampled into the last level with non-zero weight, the algorithm recovers in each level i all edges $e \in E(s,T_i)$ such that $w(e) \approx w(E(s,T))/2^i$. The recovery procedure is based on a Count-Min data structure [23], which we show can be implemented using few non-adaptive cut queries. The algorithm then uses another round of queries to learn the exact weights of all edges that were recovered in all the levels. It then concludes by sampling an edge from the distribution $\{p_e\}$, where for every edge e recovered in the i-th level, it sets $p_e \propto 2^i w(e)/w(E(s,T))$; with probability $1-\sum_e p_e$, the algorithm fails. Since the probability of being sampled into the i-th level is 2^{-i}, these probabilities yield the desired distribution over the recovered edges.

To prove that the algorithm returns an edge with high probability, we show that there is a constant probability that the outlined procedure returns some edge $e \in E(s,T)$, and hence repeating the algorithm $O(\log n)$ times yields with high probability an edge sampled from

[4] The algorithm requires $O(\log^2 n \log(1/\delta))$ cut queries and succeeds with probability $1-\delta$, setting $\delta = n^{-2}$ yields the bound.

[5] A BIS query is given two disjoint sets $A, B \subseteq V$ and returns true if there exists some edge connecting A, B and false otherwise.

$E(s,T)$ with the desired distribution. The query complexity of the algorithm is dominated by the Count-Min data structure, which requires $O(\log n \log(nW))$ cut queries in each level. Applying the recovery procedure on $O(\log(nW))$ levels, and repeating the procedure $O(\log n)$ times yields the stated query complexity.

2.2 Unweighted Minimum Cut in Two Rounds

The key idea of our algorithm is to apply an edge contraction technique that produces a graph with substantially fewer edges while preserving each minimum cut with constant probability. We then recover the entire contracted graph using a limited number of cut queries by applying existing graph recovery methods using $\tilde{O}(m)$ additive queries [22, 14]. We show that it is possible to simulate k additive queries using $O(n+k)$ cut queries, which allows us to recover a graph with $m \geq \tilde{O}(n)$ edges using $O(m)$ cut queries.

▶ **Corollary 2.3.** *Given a weighted graph G on n vertices and m edges, one can recover the graph G using $O(m+n)$ non-adaptive cut queries.*

While there has been frequent use of contraction procedures for finding minimum cuts, existing algorithms focused on reducing the number of vertices in the graph [28, 26, 3], typically to $\tilde{O}(n/\delta(G))$. This does not suffice to guarantee a graph with few edges, for example, applying such a contraction on a graph G with minimum degree $\delta(G) = \log^2 n$ and $\Omega(n)$ vertices of degree n, might yield a graph with $\tilde{\Omega}(n^2)$ edges. In contrast, our goal is to reduce the number of edges in the graph, allowing efficient recovery of the contracted graph using Corollary 2.3. The main technical contribution of this section is a new contraction algorithm, which we call τ-star contraction, that reduces the number of edges to $\tilde{O}((n/\tau)^2 + n\tau)$.

Our contraction algorithm is a thresholded version of the star-contraction algorithm introduced in [3]. The original version of the star-contraction algorithm of [3] can be roughly described as follows. Sample a subset of *center vertices* $R \subseteq V$ uniformly at random with probability $p = O(\log n/\delta(G))$. For every $v \in V \setminus R$, choose uniformly a vertex in $r \in N_G(v) \cap R$ and contract the edge (v,r), keeping parallel edges, which yields a contracted multigraph G'. The main guarantee of the algorithm is that if the minimum cut of G is non-trivial, i.e. not composed of a single vertex, then $\lambda(G) = \lambda(G')$ with constant probability, where $\lambda(G)$ denotes the value of a minimum cut in a graph. The following theorem states this formally.

▶ **Theorem 2.4** (Theorem 2.2 in [3])**.** *Let $G = (V,E)$ be an unweighted graph on n vertices with a non-trivial minimum-cut value $\lambda(G)$. Then, the star-contraction algorithm yields a contracted graph G' that,*

1. *has at most $O(n \log n/\delta(G))$ vertices with probability at least $1 - 1/n^4$, and*
2. *has $\lambda(G') \geq \lambda(G)$ always, equality is achieved with probability at least $2 \cdot 3^{-13}$.*

Note that in the cut query model, the algorithm cannot contract an edge because the underlying cut function remains constant. However, it can simulate the contraction of $e = (u,v)$ by merging the vertices u and v into a supervertex in all subsequent queries, which yields a contracted cut function. We now define our *τ-star contraction* algorithm.

▶ **Definition 2.5** (τ-star contraction)**.** *Given an unweighted graph $G = (V,E)$ on n vertices and a threshold $\tau \in [n]$, τ-star contraction is the following operation.*

1. *Let $H = \{v \in V \mid d(v) \geq \tau\}$ be the set of high-degree vertices.*
2. *Form the set $R \subseteq V$ by sampling each $v \in V$ independently with probability $p = \Theta(\log n/\tau)$.*
3. *For each $u \in H \setminus R$, pick an edge from $E(u,R)$, if non-empty, uniformly at random and contract it (keeping parallel edges).*

Notice that τ-star contraction depends on uniform edge sampling, since each vertex $u \in H \setminus R$ samples a neighbor in R. In our case, we can use the uniform edge sampling procedure described above to sample edges uniformly from $E(u, R)$.

We now sketch the proof that τ-star contraction yields a graph with few edges. To bound the number of edges, partition the vertices remaining after the contraction into two sets, R, i.e. the center vertices sampled in the sampling process, and $L = V \setminus H$, i.e. the low degree vertices. This analysis omits vertices in $H \setminus R$ that were not sampled into R or contracted, because every $v \in H \setminus R$ has in expectation $d_G(v) \cdot \log n/\tau \in \Omega(\log n)$ neighbors in R; hence with high probability it will have some neighbor in R and be contracted.

To bound the number of edges observe that $|R| \leq \tilde{O}(n/\tau)$ with high probability, and hence the number of edges between vertices in R is $O(|R|^2) \leq \tilde{O}(n^2/\tau^2)$. In addition, since the degree of the vertices of L is bounded by τ, the total number of edges incident on L, even before the contraction, is $O(n\tau)$. Hence, the total number of edges in G' is with high probability $\tilde{O}(n^2/\tau^2 + n\tau)$. The following lemma, proved in the full version of the paper, formalizes the guarantees of τ-star contraction.

▶ **Lemma 2.6.** *Let $G = (V, E)$ be an unweighted graph on n vertices with a non-trivial minimum-cut $\lambda(G)$ and let $\tau \in [n]$. A τ-star contraction of G gives G' that,*
1. *has at most $\tilde{O}(n^2/\tau^2 + n\tau)$ edges with probability at least $1 - 1/n^4$, and*
2. *has $\lambda(G') \geq \lambda(G)$ always, equality is achieved with probability at least $2 \cdot 3^{-13}$.*

We present in Algorithm 1 a procedure for finding the global minimum cut in two rounds using τ-star contraction. We will use it to prove Theorem 1.3, but first we need the following claim.

Algorithm 1 Unweighted Minimum Cut in 2 Rounds.

1: **Input:** Graph $G = (V, E)$
2: **Output:** A minimum cut $\mathcal{C} \subseteq V$
3: sample a set $R \subseteq V$ uniformly with probability $p = 400 \cdot \log n/\tau$
// Query round 1
4: query $\text{cut}_G(v)$ for all $v \in V$ ▷ find $d_G(v)$ for all $v \in V$
5: for each $v \in V \setminus R$, let $n(v) \leftarrow$ a random neighbor in $N_G(v) \cap R$ ▷ use Lemma 2.1
6: $\tau \leftarrow \max\{n^{1/3}, \delta(G)\}$
7: $V' \leftarrow V$
8: **for** $v \in \{V \setminus R \mid \text{cut}_G(v) \geq \tau\}$ **do**
9: update V' by contracting the edge $(v, n(v))$
// Query round 2
10: recover $G' = (V', E')$ with $\tilde{O}(n^{2-2/3} + n^{1+1/3})$ queries ▷ use Corollary 2.3, assuming $|E'| = \tilde{O}(n^{2-2/3} + n^{1+1/3})$
11: **return** $\min(\lambda(G'), \delta(G))$ ▷ can return also a set $S \subseteq V$ achieving the cut

▷ Claim 2.7. With probability $1 - n^{-3}$, line 9 of Algorithm 1 implements τ-star contraction with $\tau = \max\{n^{1/3}, \delta(G)\}$.

Proof. Notice that R is sampled uniformly from V and that line 9 is executed only for vertices in $H \setminus R$. It remains to show that the edge $(v, n(v))$ is sampled uniformly from $N_G(v) \cap R$, which indeed follows from Lemma 2.1. To analyze the success probability, note that the algorithm samples at most n edges in line 5 and by the guarantee of Lemma 2.1 this fails with probability at most $n \cdot n^{-4} = n^{-3}$. ◁

Proof of Theorem 1.3. The main argument is that Algorithm 1 finds a minimum cut of G with constant probability. Then, running this algorithm $O(\log n)$ times in parallel and returning the minimum output obtained gives the desired result. Notice that by Lemma 2.6, the connectivity of G' can only increase, hence, if the algorithm finds a minimum cut of G in some execution it will be returned. We assume throughout the proof that in all executions of the algorithm the size guarantee of Lemma 2.6 holds, i.e. $|E'| = \tilde{O}(n^{2-2/3} + n^{1+1/3})$, note that this holds with probability $1 - \tilde{O}(n^{-4})$. Using a union bound with Claim 2.7 above, we find that the algorithm succeeds with probability at least $1 - n^{-2}$.

By Claim 2.7 the graph G' is the result of the τ-star contraction with $\tau = n^{1/3}$, and thus $\lambda(G') = \lambda(G)$ with constant probability by Lemma 2.6. Therefore, recovering the graph G' and returning the minimum of $\lambda(G')$ and $\delta(G)$ gives a minimum cut of G with constant probability.

To bound the query complexity, observe that the query complexity of the algorithm is dominated by the number of queries in the graph recovery step, which is given by $\tilde{O}(n^{2-2/3} + n^{1+1/3}) = \tilde{O}(n^{4/3})$. In the first round, we use $\tilde{O}(n)$ queries as finding the degree of each vertex requires n queries, and sampling a neighbor in $E(v, R)$ can be done using $O(\log^3 n / \log\log n)$ queries by Lemma 2.1. This concludes the analysis of the query complexity of the algorithm, which is $\tilde{O}(n^{4/3})$.

Finally, notice that the algorithm proceeds in two rounds; in the first round the algorithm only queries the degrees of the vertices and samples an edge in $E(v, R)$ for every v, which does not require any prior information on the graph. In the second round, it recovers G' round using Corollary 2.3. This concludes the proof of Theorem 1.3. ◀

2.3 Unweighted Minimum Cut in $2r$ rounds

This section provides an algorithm for finding a minimum cut in $2r$ rounds, thus proving Theorem 1.4. The algorithm uses a known approach of first applying a contraction algorithm, yielding a graph with $\tilde{O}(n/\delta(G))$ vertices, and then packing $\delta(G)$ edge-disjoint forests in the contracted graph. The main challenge in adapting this algorithm to low round complexity, is ensuring that the forests are edge disjoint when packing them in parallel. This is achieved by leveraging a connection to cut sparsification, which shows that sampling edges with probabilities amplified by a factor of k relative to those needed for sparsifier construction ensures finding k edge disjoint forests within the sampled subset. We note that we also offer another algorithm for finding a minimum cut, that has slightly worse query complexity of $O(rn^{1+1/r})$, and is based on constructing a cut sparsifier and the results of [33], see the full version of the paper.

The contraction we use is the 2-*out contraction* procedure of [26], which is obtained by sampling uniformly two edges incident to each vertex and contracting all the sampled edges. The following theorem states the guarantees of the 2-out contraction.

▶ **Theorem 2.8** (Theorem 2.4 in [26]). *Let $G = (V, E)$ be an unweighted graph on n vertices with a non-trivial minimum-cut value $\lambda(G)$. Then, applying a 2-out contraction algorithm gives G' where,*

1. *G' has at most $\tilde{O}(n/\delta(G))$ vertices with high probability, and*
2. *has $\lambda(G') \geq \lambda(G)$ always, equality is achieved with constant probability.*

The other building block of the algorithm is maximal k-packing of forests.

▶ **Definition 2.9** (Maximal k-Packing of Forests). *Given a (possibly weighted) graph G, a maximal k-packing of forests is a set of edge-disjoint forests $T_1, \ldots, T_k$ that are maximal, i.e. for every $i \in [k]$ and $e \in E \setminus \cup_j T_j$, the edge set $T_i \cup \{e\}$ contains a cycle.*

▶ **Lemma 2.10.** *Given a (possibly weighted) graph G on n vertices and parameters $r \in \{1, \ldots, \log n\}$ and $k \in [n]$, one can find a maximal k-packing of forests of G using $\tilde{O}(krn^{1+1/r})$ cut queries in $2r$ rounds. The algorithm is randomized and succeeds with probability $1 - n^{-4}$.*

We now outline the algorithm of Lemma 2.10, for further details see the full version of the paper. The algorithm begins by creating a set of empty trees $T_1, \ldots, T_k$. Then, in each of $r-1$ rounds it uniformly samples $\tilde{O}(kn^{1+1/r})$ edges from the graph, augments the trees $\{T_i\}$ by adding all sampled edges that do not form a cycle, and contracts all vertices that are connected in all trees. We show that the number of edges decreases by an $n^{1/r}$ factor in each round; hence, after $r-1$ rounds the number of remaining edges is $O(n^{1+1/r})$. Finally, the algorithm recovers all remaining edges by sampling $\tilde{O}(n^{1+1/r})$ edges uniformly, which recovers all edges by a standard coupon collector argument, and augments the trees appropriately.

To guarantee the decrease in the number of edges we leverage an elegant connection to cut sparsifier contraction. The seminal work [13] showed that if we sample (and reweight) every edge with probability proportional to its strong connectivity, then the resulting graph approximates all the cuts of the original graph with high probability.

▶ **Definition 2.11** (Strength of an Edge)**.** *Let $G = (V, E, w)$ be a weighted graph. We say that $S \subseteq V$ is* κ-strong *if the minimum cut of the induced graph $G[S]$ is at least κ. An edge $e \in E$ is κ-strong if it is contained within some κ-strong component S.*

We also need the following lemma concerning the existence of κ-strong components in a graph.

▶ **Lemma 2.12** (Lemma 3.1 in [12])**.** *Every weighted graph H with n vertices and total edge weight at least $\kappa(n-1)$ must have a κ-strong component.*

We can now detail the process of edge reduction of Lemma 2.10. Let G be a graph with m edges and n vertices. Observe that after one round of sampling, the algorithm contracts every $(m/n^{1+1/r})$-strong component of G, and thus by Lemma 2.12 the number of edges remaining in the graph is at most $m/n^{1/r}$. Let $C \subseteq V$ be a maximal $\Omega(m/n^{1+1/r})$-strong component of G, and assume the algorithm samples every edge of the graph with probability $p_e \approx k/\kappa$. In expectation the algorithm samples at least $p_e \cdot m/n^{1+1/r} \approx k$ edges crossing each cut $S \subseteq C$, which also holds with high probability by standard concentration and cut counting results. Therefore, for every $u, v \in C$, the minimum cut in the sampled subgraph has at least k edges, hence they are connected in k edge disjoint forests and will be contracted. In conclusion, in every iteration the algorithm contracts all $(m/n^{1+1/r})$-strong components of the graph, and the number of edges in the graph decreases by a factor of $n^{1/r}$.

Proof of Theorem 1.4. The algorithm begins by performing a 2-out contraction on G to obtain a graph G' with $\tilde{O}(n/\delta(G))$ vertices which preserves each non-trivial minimum cut of G with constant probability by Theorem 2.8. In parallel, the algorithm learns $\delta(G)$ by querying the degree of each vertex. Then, it packs $k = \delta(G)$ trees $\{T_i\}_{i=1}^{k}$ in G' using Lemma 2.10. Notice that since the minimum cut of G is at most $\delta(G)$ and each edge has weight at least 1 then $\delta(G)$ trees in G' will include all edges in the minimum cut. Hence, returning the minimum between $\delta(G)$ and the minimum cut of $H = (V, E_H = \cup_i T_i)$ will yield the minimum cut of G with constant probability. To amplify the success probability we repeat the algorithm $O(\log n)$ times in parallel and return the minimum cut found.

To analyze the query complexity notice that by Lemma 2.1, sampling the edges for the 2-out contraction takes $\tilde{O}(n)$ queries in one round. Hence, the algorithm constructs G' in $\tilde{O}(n)$ queries in one round. Then, it packs $k = \delta(G)$ trees in G', which has only $\tilde{O}(n/k)$ vertices, using $\tilde{O}(\delta(G)r(n/\delta(G))^{1+1/r})$ queries in $2r$ rounds by Lemma 2.10. Hence, overall the algorithm require $\tilde{O}(\delta(G)^{-1/r}rn^{1+1/r})$ queries in $2r+1$ rounds.

We conclude the proof by analyzing the success probability of the algorithm. Our implementation of the 2-out contraction succeeds with probability $1 - n \cdot n^{-4}$ since it only requires sampling $O(n)$ edges using Lemma 2.1. Then, each tree packing succeeds with probability $1 - 2n^{-2}$ by Lemma 2.10. By a union bound, the entire algorithm succeeds with probability $1 - n^{-1}$, which concludes the proof of Theorem 1.4. ◀

2.4 Weighted Minimum Cut in $O(r)$ rounds

In this section we give an overview of our algorithm for minimum cut in a weighted graph with low adaptivity. It is an adaptation of the minimum cut algorithm of [31], which reduces the problem to constructing a cut sparsifier and solving a monotone matrix problem which is easily parallelizable. The main difficulty lies within the construction of a cut sparsifier; while previous work in the cut-query model already provided cut sparsifiers [33, 32], its core step, weight-proportional edge sampling, uses $O(\log n)$ rounds. Our main technical contribution here is our two-round weight-proportional edge sampling primitive (Lemma 2.2), which enables the construction of a $(1 \pm \epsilon)$-cut-sparsifier in $3r$ rounds and the completion of the entire reduction within $O(r)$ rounds.

The algorithm of [31] for finding a minimum cut in a weighted graph is actually based on the 2-respecting minimum-cut framework of [27]; where a spanning tree $T \subseteq E$ is called 2-*respecting* for a cut $C \subseteq E$ if $|C \cap T| \leq 2$. The main insight here is that a given a spanning tree T that is 2-respecting for a minimum cut C, there are only $O(n^2)$ cuts that one needs to check to find a minimum cut, and they correspond to all pairs of edges in T. Originally [27], proposed a clever dynamic algorithm to calculate all these cuts quickly. In contrast, the algorithm of [31] leverages the structure of the 2-respecting spanning tree to show that the number of potential cuts is actually much smaller; restricting attention to sub-trees, $\{T_i\}$, that are all paths and whose total size is bounded. Furthermore, when T is a path the task of finding a minimum cut reduces to the following problem.

▶ **Definition 2.13** (Monotone Matrix Problem). *Let $A \in \mathbb{R}_+^{a \times a}$ be a matrix. For every column j denote the first row in which j attains its minimum value by j_s and the last row in which j attains its minimum value by j_t. The matrix is called* monotone *if for every $i > j$ we have $j_t \leq i_s$; i.e. the row in which the minimum value is found increases monotonically as we move to the right.*

The monotone matrix problem *is to find the minimum value of a monotone matrix (of dimension $a \times a$) while querying the value of as few entries as possible.*

In the context of the minimum cut problem, each entry in the matrix corresponds to a cut in G and can be recovered using a single cut query. We summarize some results from [31] in the following lemma, whose proof is provided in the full version of the paper.

▶ **Lemma 2.14.** *Assume there is an algorithm to construct a $(1 \pm \epsilon)$-cut-sparsifier for a graph G on n vertices using $q_S(n)$ queries in $r_S(n)$ rounds, and an algorithm that solves the monotone matrix problem on instances of size n using $q_M(n)$ queries in $r_M(n)$ rounds. If in addition the function q_M is convex, then it is possible to find a minimum cut of G using $\tilde{O}(q_S(n) + q_M(n) + n)$ cut queries in $r_S(n) + r_M(n) + 2$ rounds.*

Therefore, in order to find a minimum cut it suffices to construct a cut sparsifier and then solve the monotone matrix problem. Our main technical contribution in this section is a construction of a cut sparsifier with low adaptivity. We begin by formally defining the notion of a cut sparsifier.

▶ **Definition 2.15.** *Let $G = (V, E, w)$ be a weighted graph and let $H = (V, E_H, w_H)$ be a weighted subgraph of G. We say that H is a* $(1 \pm \epsilon)$-cut-sparsifier *of G if,*

$$\forall S \subseteq V, \qquad \mathrm{cut}_H(S) \in (1 \pm \epsilon)\mathrm{cut}_G(S).$$

▶ **Lemma 2.16.** *Given a weighted graph $G = (V, E, w)$ on n vertices with integer edge weights bounded by W, and a parameter $r \in \{1, 2, \ldots, \log n\}$, one can construct a $(1+\epsilon)$-cut-sparsifier of G using $\tilde{O}(\epsilon^{-2} r n^{1+(1+\log_n W)/r})$ cut queries and $3r + 3$ rounds. The algorithm is randomized and succeeds with probability $1 - n^{-2}$.*

We now sketch the proof of Lemma 2.16, for more details see the full version of the paper. The sparsifier construction is similar to the one in [33, 32] but modified to work in r rounds. It follows the sampling framework of [13], that shows that if one samples every edge in a graph with probability proportional to its strong connectivity, then the sampled graph is a $(1 \pm \epsilon)$-cut-sparsifier with high probability. The algorithm is based on iteratively finding the strongest components of the graph using weight-proportional edge sampling, assigning a strength estimate to all their edges, and contracting them. It repeats this process until all vertices are contracted. Finally, it samples all the edges in the graph according to their strength estimates, and returns the sampled edges as the sparsifier.

Our algorithm improves on the round complexity of the existing constructions in two ways: the first one is the adoption of a two-round weight-proportional sampling algorithm that eliminates an $O(\log n)$ factor in the round complexity. The second improvement is in the number of contraction steps performed. The algorithm of [33, 32] uses $O(\log(nW))$ steps, where in each step strengths are approximated within factor 2. Our algorithm uses instead r steps, where each step contracts all components with strength within an $n^{(1+\log_n W)/r}$ factor. Since the maximum component strength is $O(n^{1+\log_n W})$, the algorithm can contract all the components within r steps.

Our second technical contribution is an algorithm that solves the monotone matrix problem in few rounds, the proof is provided in the full version of the paper.

▶ **Lemma 2.17.** *There exists an algorithm that solves the monotone matrix problem on instances of size n using $\tilde{O}(n^{1+1/r})$ queries in r rounds.*

The algorithm for the monotone matrix problem in [31] uses a divide-and-conquer approach. At each step the algorithm reads the entire middle column j of the matrix A and finds the minimum value in the column. Then, it splits the matrix into two submatrices $A_L = A[1, \ldots, j_s; 1, \ldots, j]$ and $A_R = A[j_t, \ldots n; j+1, \ldots n]$ based on the location of the minimum value in the column. The algorithm is guaranteed to find the minimum value in each column by the monotonicity property of the matrix. Finally, it is easy to verify that as the size of the problem decreases by factor 2 in each iteration, the overall complexity of the algorithm is $\tilde{O}(n)$.

Unfortunately, the algorithm of [31] requires $O(\log n)$ rounds to complete all the recursion calls. Our algorithm instead partitions the matrix into $n^{1/r}$ blocks and then recurses on each block, which requires a total of $O(n^{1+1/r})$ queries in r rounds.

To conclude this section, notice that Theorem 1.5 follows by combining Lemma 2.14, Lemma 2.16, and Lemma 2.17.

2.5 Open Questions

Optimal Tradeoff Between Rounds vs Queries Complexity. An interesting open question is whether our constructive results have matching lower bounds or, alternatively, if better algorithms exist. Currently, there are no known query lower bounds even for algorithms

that use a single round of queries. We note that connectivity can be solved in one round with $\tilde{O}(n)$ queries [4], however it remains unclear whether this can be extended to finding a minimum cut, or whether there exists a lower bound of $\Omega(n^{1+\epsilon})$ cut queries for one round algorithms.

Adaptivity of Deterministic Algorithms. A deterministic algorithm for finding a minimum cut in simple graphs using $\tilde{O}(n^{5/3})$ cut queries was recently shown in [2], however its round complexity is polynomial in n, and it seems challenging to achieve polylogarithmic round complexity. In comparison, graph connectivity can be solved deterministically using $\tilde{O}(n^{1+1/r})$ cut queries in $O(r)$ rounds [4],[6] though these techniques do not seem to extend to finding a minimum cut.

References

1 Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. In *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPIcs*, pages 2:1–2:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.2.

2 Aditya Anand, Thatchaphol Saranurak, and Yunfan Wang. Deterministic edge connectivity and max flow using subquadratic cut queries. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 124–142. SIAM, 2025. doi:10.1137/1.9781611978322.4.

3 Simon Apers, Yuval Efron, Pawel Gawrychowski, Troy Lee, Sagnik Mukhopadhyay, and Danupon Nanongkai. Cut query algorithms with star contraction. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 507–518. IEEE, 2022. doi:10.1109/FOCS54457.2022.00055.

4 Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and OR queries. In *29th Annual European Symposium on Algorithms, ESA 2021*, volume 204 of *LIPIcs*, pages 7:1–7:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.7.

5 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Polynomial pass lower bounds for graph streaming algorithms. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 265–276. ACM, 2019. doi:10.1145/3313276.3316361.

6 Arinta Auza and Troy Lee. On the query complexity of connectivity with global queries. *CoRR*, abs/2109.02115, 2021. arXiv:2109.02115.

7 Eric Balkanski, Adam Breuer, and Yaron Singer. Non-monotone submodular maximization in exponentially fewer iterations. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018*, pages 2359–2370, 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/a42a596fc71e17828440030074d15e74-Abstract.html.

8 Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 283–302. SIAM, 2019. doi:10.1137/1.9781611975482.19.

9 Eric Balkanski and Yaron Singer. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1138–1151. ACM, 2018. doi:10.1145/3188745.3188752.

[6] The result is for the weaker Bipartite Independent Set (BIS) query model, which can be simulated using cut queries.

10 Eric Balkanski and Yaron Singer. A lower bound for parallel submodular minimization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 130–139. ACM, 2020. doi:10.1145/3357713.3384287.

11 Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. Algorithms*, 16(4):52:1–52:27, 2020. doi:10.1145/3404867.

12 András A. Benczúr and David R. Karger. Approximating $s-t$ minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 47–55. ACM, 1996. doi:10.1145/237814.237827.

13 András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015. doi:10.1137/070705970.

14 Nader H. Bshouty and Hanna Mazzawi. Reconstructing weighted graphs with minimal query complexity. *Theor. Comput. Sci.*, 412(19):1782–1790, 2011. doi:10.1016/J.TCS.2010.12.055.

15 Nader H. Bshouty and Hanna Mazzawi. Toward a deterministic polynomial time algorithm with optimal additive query complexity. *Theor. Comput. Sci.*, 417:23–35, 2012. doi:10.1016/J.TCS.2011.09.005.

16 Deeparnab Chakrabarty, Yu Chen, and Sanjeev Khanna. A polynomial lower bound on the number of rounds for parallel submodular function minimization. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 37–48. IEEE, 2021. doi:10.1109/FOCS52979.2021.00013.

17 Deeparnab Chakrabarty, Andrei Graur, Haotian Jiang, and Aaron Sidford. Improved lower bounds for submodular function minimization. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 245–254. IEEE, 2022. doi:10.1109/FOCS54457.2022.00030.

18 Deeparnab Chakrabarty, Andrei Graur, Haotian Jiang, and Aaron Sidford. Parallel submodular function minimization. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/d8a7f2f7e346410e8ac7b39d9ff28c4a-Abstract-Conference.html.

19 Deeparnab Chakrabarty and Hang Liao. A query algorithm for learning a spanning forest in weighted undirected graphs. In *International Conference on Algorithmic Learning Theory 2023*, volume 201 of *Proceedings of Machine Learning Research*, pages 259–274. PMLR, 2023. URL: https://proceedings.mlr.press/v201/chakrabarty23a.html.

20 Lin Chen, Moran Feldman, and Amin Karbasi. Unconstrained submodular maximization with constant adaptive complexity. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 102–113. ACM, 2019. doi:10.1145/3313276.3316327.

21 Yixin Chen, Tonmoy Dey, and Alan Kuhnle. Best of both worlds: Practical and theoretically optimal submodular maximization in parallel. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021*, pages 25528–25539, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/d63fbf8c3173730f82b150c5ef38b8ff-Abstract.html.

22 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 749–758. ACM, 2008. doi:10.1145/1374376.1374484.

23 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. doi:10.1016/J.JALGOR.2003.12.001.

24 Alina Ene and Huy L. Nguyen. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 274–282. SIAM, 2019. doi:10.1137/1.9781611975482.18.

25 Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 255–273. SIAM, 2019. doi:10.1137/1.9781611975482.17.

26 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1260–1279. SIAM, 2020. doi:10.1137/1.9781611975994.77.

27 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. doi:10.1145/331605.331608.

28 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019. doi:10.1145/3274663.

29 Troy Lee, Miklos Santha, and Shengyu Zhang. Quantum algorithms for graph problems with cut queries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 939–958. SIAM, 2021. doi:10.1137/1.9781611976465.59.

30 Hang Liao and Deeparnab Chakrabarty. Learning spanning forests optimally in weighted undirected graphs with CUT queries. In *International Conference on Algorithmic Learning Theory*, volume 237 of *Proceedings of Machine Learning Research*, pages 785–807. PMLR, 2024. URL: https://proceedings.mlr.press/v237/liao24b.html.

31 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 496–509. ACM, 2020. doi:10.1145/3357713.3384334.

32 Orestis Plevrakis, Seyoon Ragavan, and S. Matthew Weinberg. On the cut-query complexity of approximating max-cut. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024*, volume 297 of *LIPIcs*, pages 115:1–115:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.115.

33 Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018*, 2018. doi:10.4230/LIPICS.ITCS.2018.39.

34 Xiaoming Sun, David P. Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 94:1–94:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.94.

Improved Algorithms for Quantum MaxCut via Partially Entangled Matchings

Anuj Apte
University of Chicago, IL, USA

Eunou Lee
Korea Institute for Advanced Study, Seoul, South Korea

Kunal Marwaha
University of Chicago, IL, USA

Ojas Parekh
Sandia National Laboratories, Albuquerque, NM, USA

James Sud
University of Chicago, IL, USA

Abstract

We introduce a 0.611-approximation algorithm for Quantum MaxCut and a $\frac{1+\sqrt{5}}{4} \approx 0.809$-approximation algorithm for the EPR Hamiltonian of [12]. A novel ingredient in both of these algorithms is to *partially* entangle pairs of qubits associated to edges in a matching, while preserving the direction of their single-qubit Bloch vectors. This allows us to interpolate between product states and matching-based states with a tunable parameter.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Quantum computing, Quantum MaxCut, Maximum matching

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.101

Related Version *Full Version*: `https://arxiv.org/pdf/2504.15276` [2]

Funding K.M and J.S. acknowledge that this material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 2140001.
Eunou Lee: E.L is supported by a KIAS Individual Grant CG093801 at Korea Institute for Advanced Study.
Kunal Marwaha: K.M acknowledges support from AFOSR (FA9550-21-1-0008).
Ojas Parekh: O.P. acknowledges that this material is based upon work supported by the U.S. Department of Energy, Office of Science, Accelerated Research in Quantum Computing, Fundamental Algorithmic Research toward Quantum Utility (FAR-Qu).
James Sud: J.S acknowledges that this work is funded in part by the STAQ project under award NSF Phy-232580; in part by the US Department of Energy Office of Advanced Scientific Computing Research, Accelerated Research for Quantum Computing Program.

Acknowledgements This work is supported by a collaboration between the US DOE and other Agencies. The work of A.A is supported by the Data Science Institute at the University of Chicago. This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan `https://www.energy.gov/downloads/doe-public-access-plan`.



© Anuj Apte, Eunou Lee, Kunal Marwaha, Ojas Parekh, and James Sud;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Given a graph $G(V, E, w)$ with positive edge weights $w : E \to \mathbb{R}_+$ and a 2-local Hamiltonian term h, define the n-qubit Hamiltonian

$$H_G \stackrel{\text{def}}{=} \sum_{(i,j)\in E(G)} w_{ij} \cdot h_{ij} \,,$$

where h_{ij} is the local term h applied on qubits (i, j). For two particular local terms h, we are interested in the problem of computing the maximum energy of H_G, which we denote as $\lambda_{max}(H_G)$, for any G. This is not an easy task in general: deciding if $\lambda_{max}(H_G)$ is above some threshold with inverse polynomial accuracy is known to be QMA-hard [18].

In the first problem, we choose the local term

$$h_{ij}^{QMC} \stackrel{\text{def}}{=} \frac{1}{2}\left(I_iI_j - X_iX_j - Y_iY_j - Z_iZ_j\right) = 2\left|\psi_-\right\rangle_{ij}\left\langle\psi_-\right|_{ij} \,,$$

where $|\psi_-\rangle = \frac{1}{\sqrt{2}}\left(|01\rangle - |10\rangle\right)$ is the singlet state. This problem has recently been studied under the name *Quantum MaxCut* (QMC) [7]. In the statistical mechanics literature, Hamiltonians H_G defined by h^{QMC} are instances of the zero-field quantum Heisenberg $\mathrm{XXX}_{1/2}$ model. The decision version of QMC is QMA-hard [18].

In the second problem, we choose the local term

$$h_{ij}^{EPR} \stackrel{\text{def}}{=} \frac{1}{2}\left(I_iI_j + X_iX_j - Y_iY_j + Z_iZ_j\right) = 2\left|\phi_+\right\rangle_{ij}\left\langle\phi_+\right|_{ij} \,,$$

where $|\phi_+\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$. This problem, named *EPR* by [12], is thought to be easier than QMC, since Hamiltonians H_G defined by h^{EPR} are *stoquastic* (sign-problem free) [18, 5]. In fact, it is not yet clear if EPR can be solved in polynomial time.

In lieu of exactly computing $\lambda_{max}(H_G)$, we may try to *approximate* this value. In both problems, the local term h is positive semidefinite, so $\lambda_{max}(H_G) \geq 0$. We judge an approximation by its *approximation ratio*. Suppose we can find efficiently computable functions ℓ, u such that for all graphs G,

$$0 \leq \ell(G) \leq \lambda_{max}(H_G) \leq u(G) \,.$$

Then, the approximation ratio α is at least

$$\alpha \geq \min_G \frac{\ell(G)}{\lambda_{max}(H_G)} \geq \min_G \frac{\ell(G)}{u(G)} \,.$$

In most works, the upper bound u is determined by solving a semidefinite programming (SDP) relaxation of the maximization problem [3, 7, 16, 19, 20, 9]. The source of such SDP relaxations are generally hierarchies of SDPs that provide increasingly better upper bounds u at the cost of solving larger-sized SDPs. The quantum moment-SOS hierarchy, typically based on Pauli operators, is widely employed for quantum local Hamiltonian problems. This hierarchy is an instance of the NPA hierarchy [15] and is also known as the quantum Lasserre hierarchy [16]. The second level of the quantum moment-SOS hierarchy is necessary for good approximations, since critical monogamy-of-entanglement properties begin to emerge at that level [16, 17].

The lower bound ℓ is usually determined by an algorithm that prepares a state with this energy. Several approximation algorithms have been proposed for QMC and EPR, using product states [7, 16, 17, 12], matchings [1, 16, 17, 14, 10, 8], and short variational

circuits [1, 13, 12, 11, 8]. During the preparation of this manuscript, the best known approximation ratios for these problems were improved to $\alpha \geq 0.603$ for QMC [8] and $\alpha \geq \frac{1+\sqrt{5}}{4} \approx 0.809$ for EPR [11].

We provide a new algorithm for both problems. For each algorithm, we start from a good product state. We then choose a matching in G and *partially* rotate matched qubits towards an entangled state. Our novel contribution is to perform this rotation such that individual single-qubit Bloch vectors are preserved, up to a rescaling of magnitude. This allows us to evaluate the energy of our entangled state in terms of the energy of the product state. For EPR, we use a *fractional* matching and provide a circuit-based algorithm. For QMC, we explicitly describe a tensor-product of single and two-qubit states, which interpolates between the product state and an *integer* matching-based state with a single tunable parameter. For QMC, we additionally introduce the technique of choosing a maximum-weight matching with respect to *rescaled* edge weights. One interpretation of this approach is that we widen the search space of our algorithm to include matchings other than the maximum-weight matching of G. Combining these techniques allows us to achieve state-of-the-art approximation ratios on each problem:

▶ **Theorem 1.** *For EPR,* $\alpha \geq \frac{\varphi}{2} \approx 0.809$*, where* $\varphi \stackrel{\text{def}}{=} \frac{1+\sqrt{5}}{2} \approx 1.618$ *is the golden ratio.*

▶ **Theorem 2.** *For QMC,* $\alpha \geq 0.611$*.*

▶ Remark 3. Theorem 1 was shown simultaneously and independently by [11] using a different algorithm. The algorithm of [11] also *partially* entangles edges while preserving single-qubit marginals. However, we believe both our algorithm and analysis are simpler.

▶ Remark 4. Theorem 2 requires an improved upper bound in addition to new algorithmic techniques. The technique of finding a matching with respect to rescaled edge weights was simultaneously and independently used by [8]; however, they use this as part of an improved approximation algorithm for QMC on triangle-free graphs rather than general graphs. Our QMC approximation does rely on a strengthening by [8] of a class of upper bounds first used in the work [14]. Without this strengthened bound, we still achieve an approximation ratio $\alpha \geq 0.610$.

Our approximation algorithm finds a quantum state ρ_G such that $\mathrm{Tr}\left(H_G \rho_G\right) \geq \ell(G)$. For the EPR Hamiltonian, the analysis of our algorithm is optimal: there exist graphs G (such as the single-edge graph) where

$$\frac{\ell(G)}{u(G)} \leq \frac{\langle \psi_G | H_G | \psi_G \rangle}{\lambda_{max}(G)} = \frac{1+\sqrt{5}}{4}.$$

For QMC, however, it is possible that a better analysis of this algorithm (particularly of $u(G)$) would give a larger approximation ratio.

We stress the novelty of this work is *new algorithmic techniques* for QMC and EPR. We use these techniques to immediately improve the approximation ratios for each algorithm, however we believe that these techniques, potentially combined with previous methods, may lead to further improvements in approximation ratios for both algorithms. In the remainder of this work, we introduce necessary notation, describe each algorithm, and prove Theorems 1 and 2. We defer some technical lemmas to the appendix, which can be found in the full version of this paper [2].

2 Preliminaries

2.1 Graph theory

Let $G(V, E, w)$ denote a graph with vertex set V, edge set E, and positive edge weights $w : E \to \mathbb{R}_+$. In this work, we always take $V = [n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$. We let $W_G \stackrel{\text{def}}{=} \sum_{(i,j)\in E} w_{ij}$. When G is inferred by context, we simply write W. Define $N(v)$ as the set of neighbors of a vertex $v \in V$. For convenience, we index edges by either e or (i, j), or simply ij in subscripts.

A fractional *matching* of a graph $G(V, E, w)$ is a function $m : E(G) \to [0, 1]$ that assigns a value to each edge in G, such that the sum over values $\sum_{j\in N(i)} m_{ij}$ is at most 1 for any vertex i. If $m_{ij} \in \{0, 1\}$ for all edges (i, j), we say that m is an *integral* matching. For an integral matching m, we say an edge (i, j) is in the matching if $m_{ij} = 1$. The *weight* $\mathrm{Wt}(m)$ of a matching m is defined as $\mathrm{Wt}(m) \stackrel{\text{def}}{=} \sum_{(i,j)\in E(G)} w_{ij}\, m_{ij}$. We define FM_G (or M_G) to be the maximum total weight of any fractional (or integral) matching of G. For convenience, we sometimes let M_G also denote the set of edges in the maximum-weight integral matching of G. When the graph is clear from context we drop the subscript G. Optimal integral and fractional matchings can be computed in polynomial time, for example with linear programming [6].

2.2 Quantum computation

We refer to $\vec{\sigma} = (X, Y, Z)$ as the canonical Pauli matrices. As such, we can define the *Bloch vector* $\vec{B}(\rho)$ of any 1-qubit density matrix ρ as the vector (b_x, b_y, b_z) in the unit sphere S^2 such that

$$\rho = \frac{1}{2}(I + \vec{B}(\rho) \cdot \vec{\sigma}) \stackrel{\text{def}}{=} \frac{1}{2}(I + b_x X + b_y Y + b_z Z)\,.$$

Pure states correspond to unit Bloch vectors. Let $|\psi_-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$ be the *singlet state*, and $|\phi_+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ be the *EPR state*.

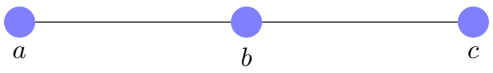
2.3 Previous algorithms

In our work, we use two previously known algorithms as subroutines:

1. The algorithm PROD takes a graph $G(V, E, w)$ as an input, and outputs a good product state ρ_{PROD}. For EPR, we define $\rho_{PROD} \stackrel{\text{def}}{=} |0\rangle^{\otimes n} \langle 0|^{\otimes n}$. For QMC, we define PROD to be the output of the [7] rounding algorithm applied to the some fixed constant level k of the quantum moment-SOS hierarchy, as in [17] and [14]. This is the only part of our algorithm for QMC that relies on an SDP. As described in [17, Sec 2.2], the output from the SDP can be interpreted as a *pseudo-density matrix*. It is called a *pseudo*-density matrix because it satisfies only some of the constraints of a valid density matrix. Furthermore, the SDP at any constant ℓ can be solved in polynomial time.
2. The algorithm MATCH takes a graph $G(V, E, w)$ as an input, and outputs a product of 2-qubit states ρ_{MATCH}. It was first formally proposed in [14]. The algorithm first finds a maximum-weight integral matching M_G. For EPR (and for QMC), it outputs the tensor product of $|\phi_+\rangle_{ij} \langle\phi_+|_{ij}$ (and $|\psi_-\rangle_{ij} \langle\psi_-|_{ij}$ for QMC) for every pair of vertices (i, j) in the matching, and the maximally mixed state for every vertex i not in the matching. This obtains energy 2 on matched edges and 1/2 on unmatched edges, for a total of $(3\mathrm{M}_G + W_G)/2$.

3 Our Algorithms

To motivate our algorithms, consider the following simple graph G:



This graph is bipartite, so QMC and EPR are equivalent under local rotations [12]. We thus focus on EPR for simplicity. It is easy to compute that $\lambda_{max}(H_G) = 3$. The optimal product state ρ_{PROD} is $\rho_0^{\otimes 3}$, where $\rho_0 \stackrel{\text{def}}{=} |0\rangle\langle 0|$, and achieves energy 2. The algorithm MATCH computes the maximum matching $\mathrm{M}_G = \{(a,b)\}$ and returns the state ρ_{MATCH} as described in Section 2. This state gains energy 2 on edge (i,j) and 1/2 on edge (b,c), achieving total energy 5/2. Thus, the algorithm of [14], which returns the better of ρ_{PROD} and ρ_{MATCH}, achieves $5/6 \approx 0.833$ of the optimal energy.

Our algorithms do better by interpolating between ρ_{PROD} and ρ_{MATCH}, rather than taking the better of the two. For our example G, the unitary

$$U = e^{i\theta\left(\frac{X_a - Y_a}{\sqrt{2}} \otimes \frac{X_b - Y_b}{\sqrt{2}}\right)},$$

takes ρ_{PROD} to ρ', a tensor product of a two-qubit state ρ_{ab} and a single-qubit state ρ_c. The parameter θ sets the entanglement for ρ_{ab}: when $\theta = 0$, ρ_{ab} is a product state; when $\theta = \pi/4$, ρ_{ab} is fully rotated into the EPR state. As such, we view this approach as smoothly interpolating between ρ_{PROD} and ρ_{MATCH} in superposition. It can be easily verified that the energy obtained by ρ' on edge (a,b) is $\mathrm{Tr}\left[h_{ab}^{EPR}\rho'\right] = 1 + 2\cos\theta\sin\theta$, and the single qubit marginals of a and b are given by

$$\rho'_a \stackrel{\text{def}}{=} \mathrm{Tr}_{bc}[\rho'] = \cos 2\theta\rho_0 + 2\sin^2\theta I = \rho'_b\,.$$

In particular, note that the single qubit marginals are simply rescaled and shifted by the identity. This fact aids in the analysis: we can still evaluate the energy of (b,c) in terms of ρ_{PROD}, even though the edge is not in our matching:

$$\begin{aligned}
\mathrm{Tr}\left[h_{bc}^{EPR}\rho'\right] &= \mathrm{Tr}\left[h_{bc}^{EPR}\left(\rho'_b \otimes \rho'_c\right)\right] \\
&= \mathrm{Tr}\left[h_{bc}^{EPR}\left(\rho'_b \otimes \rho_0\right)\right] \\
&= \mathrm{Tr}\left[h_{bc}^{EPR}\left(\left(\cos 2\theta\rho_0 + 2\sin^2\theta\ (I/2)\right) \otimes \rho_0\right)\right] \\
&= \mathrm{Tr}\left[h_{bc}^{EPR}\left(\cos 2\theta\rho_{PROD} + \sin^2\theta\, I \otimes \rho_0\right)\right] \\
&= \cos 2\theta + \sin^2\theta = \cos^2\theta\,.
\end{aligned}$$

Thus the total energy is given by

$$\mathrm{Tr}\left[(h_{ab}^{EPR} + h_{bc}^{EPR})\rho'\right] = 1 + 2\cos\theta\sin\theta + \cos^2\theta\,.$$

The parameter θ can be optimized over; in this example, taking $\theta \approx .554$ yields

$$\mathrm{Tr}\left[(h_{ab}^{EPR} + h_{bc}^{EPR})\rho'\right] \approx 2.618,$$

outperforming both ρ_{PROD} and ρ_{MATCH}.

In this example, the analysis simplifies nicely because the initial product state is the symmetric state $\rho_0^{\otimes n}$, and because the graph is bipartite. For EPR on general graphs, we provide a circuit-based algorithm that prepares an entangled state based on a *fractional* matching in the graph. This state still works by *partially* entangling edges according to the

matching while preserving single-qubit marginals, but it does not have a simple interpretation in terms of ρ_{MATCH}. We leave open the possibility that a $(\frac{1+\sqrt{5}}{4})$-approximation algorithm can be achieved with a tensor product of one and two-qubit states that simply interpolates between ρ_{PROD} and ρ_{MATCH}. For QMC, our algorithm does indeed interpolate between ρ_{PROD} and ρ_{MATCH}. However the algorithm and analysis become slightly more complicated. Crucial to our analysis is the following lemma:

▶ **Lemma 5** (Energy obtained by QMC algorithm on matched and unmatched edges). *Given two 1-qubit pure states ρ_i, ρ_j and a real parameter $\theta \in [0, \pi/2]$, there exists a 2-qubit pure state ρ_{ij} such that*

$$\begin{aligned}
\mathrm{Tr}\big[h^{QMC}\rho_{ij}\big] &= \frac{(1+\sin\theta)\left(1-\vec{B}(\rho_i)\cdot\vec{B}(\rho_j)\right)}{2}\,,\\
\vec{B}(\mathrm{Tr}_j[\rho_{ij}]) &= \cos\theta\cdot\vec{B}(\rho_i)\,,\\
\vec{B}(\mathrm{Tr}_i[\rho_{ij}]) &= \cos\theta\cdot\vec{B}(\rho_j)\,.
\end{aligned}$$

The last two constraints in Lemma 5 indicate that the Bloch vectors of the single-qubit marginals of ρ_{ij} are rescaled Bloch vectors of ρ_i and ρ_j, respectively. This is analogous to the argument in our example, where the marginals are rescaled and shifted by the identity. This again allows us to compute the energy of unmatched edges in terms of ρ_{PROD}. The parameter θ again sets the entanglement in ρ_{ij}: when $\theta = 0$, ρ_{ij} is a product state; when $\theta = \pi/2$, ρ_{ij} is maximally entangled. The proof of Lemma 5 is given in the appendix of [2].

3.1 EPR

Using the intuition from our example, we introduce the following algorithm for EPR:

▶ **Algorithm 1** (Fractional matching algorithm for EPR). *Given a weighted graph $G(V, E, w)$:*

1. *Define the 1-qubit Hamiltonian*

$$P \stackrel{\text{def}}{=} \frac{1}{\sqrt{2}}(X - Y)\,,$$

and the angle

$$\theta \stackrel{\text{def}}{=} \frac{\ln(\varphi)}{2} \approx 0.240\,, \tag{1}$$

where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

2. *Find a fractional matching $(m_{uv})_{(u,v)\in E}$ of maximum weight (e.g., via linear programming).*
3. *Output the state*

$$|\chi\rangle \stackrel{\text{def}}{=} \prod_{(u,v)\in E} e^{i\gamma_{uv}P_uP_v}\,|0\rangle^{\otimes n}\,, \tag{2}$$

where

$$\gamma_{uv} = \frac{1}{2}\cos^{-1}\exp\left[-\theta\cdot m_{uv}\right] \;\in\; [0, \pi/2]\,. \tag{3}$$

The unitary $e^{i\gamma P_uP_v}$ rotates the state $|00\rangle_{ij}$ towards the Bell state $|\phi^+\rangle$; this circuit was proposed in the approximation algorithm of [12].

3.2 QMC

Our algorithm for QMC relies on the following subroutine, which we call PMATCH since it places partially entangled 2-qubit states on the edges of a matching, while MATCH places maximally entangled states:

▶ **Algorithm 2** (PMATCH). *Given a weighted graph $G(V, E, w)$ and a real parameter $\theta \in [0, \pi/2]$:*

1. *Prepare the state $\rho_{PROD} = \bigotimes_{i\in[n]} \rho_i$ by running the* PROD *algorithm on QMC from Section 2 for some fixed, constant SDP level ℓ.*
2. *For each edge $(i, j) \in E$, compute $t_{ij} \stackrel{\text{def}}{=} \vec{B}(\rho_i) \cdot \vec{B}(\rho_j)$ from ρ_{PROD}.*
3. *Find a maximum-weight integral matching $\widetilde{M} \subseteq E$ on the* reweighted *graph $G(V, E', \widetilde{w})$, where*

$$\widetilde{w}_{ij} = w_{ij} \cdot \left(\frac{\sin\theta \left(1 - t_{ij}\left(1 + \sin\theta\right)\right)}{2} \right)^{+}, \tag{4}$$

   and $E' = \{(i, j) \in E \mid \widetilde{w}_{ij} > 0\}$. Here, we use the notation $(\cdot)^+ \stackrel{\text{def}}{=} \max(0, \cdot)$ from [14].
4. *Let $\rho_{PMATCH}(\theta)$ be the state starting from ρ_{PROD}, but for each edge (i, j) in $\widetilde{M}$, replace $\rho_i \otimes \rho_j$ with the state ρ_{ij} described in Lemma 5, parametrized by ρ_i, ρ_j, θ.*
5. *Output the state $\rho_{PMATCH}(\theta)$.*

Our algorithm for QMC chooses the better of the algorithms PMATCH and MATCH:

▶ **Algorithm 3** (Combined algorithm for QMC). *Given a weighted graph $G(V, E, w)$:*

1. *Prepare the state ρ_{MATCH} by running the* MATCH *algorithm from Section 2.*
2. *Prepare the state $\rho_{PMATCH}(\theta)$ by running* PMATCH *with parameter $\theta = 1.286$.*
3. *Output the state out of ρ_{MATCH} and $\rho_{PMATCH}(\theta)$ obtaining larger energy on the QMC Hamiltonian H_G.*

4 Analysis

We upper-bound the maximum energy $\lambda_{max}(H_G)$ of QMC and EPR Hamiltonians using a quantifiable *monogamy of entanglement*:

▶ **Lemma 6** (Monogamy of Entanglement, e.g. [14, Lemma 1]). *For both EPR and QMC, and for all graphs $G(V, E, w)$, we have*

$$\lambda_{max}(H_G) \leq W_G + \mathrm{FM}_G\,.$$

For QMC, our analysis improves with better upper bounds on $\lambda_{max}(H_G)$. We use an inequality of the form $W_G + \mathrm{M}_G/d$; this first appears in [14, Lemma 4] with the constant $d = 4/5$. By a detailed analysis and numerical verification on graphs with up to 13 vertices, this constant was recently improved to $14/15$ in [8]:

▶ **Lemma 7** (Strengthened monogamy of entanglement, [8, Lemma 3.10]). *For QMC, and for all graphs $G(V, E, w)$, we have*

$$\lambda_{max}(H_G) \leq W_G + \frac{\mathrm{M}_G}{d}\,,$$

where $d = \frac{14}{15}$. Additionally, the bound holds for the optimal value of the 13-th level of the quantum moment SoS hierarchy.

4.1 EPR

We now prove Theorem 1. Given the state in Equation (2)

$$|\chi\rangle \stackrel{\text{def}}{=} \prod_{(i,j)\in E} e^{i\gamma_{ij} P_i P_j} |0\rangle^{\otimes n} , \tag{5}$$

[12, Lemma 9] showed that the energy on the local term h_{ij}^{EPR} is at least

$$\langle\chi| h_{ij}^{EPR} |\chi\rangle \geq \frac{1}{2} \left(1 + A_{ij} B_{ij} + \sin 2\gamma_{ij} \cdot (A_{ij} + B_{ij})\right) , \tag{6}$$

where

$$A_{ij} \stackrel{\text{def}}{=} \prod_{k\in N(i)\setminus\{j\}} \cos 2\gamma_{ik} ,$$
$$B_{ij} \stackrel{\text{def}}{=} \prod_{k\in N(j)\setminus\{i\}} \cos 2\gamma_{jk} .$$

In Algorithm 1, the output state $|\chi\rangle$ is in the form of Equation (5). Using the angles γ_{ij} specified by Equation (3), we have

$$A_{ij} = \exp\left[-\theta \sum_{k\in N(i)\setminus\{j\}} m_{ik}\right] \geq \exp\left[-\theta(1 - m_{ij})\right] , \tag{7}$$

$$B_{ij} = \exp\left[-\theta \sum_{k\in N(j)\setminus\{i\}} m_{jk}\right] \geq \exp\left[-\theta(1 - m_{ij})\right] , \tag{8}$$

$$\sin 2\gamma_{ij} = \sqrt{1 - \cos^2 2\gamma_{ij}} = \sqrt{1 - \exp\left[-2\theta m_{ij}\right]} . \tag{9}$$

The inequalities in the first two lines follow because m is a matching. For example, we have $m_{ij} + \sum_{k\in N(i)\setminus\{j\}} m_{ik} \leq 1$. Using Equation (6), the energy of $|\chi\rangle$ on h_{ij}^{QMC} is at least

$$T(\theta, m_{ij}) \stackrel{\text{def}}{=} \frac{1}{2}\left(1 + \exp\left[-2\theta(1 - m_{ij})\right] + 2\sqrt{1 - \exp\left[-2\theta m_{ij}\right]} \exp\left[-\theta(1 - m_{ij})\right]\right) . \tag{10}$$

We combine Equation (10) with Lemma 6 to bound the approximation ratio on any graph $G(V, E, w)$:

$$\frac{\langle\chi| H_G |\chi\rangle}{\lambda_{max}(H_G)} \geq \frac{\sum_{(i,j)\in E} w_{ij} \cdot T(\theta, m_{ij})}{\sum_{(i,j)\in E} w_{ij} (1 + m_{ij})} .$$

Each term in the numerator and denominator is positive, so the approximation ratio is at least the approximation ratio of the worst edge

$$\min_{(i,j)\in E} \frac{T(\theta, m_{ij})}{1 + m_{ij}} \geq \min_{x\in[0,1]} \frac{T(\theta, x)}{1 + x} . \tag{11}$$

Recall from Equation (1) that we take $\theta = \frac{1}{2}\ln\varphi$. We empirically identified θ as the angle that maximizes the approximation ratio in this analysis. Substituting this value into $T(\theta, x)$ in Equation (10), the RHS of Equation (11) yields the minimization problem

$$\min_{x\in[0,1]} \left[\frac{1}{2(1+x)}\left(1 + \varphi^{-(1-x)} + 2\sqrt{1 - \varphi^{-x}} \cdot \varphi^{-\frac{1}{2}(1-x)}\right)\right],$$

whose value is found to be $\varphi/2$ by the following lemma:

▶ **Lemma 8.** *Given* $\varphi \stackrel{\text{def}}{=} \frac{1+\sqrt{5}}{2}$*, we have that*

$$\min_{x\in[0,1]} \left[\frac{1}{2(1+x)}\left(1+\varphi^{-(1-x)}+2\sqrt{1-\varphi^{-x}}\cdot\varphi^{-\frac{1}{2}(1-x)}\right)\right] = \frac{1+\sqrt{5}}{4} = \frac{\varphi}{2}\,. \tag{12}$$

The proof of Lemma 8 is given in the appendix of [2].

4.2 QMC

We now analyze Algorithm 3 and prove the 0.611-approximation in Theorem 2. We first introduce some helpful notation. Fix a graph $G(V, E, w)$. Let $\tilde{\rho}$ be the pseudo-density matrix outputted by solving the SDP of the level-ℓ quantum moment-SOS hierarchy (described in Section 2.3). We define

$$g_{ij} \stackrel{\text{def}}{=} \mathrm{Tr}[h_{ij}^{QMC}\tilde{\rho}]\,, \qquad s_{ij} \stackrel{\text{def}}{=} \frac{1}{3}\mathrm{Tr}[(X_iX_j + Y_iY_j + Z_iZ_j)\tilde{\rho}]\,.$$

The first value is the energy obtained by $\tilde{\rho}$ on h_{ij}^{QMC}; the second value is the expected value of $\tilde{\rho}$ with respect to the traceless components of the term h_{ij}^{QMC}. It is straightforward to show that $g_{ij} = \frac{1}{2}(1-3s_{ij})$. The output of the SDP gives an upper bound to the optimal energy:

$$\lambda_{max}(H_G) \leq \sum_{(i,j)\in E} w_{ij}\cdot g_{ij} = \sum_{(i,j)\in E} w_{ij}\cdot\frac{1-3s_{ij}}{2}\,. \tag{13}$$

Fix a positive integer k and suppose Lemma 7 is true for $d = \frac{2k}{2k+1}$. Then, consider a *reweighting* $\widetilde{G}(V, E, u)$ of the graph G where $u_{ij} \geq 0$. Let $x \in \mathbb{R}^E$ be a vector with

$$x_{ij} \stackrel{\text{def}}{=} d\,(g_{ij}-1)^+ = d\left(-\frac{1+3s_{ij}}{2}\right)^+.$$

Here, we again use the notation $(\cdot)^+ \stackrel{\text{def}}{=} \max(\,\cdot\,, 0)$ from [14], which ensures

$$\left(-\frac{1+3s_{ij}}{2}\right)^+ \in [0,1]\,, \qquad \forall (i,j)\in E.$$

It is shown (e.g. in [8, Lemma 3.4]) that x is in the *integral* matching polytope of $\widetilde{G}$. This means that x is a convex combination of integral matchings $\{m_\ell\}_{1\leq\ell\leq k}$. Since the weight of any integral matching m_ℓ on $\widetilde{G}$ is at most $\mathrm{M}_{\widetilde{G}}$, the weight of x on $\widetilde{G}$ is also at most $\mathrm{M}_{\widetilde{G}}$. Thus,

$$\sum_{(i,j)\in E} u_{ij}\cdot d\cdot\left(-\frac{1+3s_{ij}}{2}\right)^+ \leq \mathrm{M}_{\widetilde{G}}\,. \tag{14}$$

Equation (14) in this context was first used in [16] and also appears in [14, Lemma 4] and [10, Lemma C.2].

We now find a lower bound for the energy obtained by the matching state $\rho_{PMATCH}(\theta)$ in PMATCH. We consider the energy on a case-by-case basis in the following lemma:

▶ **Lemma 9.** *Consider the state ρ_{PMATCH} and matching $\widetilde{M}$ from Algorithm 2. For $x \in \{0,1,2\}$, let $\widetilde{S}_x$ be the subset of edges not in $\widetilde{M}$ (e.g. $\widetilde{S}_x \subseteq E \setminus \widetilde{M}$) where each edge in $\widetilde{S}_x$ has x matched endpoints in $\widetilde{M}$. Then the energy of ρ_{PMATCH} on an edge (i,j) is exactly*

$$\mathrm{Tr}\left[h_{ij}^{QMC}\rho_{PMATCH}\right] = \begin{cases} \frac{1}{2}\left(1+\sin\theta\right)\left(1-t_{ij}\right), & (i,j)\in\widetilde{M}, \\ \frac{1}{2}(1-t_{ij}), & (i,j)\in\widetilde{S}_0, \\ \frac{1}{2}\left(1-t_{ij}\cos\theta\right), & (i,j)\in\widetilde{S}_1, \\ \frac{1}{2}\left(1-t_{ij}\cos^2\theta\right), & (i,j)\in\widetilde{S}_2. \end{cases}$$

where, t_{ij} is defined in Algorithm 2.

Proof. The case when $(i,j)\in\widetilde{M}$ follows directly from how we choose ρ_{PMATCH} in Algorithm 2, Step 4. For all other edges, recall that t_{ij} is the inner product of the Bloch vectors of qubits i and j with respect to the product state ρ_{PROD}. The reduced density matrix of ρ_{PMATCH} on qubits i and j is a product state $\sigma_i\otimes\sigma_j$. By [12, Lemma 10], the energy of $\sigma_i\otimes\sigma_j$ on h_{ij}^{QMC} is

$$\frac{1}{2}\left(1-\vec{B}(\sigma_i)\cdot\vec{B}(\sigma_j)\right).$$

We handle each remaining case separately:

1. $(i,j)\in\widetilde{S}_0$: Here, the product state is exactly $\rho_i\otimes\rho_j$, and so $\vec{B}(\sigma_i)\cdot\vec{B}(\sigma_j) = t_{ij}$ by definition.[1]
2. $(i,j)\in\widetilde{S}_1$: Without loss of generality, i belongs to a matched edge in $\widetilde{M}$ and j is not in a matched edge. Lemma 5 implies that $\vec{B}(\sigma_i)$ is rescaled to $\vec{B}(\sigma_i)\cos\theta$, and so $\vec{B}(\sigma_i)\cdot\vec{B}(\sigma_j) = t_{ij}\cos\theta$.
3. $(i,j)\in\widetilde{S}_2$: i and j belong to two different matched edges. Therefore, it has *both* Bloch vectors rescaled, i.e. $\vec{B}(\sigma_i)\cdot\vec{B}(\sigma_j) = \left(\vec{B}(\rho_i)\cdot\vec{B}(\rho_j)\right)\cos^2\theta = t_{ij}\cos^2\theta$. ◀

Lemma 9 allows us to compute the energy obtained by the state $\rho_{PMATCH}(\theta)$:

$$\begin{aligned}
&\mathrm{PMATCH}(\theta) \stackrel{\text{def}}{=} \sum_{(i,j)\in E} w_{ij}\cdot\mathrm{Tr}[h_{ij}^{QMC}\rho_{PMATCH}] \\
&= \sum_{(i,j)\in\widetilde{M}} \frac{w_{ij}}{2}\left(1+\sin\theta\right)\left(1-t_{ij}\right) + \sum_{(i,j)\in\widetilde{S}_0}\frac{w_{ij}}{2}\left(1-t_{ij}\right) \\
&\qquad + \sum_{(i,j)\in\widetilde{S}_1}\frac{w_{ij}}{2}\left(1-t_{ij}\cos\theta\right) + \sum_{(i,j)\in\widetilde{S}_2}\frac{w_{ij}}{2}\left(1-t_{ij}\cos^2\theta\right) \\
&= \sum_{(i,j)\in E}\frac{w_{ij}}{2}\left(1-t_{ij}\cos^2\theta\right) + \sum_{(i,j)\in\widetilde{M}}\frac{w_{ij}}{2}\left(\left(1+\sin\theta\right)\left(1-t_{ij}\right)-\left(1-t_{ij}\cos^2\theta\right)\right) \\
&\qquad + \sum_{(i,j)\in\widetilde{S}_0}\frac{w_{ij}t_{ij}}{2}\left(\cos^2\theta-1\right) + \sum_{(i,j)\in\widetilde{S}_1}\frac{w_{ij}t_{ij}}{2}\left(\cos^2\theta-\cos\theta\right) \\
&= \sum_{(i,j)\in E}\frac{w_{ij}}{2}\left(1-t_{ij}\cos^2\theta\right) + \sum_{(i,j)\in\widetilde{M}}\frac{w_{ij}}{2}\left(\sin\theta\left(1-t_{ij}\left(1+\sin\theta\right)\right)\right) \\
&\qquad - \sum_{(i,j)\in\widetilde{S}_0}\frac{w_{ij}t_{ij}}{2}\sin^2\theta + \sum_{(i,j)\in\widetilde{S}_1}\frac{w_{ij}t_{ij}}{2}\left(\cos^2\theta-\cos\theta\right). \qquad (15)
\end{aligned}$$

[1] Although $\widetilde{M}$ is a maximum matching, its edge set E' is a subset of E, and so $\widetilde{S}_0$ may be non-empty.

Note that $\widetilde{M}$ is a maximum-weight *integral* matching with respect to the rescaled weights defined in Equation (4). Thus, we can invoke Equation (14) to replace the sum over $\widetilde{M}$ in Equation (15) with a sum over E. Since the rescaled weights are positive for all edges in $\widetilde{M}$, we have

$$\begin{aligned}\mathrm{PMATCH}(\theta) \geq \sum_{(i,j)\in E} \frac{w_{ij}}{2}\left(1 - t_{ij}\cos^2\theta + d\big(\sin\theta\,(1 - t_{ij}\,(1+\sin\theta))\big)\left(-\frac{1+3s_{ij}}{2}\right)^+\right) \\ - \sum_{(i,j)\in\widetilde{S}_0} \frac{w_{ij}t_{ij}}{2}\sin^2\theta + \sum_{(i,j)\in\widetilde{S}_1} \frac{w_{ij}t_{ij}}{2}\left(\cos^2\theta - \cos\theta\right). \end{aligned} \tag{16}$$

Note that t_{ij} is a random variable because PROD is a randomized algorithm. The analysis of PROD in [7] is based on [4], which shows that $\mathbb{E}[t_{ij}] = F(s_{ij})$. A definition of F in the context of QMC appears in [17, Equation 3]. For our purposes, it is enough to note that F is an odd function because it has the form

$$F(s) = c \cdot s \cdot G(s^2)\,, \tag{17}$$

where $c > 0$ is a constant and $G \geq 0$ is a hypergeometric function. Since Equation (16) is linear in t_{ij}, we use linearity of expectation to conclude

$$\begin{aligned}&\mathbb{E}[\mathrm{PMATCH}(\theta)] \geq \\ &\sum_{(i,j)\in E} \frac{w_{ij}}{2}\left(1 - F(s_{ij})\cos^2\theta + d\sin\theta\,(1 - F(s_{ij})\,(1+\sin\theta))\left(-\frac{1+3s_{ij}}{2}\right)^+\right) \\ &\quad - \sum_{(i,j)\in\widetilde{S}_0} \frac{w_{ij}}{2}F(s_{ij})\sin^2\theta + \sum_{(i,j)\in\widetilde{S}_1} \frac{w_{ij}}{2}\Big(\big(\cos^2\theta - \cos\theta\big)\;F(s_{ij})\Big). \end{aligned} \tag{18}$$

The following lemma lets us analyze the performance of Algorithm 3, which takes the better of two algorithms, MATCH and PMATCH.

▶ **Lemma 10** (Reducing worst-case bounds to a single edge). *Suppose we have a collection of k approximation algorithms $\{A_\ell\}_{\ell\in[k]}$ and an upper bound on the maximum energy $\lambda_{max}(H_G) \leq \sum_{e\in E} w_e b_e$, where $b_e \in B = (0, b_{max}]$ for all $e \in E$, and b_{max} is a constant. Furthermore, suppose that the energy that algorithm A_ℓ earns on edge e is a function of b_e, which we denote $a_\ell(b_e)$, such that A_ℓ earns $\sum_{e\in E} w_e\, a_\ell(b_e)$. Then, by running each of $A_{\ell\in[k]}$ and taking the output with the maximum energy, we can obtain an approximation ratio*

$$\alpha \geq \max_{\mu_\ell}\ \min_{b} \sum_{\ell} \mu_\ell \frac{a_\ell(b)}{b},$$

where the maximum is taken over valid probability distributions

$$\{\ \mu \mid \sum_{\ell\in[k]} \mu_\ell = 1,\ 0 \leq \mu_\ell \leq 1\ \forall \ell \in [k]\ \}\,,$$

and the minimum is taken over $b \in B$.

Proof. Given a specific graph G, outputting the best of A_ℓ is at least as good as outputting the result of A_ℓ with probability μ_ℓ, for all $\ell \in [k]$ and for any possible μ. Now, given a fixed distribution μ, the approximation ratio is at least

$$\begin{aligned}
\alpha &\geq \sum_\ell \mu_\ell \frac{\sum_{e\in E} w_e\, a_\ell(b_e)}{\sum_{e\in E} w_e b_e} \\
&= \sum_\ell \mu_\ell \sum_{e\in E} \left(\frac{w_e b_e}{\sum_{e\in E} w_e b_e}\right) \frac{a_\ell(b_e)}{b_e} \\
&\geq \min_{e\in E} \sum_\ell \mu_\ell \frac{a_\ell(b_e)}{b_e} \\
&\geq \min_{b\in B} \sum_\ell \mu_\ell \frac{a_\ell(b)}{b}\,.
\end{aligned}$$

The third line follows because

$$\left\{\frac{w_e b_e}{\sum_{e\in E} w_e b_e}\right\}_{e\in E}$$

is a distribution over E. Since the statement is true for all possible distributions μ, we take the maximum over μ to finish the lemma. ◀

We apply Lemma 10 with the SDP upper bound in Equation (13) and our two algorithms MATCH and PMATCH as described in Algorithm 3. For PMATCH, we consider the minimum of three different functions, depending on if an edge is in $\widetilde{S}_0$, in $\widetilde{S}_1$, or otherwise. For MATCH, we derived in Section 2.3 that the energy obtained by MATCH on G is $\frac{3\mathrm{M}+W}{2}$. Since M is a maximum-weight matching with respect to G, we may invoke Equation (14) to express

$$\mathrm{M} \geq \sum_{(i,j)\in E} w_{ij} \cdot d \cdot \left(-\frac{1+3s_{ij}}{2}\right)^+,$$

implying that MATCH achieves energy at least

$$\frac{3\mathrm{M}+W}{2} \geq \sum_{(i,j)\in E} \frac{w_{ij}}{2}\left(1+3d\left(-\frac{1+3s_{ij}}{2}\right)^+\right).$$

We have lower-bounded the energy that each algorithm earns on an edge (i,j) as a function of s_{ij}. Note that the SDP upper bound is trivial (i.e. at most 0) when $s_{ij} \geq \frac{1}{3}$, so we may search over $s \in [-1, 1/3)$. Altogether, Lemma 10 implies that Algorithm 3 obtains approximation ratio at least

$$\begin{aligned}
\alpha \geq \max_{\mu\in[0,1],\,\theta\in[0,\pi/2]} \min\Big\{ &\min_{s\in[-1,1/3)} \ \mu\, E_{PM}(\theta) + (1-\mu)E_M, \\
&\min_{s\in[-1,1/3)} \ \mu\, E'_{PM}(\theta) + (1-\mu)E_M, \\
&\min_{s\in[-1,1/3)} \ \mu\, E''_{PM}(\theta) + (1-\mu)E_M \Big\},
\end{aligned} \tag{19}$$

$$E_{PM}(\theta) \stackrel{\text{def}}{=} \frac{1 - F(s)\cos^2\theta + d\sin\theta\,(1 - F(s)\,(1+\sin\theta)) \cdot \left(-\frac{1+3s}{2}\right)^+}{1-3s}, \tag{20}$$

$$E'_{PM}(\theta) \stackrel{\text{def}}{=} \frac{1 - F(s)\cos\theta + d\sin\theta\,(1 - F(s)\,(1+\sin\theta)) \cdot \left(-\frac{1+3s}{2}\right)^+}{1-3s}, \tag{21}$$

$$E''_{PM}(\theta) \stackrel{\text{def}}{=} \frac{1 - F(s) + d\sin\theta\,(1 - F(s)\,(1+\sin\theta)) \cdot \left(-\frac{1+3s}{2}\right)^+}{1-3s}, \tag{22}$$

$$E_M \stackrel{\text{def}}{=} \frac{1 + 3\,d\,(-\frac{1+3s}{2})^+}{1-3s}. \tag{23}$$

In the above, μ is demoted to a scalar (which fully describes a probability distribution over two variables). Equation (23) is the approximation ratio obtained by MATCH on an edge with $s = s_{ij}$. The expected approximation ratio obtained by PMATCH is Equation (22) on an edge in $\widetilde{S}_0$, Equation (21) on an edge in $\widetilde{S}_1$, and is Equation (20) on all other edges.

We further simplify Equation (19). Recall that F is an odd function (Equation (17)). So, for all $\theta \in [0, \pi/2]$ we have $E''_{PM}(\theta) \le E'_{PM}(\theta) \le E_{PM}(\theta)$ when $s > 0$ and $E_{PM}(\theta) \le E'_{PM}(\theta) \le E''_{PM}(\theta)$ when $s \le 0$. So we may rewrite Equation (19) as

$$\alpha \ge \max_{\mu\in[0,1],\,\theta\in[0,\pi/2]} \min\left\{ \min_{s\in[-1,0)} \mu\,E_{PM}(\theta) + (1-\mu)E_M,\ \min_{s\in[0,1/3)} \mu\,E''_{PM}(\theta) + (1-\mu)E_M \right\}. \tag{24}$$

One may then search over the three free parameters μ, θ, and s using an enumeration of the feasible ranges. For a more efficient approach, it is also possible to solve a linear program where the number of variables and constraints is based on an enumeration of the feasible ranges for s and θ. We obtain $\alpha > 0.611$ for $d = 14/15$ at $\theta = 1.286$ and $\mu = 0.861$. By Lemma 7, our algorithm thus achieves a 0.611-approximation in expectation if $\ell = 13$. If the factor d in Lemma 7 is improved to 1, we would obtain $\alpha > 0.614$ at $\theta = 1.288$ and $\mu = 0.881$.

▶ Remark 11. We may simplify Equation (19) by assuming $s \in [-1, 0)$. This is because if $s \in [0, 1/3)$ then $E_M \ge 1$ and $E''_{PM}(\theta) \ge 1$ for all $\theta \in [0, \pi/2]$. This follows from F being odd, and in addition $|F(s)| \le |s|$ for all $s \in [-1, 1]$.

5 Discussion

In this work, we introduce two novel techniques for constructing algorithms for the Quantum MaxCut and EPR Hamiltonians on arbitrary graphs. The main techniques are a) *partially* entangling edges in a matching towards an entangled state while maintaining the direction of the marginal Bloch vectors as b) choosing matching with respect to *reweighted* graphs. We show that these techniques immediately lead to state-of-the-art approximation ratios for both problems. We believe that these techniques may be combined with other methods to obtain further improvements, and encourage further studies in this direction.

References

1 Anurag Anshu, David Gosset, and Karen Morenz. Beyond product state approximations for a quantum analogue of Max Cut. *LIPIcs, Volume 158, TQC 2020*, 158:7:1–7:15, 2020. doi:10.4230/LIPIcs.TQC.2020.7.

2 Anuj Apte, Eunou Lee, Kunal Marwaha, Ojas Parekh, and James Sud. Improved algorithms for quantum maxcut via partially entangled matchings, 2025. arXiv:2504.15276.

3 Fernando G. S. L. Brandão and Aram W. Harrow. Product-state Approximations to Quantum Ground States, December 2014. arXiv:1310.0017.

4 Jop Briet, Fernando Mario de Oliveira Filho, and Frank Vallentin. The positive semidefinite Grothendieck problem with rank constraint, 2010. doi:10.1007/978-3-642-14165-2_4.

5 Toby Cubitt and Ashley Montanaro. Complexity classification of local Hamiltonian problems, March 2016. arXiv:1311.3161.

6 Jack Edmonds. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, 17:449–467, January 1965. doi:10.4153/CJM-1965-045-4.

7 Sevag Gharibian and Ojas Parekh. Almost optimal classical approximation algorithms for a quantum generalization of Max-Cut. *LIPIcs, Volume 145, APPROX/RANDOM 2019*, 145:31:1–31:17, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.31.

8 Sander Gribling, Lennart Sinjorgo, and Renata Sotirov. Improved approximation ratios for the Quantum Max-Cut problem on general, triangle-free and bipartite graphs, April 2025. arXiv:2504.11120.

9 Felix Huber, Kevin Thompson, Ojas Parekh, and Sevag Gharibian. Second order cone relaxations for quantum Max Cut, November 2024. arXiv:2411.04120.

10 Zackary Jorquera, Alexandra Kolla, Steven Kordonowy, Juspreet Singh Sandhu, and Stuart Wayland. Monogamy of Entanglement Bounds and Improved Approximation Algorithms for Qudit Hamiltonians, November 2024. arXiv:2410.15544.

11 Nathan Ju and Ansh Nagda. Improved approximation algorithms for the EPR Hamiltonian, April 2025. doi:10.48550/arXiv.2504.10712.

12 Robbie King. An Improved Approximation Algorithm for Quantum Max-Cut. *Quantum*, 7:1180, November 2023. doi:10.22331/q-2023-11-09-1180.

13 Eunou Lee. Optimizing quantum circuit parameters via SDP, September 2022. arXiv:2209.00789.

14 Eunou Lee and Ojas Parekh. An improved Quantum Max Cut approximation via matching, February 2024. arXiv:2401.03616.

15 Miguel Navascues, Stefano Pironio, and Antonio Acin. A convergent hierarchy of semidefinite programs characterizing the set of quantum correlations. *New Journal of Physics*, 10(7):073013, July 2008. doi:10.1088/1367-2630/10/7/073013.

16 Ojas Parekh and Kevin Thompson. Application of the Level-2 Quantum Lasserre Hierarchy in Quantum Approximation Algorithms. *LIPIcs, Volume 198, ICALP 2021*, 198:102:1–102:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.102.

17 Ojas Parekh and Kevin Thompson. An Optimal Product-State Approximation for 2-Local Quantum Hamiltonians with Positive Terms, June 2022. doi:10.48550/arXiv.2206.08342.

18 Stephen Piddock and Ashley Montanaro. The complexity of antiferromagnetic interactions and 2D lattices, December 2015. arXiv:1506.04014.

19 Jun Takahashi, Chaithanya Rayudu, Cunlu Zhou, Robbie King, Kevin Thompson, and Ojas Parekh. An SU(2)-symmetric Semidefinite Programming Hierarchy for Quantum Max Cut, August 2023. arXiv:2307.15688.

20 Adam Bene Watts, Anirban Chowdhury, Aidan Epperly, J. William Helton, and Igor Klep. Relaxations and Exact Solutions to Quantum Max Cut via the Algebraic Structure of Swap Operators. *Quantum*, 8:1352, May 2024. doi:10.22331/q-2024-05-22-1352.

A Combinatorial Proof of Universal Optimality for Computing a Planar Convex Hull

Ivor van der Hoog
IT University of Copenhagen, Denmark

Eva Rotenberg
IT University of Copenhagen, Denmark

Daniel Rutschmann
IT University of Copenhagen, Denmark

Abstract

For a planar point set P, its convex hull is the smallest convex polygon that encloses all points in P. The construction of the convex hull from an array I_P containing P is a fundamental problem in computational geometry. By sorting I_P in lexicographical order, one can construct the convex hull of P in $O(n \log n)$ time which is worst-case optimal. Standard worst-case analysis, however, has been criticized as overly coarse or pessimistic, and researchers search for more refined analyses.

For an algorithm A, worst-case analysis fixes n, and considers the maximum running time of A across all size-n point sets P and permutations I_P of P. Output-sensitive analysis fixes n and k, and considers the maximum running time across all size-n points sets P with k hull points and permutations I_P of P. Universal analysis provides an even stronger guarantee. It fixes a point set P and considers the maximum running time across all permutations I_P of P.

Kirkpatrick, McQueen, and Seidel [SICOMP'86] consider output-sensitive analysis. If the convex hull of P contains k points, then their algorithm runs in $O(n \log k)$ time. Afshani, Barbay, Chan [FOCS'07] prove that the algorithm by Kirkpatrick, McQueen, and Seidel is also universally optimal. Their proof restricts the model of computation to any algebraic decision tree model where the test functions have at most constant degree and at most a constant number of arguments. They rely upon involved algebraic arguments to construct a lower bound for each point set P that matches the universal running time of [SICOMP'86].

We provide a different proof of universal optimality. Instead of restricting the computational model, we further specify the output. We require as output (1) the convex hull, and (2) for each internal point of P a witness for it being internal. Our argument is shorter, perhaps simpler, and applicable in more general models of computation.

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

Keywords and phrases Convex hull, Combinatorial proofs, Universal optimality

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.102

Funding This work was supported by the Carlsberg Foundation Fellowship CF21-0302 "Graph Algorithms with Geometric Applications", the VILLUM Foundation grant (VIL37507) "Efficient Recomputations for Changeful Problems", and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987.

1 Introduction

Convex hulls are a central topic in computational geometry. By sorting a point set P in lexicographical order, one can construct its convex hull in $O(n \log n)$ time, which is worst-case optimal. Traditional worst-case analysis often faces criticism for being overly coarse or pessimistic. To address these concerns, researchers introduced more nuanced complexity measures that account not only for the input size but also for additional parameters that


© Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann;
licensed under Creative Commons License CC-BY 4.0



capture the difficulty of the instance. A classical example is output-sensitive analysis, where the running time analysis depends on the output size [7]. Other algorithms measure their performance by the spread of the point set, defined as the ratio of the maximum to minimum pairwise distances among the points [4]. More examples include algorithmic complexity parametrized on various geometric characteristics of P, such as the ratio of circumradii to inradii or the number of reflex angles in an input polygon [3, 8].

This paper improves upon the work by Afshani, Barbay, and Chan [1] (see also their journal version [2]). They observe that, for geometric problems, the worst-case algorithmic analysis contains a double maximum. They only consider *correct* algorithms, which we formally define in the preliminaries. For now, denote by $\mathbb{P}_n$ all point sets of size n and for any $P \in \mathbb{P}_n$ by $\mathbb{I}_P$ all size-n arrays that contain the points of P in some order. For an algorithm A, denote by $\rho(A, I_P)$ its runtime when the input is I_P. Then, for a fixed correct algorithm A and input size n, the worst-case running time is:

$$\text{worst-case}(A, n) := \max_{P \in \mathbb{P}_n} \max_{I_P \in \mathbb{I}_P} \rho(A, I_P).$$

An algorithm A is worst-case optimal if there exists no algorithm A' whose worst-case running time is asymptotically smaller than that of A. Prior works perform a finer-grained analysis by restricting the first maximum. For example, output-sensitive running time is defined as:

$$\text{output-sensitive}(A, n, k) := \max_{P \in \mathbb{P}_n \text{ and } P \text{ has } k \text{ points on the convex hull}} \max_{I_P \in \mathbb{I}_P} \rho(A, I_P).$$

Afshani, Barbay, and Chan [1] obtain an even stronger quality guarantee by eliminating the first maximum all-together. They call their notion of optimality *instance-optimality in the order-oblivious setting*. Recently, this notion has been called *universal optimality* [5, 6, 9]. For a fixed point set P, we define the *universal* running time of an algorithm A as:

$$\text{universal}(A, P) := \max_{I_P \in \mathbb{I}_P} \rho(A, I_P).$$

An algorithm A is universally optimal if for all point sets P, there exists no algorithm A' whose universal running time is asymptotically smaller than that of A. Observe that any universally optimal algorithm is automatically output-sensitive.

Contribution. Afshani, Barbay, Chan prove that the algorithm by Kirkpatrick, McQueen, and Seidel [7] is universally optimal.[1] Their proof restricts the model of computation to any algebraic decision tree model where the test functions have at most constant degree and have at most a constant number of arguments. They present an involved algebraic argument to construct a lower bound for each point set P that matches the universal running time of [7].

Here, we give a different proof. Rather than restricting the model of computation, we give a more extensive but natural specification of the output. We call a point of P *internal* if it does not appear on the boundary of the convex hull. We require that any algorithm computes (1) the convex hull and (2) for each internal point a witness for its being internal. For a point set P, we apply a simple combinatorial counting argument over all outputs to obtain a lower bound that matches the universal running time of [7]. Our argument is shorter, arguably simpler, and holds in more general models of computation.

[1] We note that the paper has only Kirkpatrick and Seidel as authors. However, a footnote remarks that McQueen suggests an adaption to the algorithm. Only with this adaption, the algorithm becomes universally optimal.

2 Preliminaries

We follow [1] and assume that the input points lie in general position. I.e., P contains no duplicates and no collinear triples. For any integer n, we denote by $\mathbb{P}_n$ all planar point sets of n points that lie in general position. We denote by $\mathbb{I}_P$ all arrays of size n that store P.

▶ **Definition 1.** *For a planar point set P, its* convex hull $\mathrm{CH}(P)$ *is the minimum convex region that contains all points in P.*

▶ **Definition 2.** *For a planar point set P, its* convex hull vertices $\mathrm{ch}(P)$ *are the points in P that lie on the boundary of* $\mathrm{CH}(P)$.

The Convex Hull Problem. Our input is an array $I_P \in \mathbb{I}_P$ of n points P in general position. Convex hull algorithms output $\mathrm{ch}(P)$ in cyclical ordering. For our analysis, we further require that the output contains, for all points in $P - \mathrm{ch}(P)$, a *witness* that they are internal:

▶ **Definition 3.** *For any $p \in P$, a triangle t (whose corners lie in P) is a* witness *of p if p is contained in the interior of t.*

▶ **Observation 4.** *A point $p \in P$ does not lie on the boundary of* $\mathrm{CH}(P)$ *if and only if there exists at least one witness of p.*

Requiring the output to contain witnesses is a natural condition as it is effectively asking the algorithm A to verify its correctness. The concept of certification and verifiability is central to the design of algorithms. Without this requirement, there are oblivious decision trees (defined below) that compute the convex hull in $O(k \log n)$ time.

Formalising output. Our output consists of two lists: A *hull list* H encodes $\mathrm{ch}(P)$ in cyclical order, and a *witness list* W defines for each point in $P - \mathrm{ch}(P)$ a witness. Formally, a *hull list* $H = (h_1, \ldots, h_k)$ of I_P is a sequence of integers such that the points $I_P[h_i]$, for $i \in [k]$, are precisely the k points on the boundary of $\mathrm{CH}(P)$ in their cyclical ordering. We require that $I_P[h_1]$ is the leftmost point of $\mathrm{ch}(P)$. A *witness list* (we refer ahead to Figure 1) of I_P is a sequence $W = (a_i, b_i, c_i)_{i=1}^{n}$ of integer triples such that:

- $a_i = b_i = c_i = -1$ if $I_P[i]$ lies on $\mathrm{CH}(P)$, and otherwise
- the triangle $(I_P[a_i], I_P[b_i], I_P[c_i])$ is a witness of $I_P[i]$.

Algorithms. We define an *algorithm* for convex hull construction as any comparison-based algorithm that receives any ordered set of planar points I_P and outputs a hull list and witness list of I_P. Given a fixed input I_P and an algorithm A, the *running time* $\rho(A, I_P)$ is the number of instructions used by A to terminate on the input I_P.

Decision trees. For $n \in \mathbb{N}$, we define an *abstract decision tree* T as a tree whose inner nodes have two children. The inner nodes contain no additional information. Its leaves contain two ordered lists. The first is an arbitrary list of integers and the second is a list of n integer triples. We define an *oblivious* decision tree T as any abstract decision tree such that for any point set $P \in \mathbb{P}_n$, and any $I_P \in \mathbb{I}_P$, there exists a root-to-leaf path in T where the two lists at the leaf are a hull list and witness list of I_P, respectively. We denote by $\mathcal{T}_n^O$ the set of all oblivious decision trees for $n \in \mathbb{N}$.

▶ **Definition 5.** *A comparison-based algorithm A is* correct *if for any input I_P it outputs a correct hull and witness list. Observe that any correct comparison-based algorithm A for convex hull construction has, for each n, a corresponding oblivious decision tree in $\mathcal{T}_n^O$.*

Worst-case optimality. Let $\mathcal{A}$ denote the set of all correct algorithms. For an algorithm $A \in \mathcal{A}$ the *worst-case running time* is defined as

$$\text{worst-case}(A, n) := \max_{P \in \mathbb{P}_n} \max_{I_P \in \mathbb{I}_P} \rho(A, I_P).$$

An algorithm is worst-case optimal if there exists a constant c such that, for all n large enough, $\text{worst-case}(A, n) \leq c \cdot \min_{A' \in \mathcal{A}} \text{worst-case}(A', n)$.

▶ **Observation 6.** *For any algorithm A and n, the worst-case running time of A is lower bounded by the minimum height among all decision trees in $\mathcal{T}_n^O$. Formally, $\forall A \in \mathcal{A}, \forall n \in \mathbb{N}$,*

$$\text{worst-case}(A, n) = \max_{P \in \mathbb{P}_n} \max_{I_P \in \mathbb{I}_I} \rho(A, I_P) \geq \min_{T \in \mathcal{T}_n^O} \text{Height}(T).$$

Universal optimality. Afshani, Barbay and Chan [1] introduce a strictly stronger notion of algorithmic optimality which they call *instance optimality in the order-oblivious setting.* We will instead use the equivalent modern term *universal optimality* [5, 6, 9]. Intuitively, an algorithm is *universally optimal* if it is worst-case optimal for every fixed point set P. Formally, we define the *universal* running time of an algorithm A and point set P as

$$\text{universal}(A, P) := \max_{I_P \in \mathbb{I}_P} \rho(A, I_P).$$

An algorithm is universally optimal if there exists a constant c such that for all P, $\text{universal}(A, P) \leq c \cdot \min_{A' \in \mathcal{A}} \text{universal}(A', P)$. To show universal optimality, we introduce the concept of *clairvoyant decision trees.* For a fixed point set P, a *clairvoyant decision tree* T is an abstract decision tree such that for any input $I_P \in \mathbb{I}_P$, there exists a root-to-leaf path in T such that the two lists at the leaf are a hull list and witness list of I_P, respectively. We denote by $\mathcal{T}_P^C$ the set of all clairvoyant decision trees for a fixed planar point set P.

▶ **Observation 7.** *For any planar point set P in general position, for any algorithm A, the universal running time of A is lower bounded by the minimum height amongst all clairvoyant decision trees in $\mathcal{T}_P^C$. Formally, for all planar point set P in general position, for all $A \in \mathcal{A}$,*

$$\text{universal}(A, P) = \max_{I_P \in \mathbb{I}_I} \rho(A, I_P) \geq \min_{T \in \mathcal{T}_P^C} \text{Height}(T).$$

Prior work. Kirkpatrick, McQueen, and Seidel [7] give an efficient algorithm to compute for any point set P its convex hull. We present this algorithm in Section 4 and observe that it naturally produces a hull list and a witness list. Afshani, Barbay, and Chan [1] show that the algorithm in [7] is universally optimal in a somewhat restricted model of computation which we define below. We note for the reader that this definition is not vital to our analysis.

▶ **Definition 8** (Definition 3.1 [1])**.** *A function $f : (\mathbb{R}^d)^c \mapsto \mathbb{R}$ is* multilinear *if the restriction of f is a linear function from R^d to R when any $(c-1)$ of the c arguments are fixed. Equivalently, f is multilinear if $f((x_{11}, \ldots, x_{1d}), \ldots, (x_{c1}, \ldots, x_{cd}))$ is a multivariate polynomial function that never multiplies coordinates of the same point.*

In the *multilinear decision tree model* [1], algorithms can access the input points only through tests of the form $f(p_1, \ldots, p_c) > 0$ for a multilinear function f with c constant.

▶ **Theorem 9** (Theorem 3.5+3.6 in [1])**.** *The algorithm in [7] is universally optimal for dimension $d = 2$ in the multilinear decision tree model.*

We instead show universal optimality by a counting all possible hull and witness lists:

▶ **Theorem 10.** *If we define $\mathcal{A}$ as the set of all* correct *comparison-based convex hull algorithms (Definition 5) then the algorithm in [7] for $d = 2$ is universally optimal.*

3 Creating a universal lower bound

Let P be a point set of n points in general position. We use our definition of hull lists and witness lists to show a universal lower bound. That is, we define a quantity Q such that for all algorithms $A \in \mathcal{A}$, $\text{Universal}(A, P) \geq Q$. To this end, we explore the definition of a witness list. Recall that $\mathbb{I}_P$ is the set of all $n!$ arrays that contain P. Consider any list $W = (a_i, b_i, c_i)_{i=1}^{n}$ of integer triples.

For a fixed W, there can be many $I_P \in \mathbb{I}_P$ for which W is a witness list. For example, let P be a point set containing $n-3$ points in a small ball around $(0,0)$, and the three corners (α, β, γ) of a unit equilateral triangle centred at $(0,0)$ (we refer to Figure 1). Let:

$$W = ((-1,-1,-1),(-1,-1,-1),(-1,-1,-1),(0,1,2),(0,1,2),\ldots,(0,1,2)).$$

Any $I_P \in \mathbb{I}_P$ which contains (α, β, γ) in the first three positions has W as a witness list. Specifically, there are $3! \cdot (n-3)!$ arrays $I_P \in \mathbb{I}_P$ which have W as a witness list. There does not exist any other witness list W^* where more than $3!(n-3)!$ arrays in $\mathbb{I}_P$ have W^* as a witness list. We use this maximum quantity for our lower bound:

▶ **Definition 11.** *Let P be a point set and $W = (a_i, b_i, c_i)_{i=1}^{n} \subset \mathbb{Z}^{n\times 3}$. We define*

$$V(P, W) := \left\{ I_P \in \mathbb{I}_P \middle| W \text{ is a witness list of } I_P \right\}.$$

We consider the maximal number of input arrays that can have the same witness list:

$$V_{\max}(P) := \max_{W \in \mathbb{Z}^{n\times 3}} |V(P, W)|.$$

▶ **Theorem 12.** *Let P be a point set of n points and A be any algorithm in $\mathcal{A}$, then*

$$\text{universal}(A, P) \in \Omega\Big(n + |\operatorname{ch}(P)| \cdot \log n + \log \frac{n!}{V_{\max}(P)}\Big).$$

Proof. Any algorithm must spend $\Omega(n)$ time to read the input. Denote by ℓ the minimum height across all clairvoyant decision trees in $\mathcal{T}_P^C$. By Observation 7, $\Omega(\ell)$ is a universal lower bound. What remains is to bound ℓ. Let $k = |\operatorname{ch}(P)|$. Let T be an arbitrary clairvoyant decision tree in $\mathcal{T}_P^C$. Recall that each leaf of T stores a hull list and a witness list.

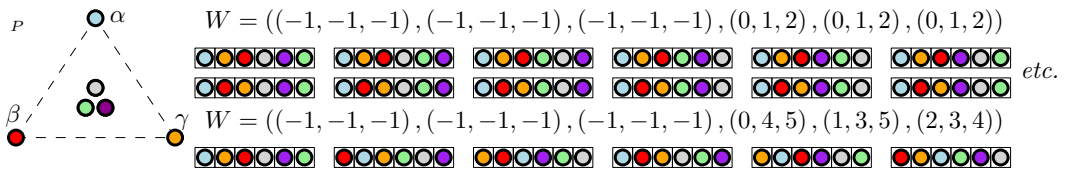


Figure 1 For $W = ((-1,-1,-1),(-1,-1,-1),(-1,-1,-1),(0,1,2),(0,1,2),(0,1,2))$ there are 36 $I_P \in \mathbb{I}_P$ for which W is a witness list (any array that assigns (α, β, γ) to the first three cells). For $W = ((-1,-1,-1),(-1,-1,-1),(-1,-1,-1),(0,4,5),(1,3,5),(2,3,4))$ there are only 6.

For any input $I_P \in \mathbb{I}_P$, there is a unique hull list of I_P. It follows that the number of distinct hull lists, and therefore the number of distinct leaves of T, across all I_P is at least $\frac{n!}{(n-k)!}$. Hence, $\ell \geq \log(\frac{n!}{(n-k)!}) \in \Omega(k \log n)$. For any input $I \in \mathbb{I}_P$, there is at least one leaf in T containing a witness list W of I. Therefore,

$$n! = |\mathbb{I}_P| \leq \sum_{\ell \in L} |V(P, W)| \leq (\#\text{ leaves of } T) \cdot V_{\max}(P) \quad \Rightarrow \quad \ell \geq \log \frac{n!}{V_{\max}(P)}. \qquad \blacktriangleleft$$

4 Analysing Kirkpatrick-McQueen-Seidel

Kirkpatrick and Seidel present an output-sensitive algorithm for computing the convex hull for a planar point set [7]. McQueen suggests a modification to the algorithm by Kirkpatrick and Seidel in Footnote 2 of [7]. The modified algorithm, depicted in Algorithm 1, is universally optimal. We first describe their algorithm. Next, we construct a geometric object that we will call a *quadrangle tree* and use it to analyse the running time of Algorithm 1.

High-level overview. Their algorithm is recursive and it constructs the *upper convex hull* of P. The original input consists of an unordered array I_P, from which they find the leftmost and rightmost vertex of P in linear time. Each recursive call has as input some set $S \subset P$ and two points $p_\ell, p_r \in S$ such that (p_ℓ, p_r) is an edge of $\mathrm{CH}(S)$ (Figure 2) – we assume that the line segment between the leftmost and rightmost vertex lies on $\mathrm{CH}(P)$. Otherwise, this line segment splits the problem into two independent convex hull construction problems.

Given (S, p_ℓ, p_r), the algorithm first finds a point $m \in S$ with the median x-coordinate in linear time. It then finds in linear time an edge (p_i, p_j) of $\mathrm{CH}(P) \cap S$ with $(p_i, p_j) \neq (p_\ell, p_r)$ such that m lies inside or on the quadrangle $Q = (p_\ell, p_r, p_j, p_i)$ (Q may also be a triangle, for example when $p_i = p_\ell$). The algorithm partitions S in linear time into three sets:

- S_1 consists of all points of S that lie above or on the line $p_\ell p_i$,
- S_2 consists of all points of S that lie above or on the line $p_r p_j$, and
- $S^* = S - S_1 - S_2$. S^* lies within Q and, therefore, its points are not on the convex hull. Through Q we assign these points a witness in $O(|S^*|)$ time at no asymptotic overhead.

The algorithm recurses on S_1 and S_2, using the segments (p_ℓ, p_i) and (p_r, p_j), respectively.

4.1 Runtime analysis

We propose a novel fine-grained runtime analysis of this algorithm. To this end, we define what we will call the *quadrangle tree* (we refer to Figure 3 (a)).

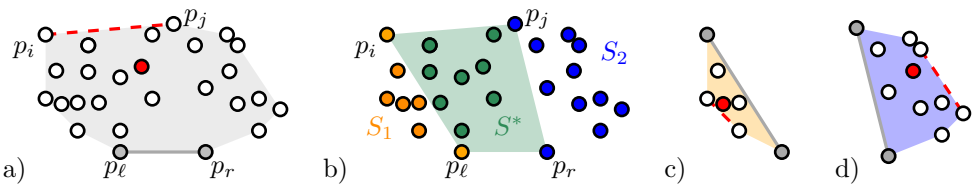


Figure 2 (a) The input are the points and the grey segment. We compute the point m with median x-coordinate and the edge (p_i, p_j). (b) Given $Q = (p_\ell, p_r, p_j, p_i)$, we partition the points into three sets: S_1, S^*, S_2. (c + d) We recurse on S_1 and S_2 using their respective edges of Q.

Algorithm 1 Kirkpatrick-McQueen-Seidel(unordered point set S, edge (p_ℓ, p_r) of $\mathrm{CH}(S)$) [7].

Require: $S = \{p \in P \mid p \text{ lies on or above the line } p_\ell p_r\}$ and $p_\ell, p_r \in \mathrm{ch}(P)$.

if $p_\ell = p_r$ **then**
  Append p_ℓ to the hull list
  return
end if
$m \leftarrow \arg\mathrm{median}\,\{x(p) \mid p \in S\}$
$(p_i, p_j) \leftarrow$ edge of $\mathrm{CH}(S)$ such that $m \in Q = (p_\ell, p_r, p_j, p_i)$ *bridge-finding* [7].
$S_1 \leftarrow \{p \in S \mid p \text{ lies on or above the line } p_\ell p_i\}$
$S_2 \leftarrow \{p \in S \mid p \text{ lies on or above the line } p_j p_r\}$
$S^* \leftarrow S - S_1 - S_2$
Kirkpatrick-McQueen-Seidel$(S_1, (p_\ell, p_i))$
Kirkpatrick-McQueen-Seidel$(S_2, (p_j, p_r))$
GiveWitness(S^*, p_ℓ, p_r, p_j, p_i) *Note that this last step is not explicit in [7].*

▶ **Definition 13.** *A* rooted convex polygon (C, p, q) *is a convex polygon C together with an edge pq of C. Given distinct vertices of r, s of C the line rs splits C into two pieces (one piece is empty if rs is an edge of C). Let C^{rs} denote the piece that* does not *contain pq.*

▶ **Definition 14.** *A* quadrangle tree $\mathcal{Q}$ *of a rooted convex polygon (C, p, q) is a binary tree in which every node stores a quadrangle spanned by (up to) four vertices of C, such that:*

- *The quadrangle at the root node is spanned by p, q, s, r where rs is an edge of C. We allow for $p = r$ and/or $q = s$.*
- *If $q \neq r$, then the root has a child node whose subtree is a quadrangle tree of (C^{pr}, p, r).*
- *If $s \neq p$, then the root has a child node whose subtree is a quadrangle tree of (C^{qs}, q, s).*

Given a point set P and a quadrangle tree $\mathcal{Q}$, the *population* of the root node are all points in P that lie on or inside the quadrangle $pqrs$, except for those that lie on the line pq. Observe that, since we assume that P lies in general position, the population of the root node equals all points in the interior of $pqrs$ plus: r if $p \neq r$ and s if $q \neq s$.

▶ **Observation 15.** *Given the input array I_P, the algorithm by Kirkpatrick, McQueen and Seidel finds the leftmost and rightmost vertices of P in linear time. For each point set P Algorithm 1 constructs, independent of the input ordering $I_P \in \mathbb{I}_P$, a quadrangle tree $\mathcal{Q}$ where each call of Kirkpatrick-McQueen-Seidel(S, (p_ℓ, p_r)) uniquely corresponds to a node (C, p, q) in $\mathcal{Q}$ with $C = \mathrm{CH}(S)$, $p = p_\ell$ and $q = p_r$.*

Ordered downdrafts. For a point set P and a quadrangle tree $\mathcal{Q}$ of $\mathrm{CH}(P)$, we create an *ordered downdraft*. Intuitively, this is a mapping $\varphi : P - ch(P) \to P$ such that for each $p \in P - ch(P)$, the point $\varphi(p)$ lies deeper in the quadrangle tree, or, in the same node as p but farther from the corresponding root edge. This is our most technically involved definition.

▶ **Definition 16** (Figure 3(b))**.** *We define for any quadrangle tree $\mathcal{Q}$ of $\mathrm{CH}(P)$ a partial order $\prec_{\mathcal{Q}}$ on P. For $p \in P$, let $r(p)$ be the node in $\mathcal{Q}$ within whose population p lies. Let $H_{r(p)}$ be the halfplane defined by the rooted edge of $r(p)$. For $p, q \in P$, we say that $p \prec_{\mathcal{Q}} q$ if*

- *$r(p)$ is a strict ancestor of $r(q)$, or*
- *$r(p) = r(q)$, and q lies deeper inside the halfspace $H_{r(p)}$ than p.*

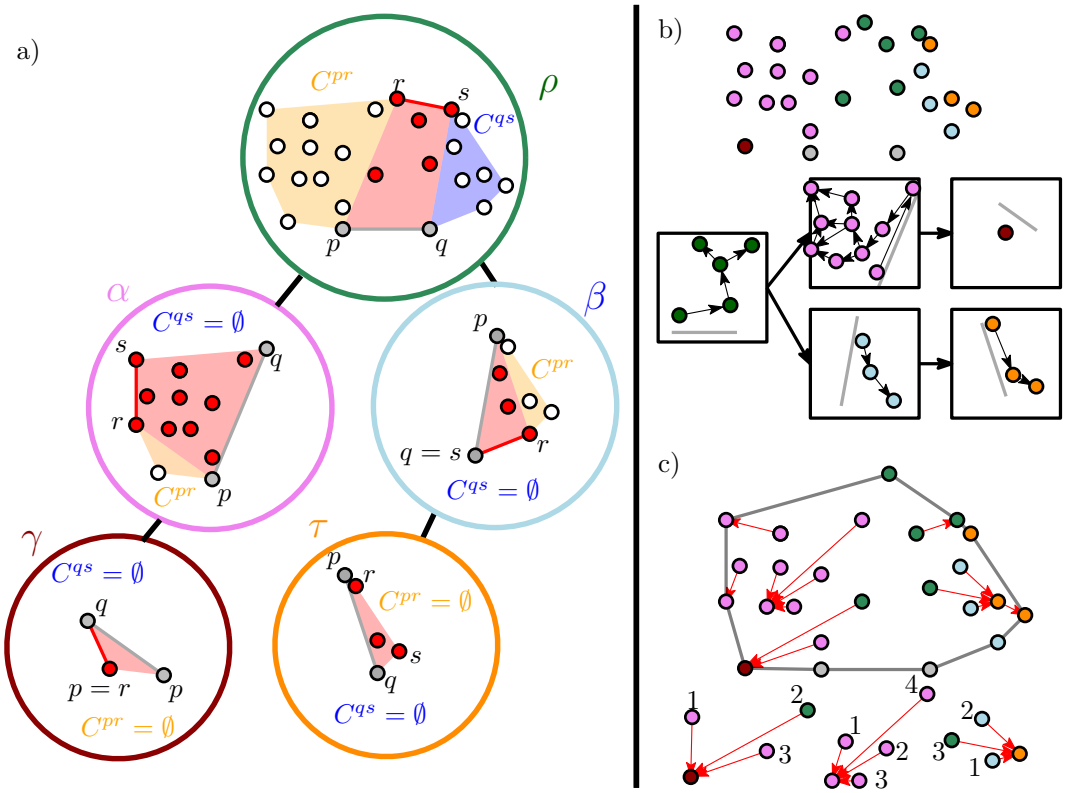


Figure 3 (a) From a rooted polygon (C, p, q) we construct a quadrangle tree $\mathcal{Q}$ with six nodes: $\rho, \alpha, \beta, \gamma, \tau$. For each node in the tree, the red coloured vertices indicate its population.
(b) We assign all points $p \in P$ a colour based on $r(p)$. For example, for all pink points p the node $r(p)$ equals $\alpha \in \mathcal{Q}$. We illustrate a partial order over the points in P where for all p, q with $r(p) = r(q)$ the point $p \prec_{\mathcal{Q}} q$ if and only if p lies further from the line supporting the grey segment.
(c) We create an ordered downdraft φ by mapping each point in $P - ch(P)$ to a point in P. For all points q where multiple points in P map to q, we create a linear order over $\varphi^{-1}(\{q\})$.

Given a quadrangle tree $\mathcal{Q}$, we define a *downdraft* φ as a map that maps each point in $P - \mathrm{ch}(P)$ to a point in P. Specifically, it maps the point either to a quadrangle lower in the tree, or to a point in the same quadrangle that is strictly further away from the root edge.

▶ **Definition 17.** *Let P be a point set and $\mathcal{Q}$ be a quadrangle tree for* $\mathrm{ch}(P)$*. A* downdraft *for $(P, \mathcal{Q})$ is a map $\varphi : (P - \mathrm{ch}(P)) \to P$ such that $\forall p \in P$, we have $p \prec_{\mathcal{Q}} \varphi(p)$.*

Given any downdraft φ and point $q \in P$, the *fiber* $\varphi^{-1}(\{q\})$ is the set of all points $p \in P$ that get mapped to q.

▶ **Definition 18** (Figure 3(c))**.** *An* ordered downdraft $\overline{\varphi} = (\varphi, \prec_{\overline{\varphi}})$ *of $(P, \mathcal{Q})$ is a downdraft φ of $(P, \mathcal{Q})$ plus a total order $\prec_{\overline{\varphi}}$ on each fiber $\varphi^{-1}(\{p\})$ for $p \in P$. We define:*

$\mathrm{OD}(P, \mathcal{Q})$ to be the set of all ordered downdrafts of $(P, \mathcal{Q})$.

4.2 A universal upper bound

Let A denote the Kirkpatrick-McQueen-Seidel algorithm. Recall that Algorithm 1 naturally creates a quadrangle tree $\mathcal{Q}$ of $\mathrm{CH}(P)$, where each call of $\mathrm{A}(S, (p_\ell, p_r))$ uniquely corresponds to a node (C, p, q) in $\mathcal{Q}$ with $C = \mathrm{CH}(S)$, $p = p_\ell$ and $q = p_r$. We show a universal upper bound for the running time of A by counting the number of ordered downdrafts of $\mathcal{Q}$.

▶ **Theorem 19.** *Let P be a planar point set in general position with n points. Let $\mathcal{Q}$ be the corresponding quadrangle tree from Observation 15 and let A denote the algorithm by Kirkpatrick, McQueen and Seidel which as input receives some $I_P \in \mathbb{I}_P$ (and the leftmost and rightmost point in P). Then there exists a constant c such that*

$$\forall I_P \in \mathbb{I}_P, \quad \rho(A, I_P) \leq c \cdot \left(n + \log \frac{n^n}{|\operatorname{OD}(P, \mathcal{Q})|}\right).$$

In Section 5, we show that Algorithm 1 is universally optimal by combining the upper bound from Theorem 19 with the lower bound from Theorem 12.

The proof. Every node in $\mathcal{Q}$ corresponds to a call $A(S, (p_\ell, p_r))$ with $p_\ell \neq p_r$. The quadrangle at this node is $p_\ell p_r p_j p_i$, and its child nodes correspond to the calls $A(S_1, (p_\ell, p_i))$ and $A(S_2, (p_j, p_r))$. We denote the subtree of this node by $\mathcal{Q}(S)$. We put $\mathcal{Q} = \mathcal{Q}(P)$.

▶ **Lemma 20.** *Let S, S_1, S_2 be the sets in some recursive call of Algorithm 1. Then*

$$|\operatorname{OD}(S, \mathcal{Q}(S))| \leq |S|^{|S \setminus (S_1 \cup S_2)|} \cdot |\operatorname{OD}(S_1, \mathcal{Q}(S_1))| \cdot |\operatorname{OD}(S_2, \mathcal{Q}(S_2))|.$$

Proof. First, we show the following upper bound:

$$|\operatorname{OD}(S, \mathcal{Q}(S))| \leq |S|^{|S \setminus (S_1 \cup S_2)|} \cdot |\operatorname{OD}(S_1 \cup S_2, \mathcal{Q}(S))|.$$

Indeed, any ordered downdraft in $\operatorname{OD}(S, \mathcal{Q}(S))$ can be obtained from an ordered downdraft in $\operatorname{OD}(S_1 \cup S_2, \mathcal{Q}(S))$ by placing each element of $S \setminus (S_1 \cup S_2)$ in some fiber, and by placing it at some point in the ordering of the fiber. Within a fiber, an element of S can be placed in at most $|S|$ different ways.

Finally, note that all points in $S_1 - CH(P)$ lie in the left child of $\mathcal{Q}(S)$ and all points in $S_2 - CH(P)$ lie in its right child. Thus, any downdraft over $(S_1 \cup S_2, \mathcal{Q}(S))$ may be obtained by constructing downdrafts over $(S_1, \mathcal{Q}(S_1))$ and $(S_2, \mathcal{Q}(S_2))$ independently, and thus

$$|\operatorname{OD}(S_1 \cup S_2, \mathcal{Q}(S))| = |\operatorname{OD}(S_1, \mathcal{Q}(S_1))| \cdot |\operatorname{OD}(S_2, \mathcal{Q}(S_2))|. \quad \blacktriangleleft$$

For any call $A(S, (p_\ell, p_r))$ made during the execution of the Kirkpatrick-McQueen-Seidel algorithm, let $T(S)$ denote total running time spent on this call (including recursion). Since both median finding and bridge finding take $O(|S|)$ time, there is an absolute constant c such that:

$$T(S) \leq T(S_1) + T(S_2) + c \cdot |S|. \qquad \text{For this constant } c\text{, we show:}$$

▶ **Lemma 21.** *Denote for any call $A(S, (p_\ell, p_r))$ by $T(S)$ the time spent on executing the call (including all time spent during recursion). Then*

$$T(S) \leq c \cdot \left(|S| + |S| \log |S| - \log |\operatorname{OD}(S, \mathcal{Q}(S))|\right).$$

Proof. We use induction over $|S|$. If $p_\ell = p_r$, then $|S| = 1$ and $T(S) \leq c$. Otherwise, let $n = |S|, n_1 = |S_1|, n_2 = |S_2|$. By induction,

$$\begin{aligned} T(S) &\leq T(S_1) + T(S_2) + c \cdot |S| \\ &\leq c \cdot \left(n_1 + n_2 + n_1 \log n_1 + n_2 \log n_2 - \log |\operatorname{OD}(S_1, \mathcal{Q}(S_1))| - \log |\operatorname{OD}(S_2, \mathcal{Q}(S_2))| + n\right) \\ &\leq c \cdot \left(n_1 \log(2n_1) + n_2 \log(2n_2) - \log |\operatorname{OD}(S_1, \mathcal{Q}(S_1))| - \log |\operatorname{OD}(S_2, \mathcal{Q}(S_2))| + n\right) \end{aligned}$$

Since, S_1, S_2 are computed via the median x-coordinate, $n_1, n_2 \leq \frac{n}{2}$. By Lemma 20,

$$-\log|\operatorname{OD}(S_1, \mathcal{Q}(S_1))| - \log|\operatorname{OD}(S_2, \mathcal{Q}(S_2))| \leq -\log|\operatorname{OD}(S, \mathcal{Q}(S))| + (n - n_1 - n_2)\log n$$

Therefore,

$$\begin{aligned} T(S) &\leq c \cdot \Big(n_1 \log n + n_2 \log n - \log|\operatorname{OD}(S, \mathcal{Q}(S))| + (n - n_1 - n_2)\log n) + n\Big) \\ &= c \cdot \Big(n + n\log n - \log|\operatorname{OD}(S, \mathcal{Q}(S))|)\Big) \end{aligned}$$ ◀

We now apply Lemma 21 to the first call $A(P, (p_\ell, p_r))$ of Algorithm 1. Note that per definition, $\mathcal{Q}(P) = \mathcal{Q}$ and $|P| = n$. Lemma 21 guarantees that, regardless of the order I_P we receive the input P in, the running time is at most

$$c \cdot \Big(n + n\log n - \log|\operatorname{OD}(P, \mathcal{Q})|)\Big) = c \cdot \Big(n + \log \frac{n^n}{|\operatorname{OD}(P, \mathcal{Q})|}\Big),$$

which proves Theorem 19.

5 Showing Universal Optimality

In Section 3, we showed for each point set P a universal lower bound (Theorem 12). This bound uses the quantity $V_{\max}(P)$. This quantity is the maximum m, for which there exists a list $W = (a_i, b_i, c_i)_{i=1}^n \subset \mathbb{Z}^{n\times 3}$ with m inputs $I_P \in \mathbb{I}_P$ for which W is a witness list of I_P.

In Section 4, we showed an upper bound for Algorithm 1 (Theorem 19). For each input I_P, the execution of Algorithm 1 on I_P (together with an edge between the leftmost and rightmost vertex in I_P) defines a *quadrangle tree* $\mathcal{Q}$. Our upper bound uses the quantity $|\operatorname{OD}(P, \mathcal{Q})|$. This quantity counts the maximum number of ordered downdrafts of $(P, \mathcal{Q})$.

Here, we show that Algorithm 1 is universally optimal by correlating these two quantities. We achieve this through two intermediate quantities: the number of *corner assignments* of W and the number of *hull lists* of P.

▶ **Definition 22.** *Let $W = (a_i, b_i, c_i)_{i=1}^n$. A* corner assignment *picks a corner for each of the n triangles in W. Formally, it is a map $C : [n] \to [n]$ such that $C(i) \in \{a_i, b_i, c_i\}$ for all $i \in [n]$. Let $\operatorname{CA}(W)$ denote the set of all corner assignments of W. Note that $|\operatorname{CA}(W)| = 3^n$.*

▶ **Definition 23.** *We define $\operatorname{HL}(P)$ as the set of all lists of $k = |ch(P)|$ integers such that $\forall H \in \operatorname{HL}(P)$ there exists an input $I_P \in \mathbb{I}_P$ where H is a hull list of I_P. Note that $|\operatorname{HL}(P)| = \frac{n!}{(n-k)!} \leq n^k$.*

▶ **Lemma 24.** *Let W be an arbitrary witness for P. Let $\mathcal{Q}$ be an arbitrary quadrangle tree for P. There exists an injective map*

$$\Phi : V(P, W) \hookrightarrow \operatorname{OD}(P, \mathcal{Q}) \times \operatorname{CA}(W) \times \operatorname{HL}(P).$$

Proof. Fix $I_P \in V(P, W)$. By our canonical ordering of the hull lists, there exists exactly one $H \in HL(P)$ for which H is a hull list of I_P. We define our map Φ by additionally constructing for I_P a corner assignment C and ordered downdraft $\overline{\varphi}$.

For $i \in [n]$, observe that if $I_p[i]$ is not in $\operatorname{ch}(P)$ then $I_p[i]$ lies strictly inside the triangle formed by $(I_P[a_i], I_P[b_i], I_P[c_i])$. Therefore, at least one of $I_P[i] \prec_{\mathcal{Q}} I_P[a_i], I_P[i] \prec_{\mathcal{Q}} I_P[b_i]$ or $I_P[i] \prec_{\mathcal{Q}} I_P[c_i]$ holds. We construct the corner assignment C by arbitrarily choosing for

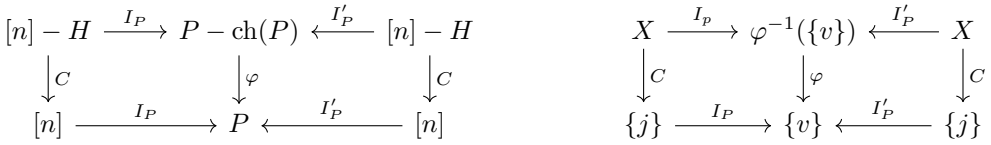


Figure 4 (left) the commuting diagram corresponding to (I_P, I'_P, φ, C). (right) In our proof, we chase this diagram from v encountering the integer j, the set X, and the fiber $\varphi^{-1}(\{v\})$.

each $i \in [n]$, the value $C(i) \in \{a_i, b_i, c_i\}$ such that $I_P[i] \prec_{\mathcal{Q}} I_P[C(i)]$. We then construct a downdraft φ by setting $\varphi(I_P[i]) = I_P[C(i)]$. To obtain an ordered downdraft, we order for all $q \in P$ each fiber $\varphi^{-1}(\{q\})$ via the following rule:

$$\forall I_P[i], I_P[j] \in \varphi^{-1}(\{q\}) : \qquad I_P[i] \prec_{\overline{\varphi}} I_P[j] \quad \Leftrightarrow \quad i < j$$

Then $\overline{\varphi}$ is an ordered downdraft and we have defined the map $\Phi(I_P) = (\overline{\varphi}, C, H)$.

We now show that this map Φ is injective. Suppose that I_P and I'_P are elements of $V(P, W)$ such that $\Phi(I_P) = (\overline{\varphi}, C, H) = \Phi(I'_P)$. Observe that, per definition, the hull list $H \in \Phi(I_P) = \Phi(I'_P)$ is a sequence of integers with the following property: if u is the x'th point on $\mathrm{ch}(P)$ then $I_P[H[x]] = I'_P[H[x]] = u$. Suppose for the sake of contradiction that there exist integers $i, i' \in [n]$ with $i \neq i'$ such that $u = I_P[i] = I'_P[i']$. We pick the pair (i, i') such that u is maximal in the partial order $\prec_{\mathcal{Q}}$.

First, consider the case where u is a point on $\mathrm{ch}(P)$. Let u be the x'th element in the canonical ordering of $\mathrm{ch}(P)$. Then $u = I_P[H[x]] = I'_P[H[x]]$. However, we chose u such that $u = I_P[i] = I'_P[i']$ and so $i = i'$, a contradiction.

Alternatively, suppose that $u \notin \mathrm{ch}(P)$. By maximality of our choice of u, it follows that for all points v' with $u \prec_{\mathcal{Q}} v'$, there exists a unique integer j' such that $v' = I_P[j'] = I'_P[j']$. We define $v := \varphi(u)$. By the definition of a downdraft, $u \prec_{\mathcal{Q}} v$ and so there is a unique integer $j \in [n]$ with $v = \varphi(u) = I_P[j] = I'_P[j]$.

We observe that the objects I_P, I'_P, φ and C are all maps. In Figure 4 we illustrate the diagram corresponding to these maps. Recall that H is the hull list of both $\Phi(I_P)$ and $\Phi(I'_P)$. The left diagram in Figure 4 commutes. Indeed, by our definition of the downdraft φ, we chose φ such that $\varphi(I_p[x]) = I_P[C(x)]$ for all $x \in [n] \setminus H$.

Consider chasing the left square of this diagram from $v = \varphi(u)$ (see Figure 4, right). From v, we may first apply I_P^{-1} to obtain j. We then apply C^{-1} to j to get a set of indices $X := C^{-1}(I_P^{-1}(\{v\}))$. Conversely, we may chase this diagram from v by first applying φ^{-1} to obtain a set of points, and then applying I_P^{-1} to obtain a set of indices. Since the diagram commutes, also $I_P^{-1}(\varphi^{-1}(\{v\})) = X$. Next, we focus on the right square of the diagram. Since $v = I'_P[j]$, we have $C^{-1}(I'^{-1}_P(\{v\})) = X$. We may chase the right square from v by first applying φ^{-1} to obtain a set of points, and then applying I'^{-1}_P. Since this diagram commutes, it follows that $I'^{-1}_P(\varphi^{-1}(\{v\})) = X$ too. We now use this set X to obtain a contradiction:

We assumed that $(\overline{\varphi}, C, H) = \Phi(I_P) = \Phi(I'_P)$. Recall that $\prec_{\overline{\varphi}}$ induces a total order on $\varphi^{-1}(\{v\})$. Per definition of $\Phi(I_P)$, all points $p \in \varphi^{-1}(\{v\})$ are ordered by their index x defined via $p = I_P[x]$. The rank of a value u with respect to an ordered set is its index in the order. We can consider the rank of u in two sets: X and $\varphi^{-1}(\{v\})$. Recall that we ordered for all $q \in P$ each fiber $\varphi^{-1}(\{q\})$ via the following rule:

$$\forall\, I_P[a], I_P[b] \in \varphi^{-1}(\{q\}) : \qquad I_P[a] \prec_{\overline{\varphi}} I_P[b] \quad \Leftrightarrow \quad a < b$$

It follows that the rank of u satisfies:

$$\mathrm{Rank}_{\prec_{\overline{\varphi}}}\left(u, \varphi^{-1}(\{v\})\right) = \mathrm{Rank}_{<}\left(i, I_P^{-1}(\varphi^{-1}(\{v\}))\right) = \mathrm{Rank}_{<}\left(i, X\right)$$

Similarly,

$$\mathrm{Rank}_{\prec_{\overline{\varphi}}}\Big(u, \varphi^{-1}(\{\varphi(u)\})\Big) = \mathrm{Rank}_{<}\Big(i', C^{-1}(\{j\})\Big) = \mathrm{Rank}_{<}\Big(i', X\Big).$$

Since i and i' both have the same rank in X, it follows that $i = i'$, contradiction. This shows that $I_P = I'_P$, so our constructed map Φ is injective. ◀

The above lemma immediately implies the following relation between $V_{\max}(P)$ and $\mathrm{OD}(P, \mathcal{Q})$:

▶ **Corollary 25.** *For every quadrangle tree $\mathcal{Q}$ for P,*

$$V_{\max} \leq |\,\mathrm{OD}(P, \mathcal{Q})| \cdot 3^n \cdot n^k.$$

Finally, we are ready to show universal optimality of Algorithm 1:

▶ **Theorem 10.** *If we define $\mathcal{A}$ as the set of all* correct *comparison-based convex hull algorithms (Definition 5) then the algorithm in [7] for $d = 2$ is universally optimal.*

Proof. Let $\mathcal{Q}$ denote the quadrangle tree that corresponds to executing Algorithm 1. We apply Theorem 19 to note that there exists a constant c such that:

$$\max_{I_P \in \mathbb{I}_P} \rho(A, I_P) \leq c \cdot \Big(n + \log \frac{n^n}{|\,\mathrm{OD}(P, \mathcal{Q})|}\Big).$$

By Corollary 25,

$$\log \frac{n^n}{|\,\mathrm{OD}(P, \mathcal{Q})|} \leq \log \frac{3^n \cdot n^k \cdot n^n}{V_{\max}} = n \log(3) + k \log n + \log \frac{n^n}{V_{\max}}$$

Since $\log(3) \leq 2$ and $\log n^n \leq 2n + \log n!$, this yields

$$\log \frac{n^n}{|\,\mathrm{OD}(P, \mathcal{Q})|} \leq 4n + k \log n + \log \frac{n!}{V_{\max}}$$

We may now apply the universal lower bound from Theorem 12 to obtain the theorem. ◀

References

1 Peyman Afshani, Jérémy Barbay, and Timothy Chan. Instance-optimal geometric algorithms. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 129–138. Association for Computing Machinery, 2009. doi:10.1109/FOCS.2009.34.

2 Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-optimal geometric algorithms. *Journal of the ACM (JACM)*, 64(1), March 2017. doi:10.1145/3046673.

3 Mark de Berg, Matthew Katz, A Frank Van der Stappen, and Jules Vleugels. Realistic input models for geometric algorithms. In *Symposium on Computational geometry (SoCG)*, pages 294–303, 1997. doi:10.1145/262839.262986.

4 Jeff Erickson. Dense point sets have sparse delaunay triangulations or "... but not too nasty". *Discrete & Computational Geometry*, 33:83–115, 2005. doi:10.5555/3115459.3115691.

5 Bernhard Haeupler, Richard Hladík, Václav Rozhoň, Robert E Tarjan, and Jakub Tetěk. Universal optimality of dijkstra via beyond-worst-case heaps. In *Symposium on Foundations of Computer Science (FOCS)*, pages 2099–2130. IEEE, 2024. doi:10.1109/FOCS61266.2024.00125.

6 Bernhard Haeupler, Richard Hladík, John Iacono, Václav Rozhoň, Robert E. Tarjan, and Jakub Tětek. Fast and simple sorting using partial information. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3953–3973, 2025. doi:10.1137/1.9781611978322.134.

7 David G Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986. doi:10.1137/0215021.

8 Jiří Matoušek, János Pach, Micha Sharir, Shmuel Sifrony, and Emo Welzl. Fat triangles determine linearly many holes. *SIAM Journal on Computing*, 23(1):154–169, 1994. doi:10.1109/SFCS.1991.185347.

9 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. Simpler optimal sorting from a directed acyclic graph. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 350–355. SIAM, 2025. doi:10.1137/1.9781611978315.26.

On the Satisfiability of Random 3-SAT Formulas with k-Wise Independent Clauses

Ioannis Caragiannis
Department of Computer Science, Aarhus University, Denmark

Nick Gravin
Key Laboratory of Interdisciplinary Research of Computation and Economics,
Shanghai University of Finance and Economics, Ministry of Education, China

Zhile Jiang
Department of Computer Science, Aarhus University, Denmark

Abstract

The problem of identifying the satisfiability threshold of random 3-SAT formulas has received a lot of attention during the last decades and has inspired the study of other threshold phenomena in random combinatorial structures. The classical assumption in this line of research is that, for a given set of n Boolean variables, each clause is drawn uniformly at random among all sets of three literals from these variables, independently from other clauses. Here, we keep the uniform distribution of each clause, but deviate significantly from the independence assumption and consider richer families of probability distributions. For integer parameters n, m, and k, we denote by $\mathcal{F}_k(n,m)$ the family of probability distributions that produce formulas with m clauses, each selected uniformly at random from all sets of three literals from the n variables, so that the clauses are k-wise independent. Our aim is to make general statements about the satisfiability or unsatisfiability of formulas produced by distributions in $\mathcal{F}_k(n,m)$ for different values of the parameters n, m, and k.

Our technical results are as follows: First, all probability distributions in $\mathcal{F}_2(n,m)$ with $m \in \Omega(n^3)$ return unsatisfiable formulas with high probability. This result is tight. We show that there exists a probability distribution $\mathcal{D} \in \mathcal{F}_3(n,m)$ with $m \in O(n^3)$ so that a random formula drawn from $\mathcal{D}$ is almost always satisfiable. In contrast, for $m \in \Omega(n^2)$, any probability distribution $\mathcal{D} \in \mathcal{F}_4(n,m)$ returns an unsatisfiable formula with high probability. This is our most surprising and technically involved result. Finally, for any integer $k \geq 2$, any probability distribution $\mathcal{D} \in \mathcal{F}_k(n,m)$ with $m \in O(n^{1-1/k})$ returns a satisfiable formula with high probability.

2012 ACM Subject Classification Theory of computation → Generating random combinatorial structures; Mathematics of computing → Combinatoric problems; Mathematics of computing → Probability and statistics

Keywords and phrases Random 3-SAT, k-wise independence, Random bipartite graph

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.103

Related Version *Full Version*: `https://arxiv.org/abs/2411.03813`

Funding The research is supported by National Key R&D Program of China (2023YFA1009500) and "the Fundamental Research Funds for the Central Universities".

1 Introduction

Satisfiability of propositional formulas (SAT) is one of the most renowned problems in theoretical computer science. It appeared in the first lists of NP-complete problems independently proposed by Cook and Levin [44], and is pivotal for many developments in modern complexity theory. Today, many lower bounds on the running time of algorithms rely on the Exponential Time Hypothesis for solving SAT [11, 18, 36, 37]. On the practical side, SAT solvers are frequently deployed in hardware circuit design, model checking, program verification, automated planning and scheduling, as well as in solving real-life instantiations


© Ioannis Caragiannis, Nick Gravin, and Zhile Jiang;
licensed under Creative Commons License CC-BY 4.0



of combinatorial optimization problems such as FCC spectrum auctions. Modern SAT solvers often find solutions to large industrial instances with thousands or even millions of variables despite the NP-hardness of the problem. However, there is still a large discrepancy between the performance of SAT solvers on those instances and theoretical average-case predictions, which have been studied in great depth under the line of research on *random SAT*.

Random SAT

A j-CNF formula ϕ over n variables is composed of m OR-clauses, each containing exactly j literals of j different variables. In the most commonly studied random SAT model, a formula ϕ is generated uniformly at random from all possible j-CNF formulas over n variables and m clauses. The most prominent theoretical question related to random SAT is to identify the satisfiability threshold r_j such that $\lim_{n\to\infty} \mathbf{Pr}[\phi \text{ is satisfiable}]$ is equal to 0 when $m/n > r_j$, and equal to 1 when $m/n < r_j$. It has been established [15] that 2-SAT has $r_2 = 1$, and its phase transition window [10] is $m \in [n - \Theta(n^{1/3}), n + \Theta(n^{1/3})]$. For $j \geq 3$, the asymptotic j-SAT threshold was shown to be $2^j \log 2 - \frac{1}{2}(1 + \log 2) \pm o_j(1)$ as $j \to \infty$ [17] (improving previous results from [3]), while for large enough j the exact value of r_j was determined in [20]. However, the question of identifying r_j for small values of j remains open. In particular, random 3-SAT has attracted a lot of attention. For the lower bound part, it has been shown in a series of papers [15, 12, 32, 1, 35, 39] that $r_3 \geq 3.52$ (the currently best known bound is due to [35, 39]). The upper bound part is studied by [27, 38, 21, 19]; the currently best known bound is $r_3 < 4.49$ due to [19]. The estimate $r_3 \approx 4.26$ was derived from numerical experiments [40] (see also [14, 41]).

A more recent line of work [29, 30, 31, 42] extends the standard model of random j-SAT to non-uniform distributions. Their motivation comes from the empirical observation that, in practice, CNF formulas often have rather different frequencies/probabilities for the n variables to appear in each clause (following a power-law distribution instead of a uniform one). Namely, Friedrich and Rothenberger [31] proposed a non-uniform random model, where the literals $\{x_i, \overline{x_i}\}_{i\in[n]}$ are selected independently at random in each clause c of the random j-CNF with $\mathbf{Pr}[x_i \in c] = \mathbf{Pr}[\overline{x_i} \in c] = p_i$ and where probabilities $\mathbf{p} = (p_i)_{i\in[n]}$ may vary across different variables. They find satisfiability threshold $r_2(\mathbf{p})$ of non-uniform random 2-SAT for certain regimes depending on $\mathbf{p}$. However, the non-uniform model of [31] does not capture the community biases/correlations (i.e., the fact that certain variables are more likely to appear together in a clause), which are often observed in practice [6]. This leads us to the question of whether it is possible to relax the strong independence assumption in the existing random SAT literature.

Relaxation of independence

We first observe that it does not make much sense to study distributions of SAT formulas with arbitrary correlations over the clauses. Indeed, by allowing correlation between several clauses, one may enforce that the random formula ϕ contains large fixed sub-formulas corresponding to NP-hard SAT variants. This would be at odds with our goal of studying average-case complexity. Therefore, we must keep a certain degree of independence in the distribution of instances. We propose to consider the relaxation of mutual independence over m clauses in a random formula ϕ *to k-wise independence* for a small constant k. To keep the new model tractable, we focus on 3-SAT and uniform distribution of literals within each clause. I.e., we assume that (i) every 3-OR-clause c of a random 3-CNF formula ϕ has three literals of three distinct variables drawn uniformly at random among all such triplets of literals

and that (ii) given this marginal distribution of each clause $c \sim F_{\text{uni.}}$, the distribution $\mathcal{D}$ over the clause set C in ϕ is only k-wise independent instead of the mutually independent distribution $\mathcal{D}_{\text{Ind.}} = (F_{\text{uni.}})^{\otimes m}$ in the standard model. This is a natural generalization that has been considered in a number of different settings but, to the best of our knowledge, not in the context of random SAT. Note that the smaller k is, the bigger the set of possible distributions $\mathcal{D}$. Furthermore, for small values of k, a k-wise independent distribution $\mathcal{D}$ can still capture a large class of dependencies among clauses but at the same time does not allow correlation between any k-tuples of clauses. In mathematical terms, the family of discrete k-wise independent distributions naturally appears when we map the set of distributions to the set of their low-degree moments. Specifically, if a distribution $\mathcal{D}$ is supported on the n-dimensional binary cube $\texttt{supp}(D) = \{-1, 1\}^n$, then all its moments of degree up to k can be described as $\mu(D) = (\mathbf{E}[\prod_{i \in S} x_i])_{|S| \le k}$. As low-degree moments (basically, the image of μ) are extremely important in statistical analysis, it is equally important to study the kernel of the aforementioned mapping, which exactly corresponds to the family of k-wise independent distributions. Let us provide additional justifications of our framework by discussing some of the theoretical work on random 3-SAT and on other settings with a similar k-wise independence relaxation.

Pseudo-randomness. Historically, the k-wise relaxation of independence has been actively used in the literature on derandomization and pseudo-randomness, as it allows to significantly reduce the amount of random bits needed to generate random objects. For example, Alon and Nussboim [5] consider random Erdős-Rényi graphs and examine the minimal degree k of independence needed to achieve a variety of graph properties and statistics (such as connectivity, existence of perfect matchings, existence of Hamiltonian cycles, clique and chromatic numbers, etc.) that match those in the mutually independent case. Benjamini et al. [9] consider similar questions for monotone boolean functions. The motivation in [5] comes from the fact that there are efficient constructions of k-wise independent distributions with "low degree of independence" (say $k = O(\log n)$) that utilize only polylog(n) random bits, i.e., much fewer than the polynomial number of random bits required to generate mutually independent distributions. While some of this motivation can be applied to our setting of random 3-SAT, it is a conceptually different story. Indeed, the perspective of pseudo-random generation is through the lenses of "probability theory", where one controls the distributions and can simply choose one that satisfies necessary conditions such as, e.g., $(\log n)$-wise independence. On the other hand, our motivation stems from "statistics", as our ideal model should have a reasonable fit to empirical observations. So, we would like to use as minimal assumptions as possible and study small (constant) degrees of independence.

Refutability of 3-SAT. While the research on lower bounds for random 3-SAT often comes up with certain simple heuristics that efficiently find a satisfying assignment (see the surveys by Achlioptas [2] and Flaxman [26]), it is extremely hard to find an efficient refutation of a unsatisfiable 3-SAT formula. Indeed, the common approach to refute a given SAT formula is proof in resolution. Chvatal and Szemeredi [16] first showed that a random 3-CNF formula with $m = \Theta(n)$ clauses (which is almost surely unsatisfiable) almost surely admits exponential size proof in resolution. Later, Ben-Sasson and Wigderson [8] derived similar result for much larger $m = O(n^{3/2-\varepsilon})$. On the positive side, [28] gave the first polynomial time algorithm via spectral techniques that almost surely[1] refutes a random

[1] Refutation in this case is an algorithm with one-sided error: it always refutes the formula correctly by

3-SAT formula with $m = n^{3/2+\varepsilon}$ clauses. The best known bound on m is due to Feige and Ofek [25] who proved that, for a sufficiently large constant c, random 3-SAT formulas with $m = c \cdot n^{3/2}$ clauses can be almost surely refuted in polynomial time using another spectral graph algorithm. We note that a similar situation (extremely high probability of unsatisfiability for a random formula and inability to efficiently confirm it) is unlikely to happen in our k-clause independent model for constant k. Indeed, the main proof approach for dealing with arbitrary k-wise independent distribution is to define a k-wise statistic, which differentiates any satisfiable formula from a typical unsatisfiable one.

Testing k-wise independence. The property of k-wise independence of a distribution with n components can be tested using $n^{O(k)} = \text{poly}(n)$ many samples in polynomial time, when k is a constant [4, 43]. This is a useful property to have, as it allows one to verify with only polynomially many instances of random 3-SAT, whether these instances conform to k-wise independence or not.

Robust mechanism design. A recent line of work in robust mechanism design also considers families of k-wise independent Bayesian priors in single and multi-unit auctions [13, 22, 33, 34]. Their motivation is similar to ours, as they also rely on the statistical point of view to justify the extension of the results for mutual independent priors typically assumed in Bayesian mechanism design to k-wise independent ones.

1.1 Problem formulation

We consider random 3-CNF formulas with n variables generated from a distribution $\mathcal{D}$ over m clauses, where the mutual independence assumption over clauses is relaxed to k-wise independence. We use the term *k-clause independence* to refer to such distributions. We denote such families of distributions by $\mathcal{F}_k(n, m)$, where each $\mathcal{D} \in \mathcal{F}_k(n, m)$ has identical marginals uniformly distributed over all possible OR-clauses and those marginals are only assumed to be k-wise independent in $\mathcal{D}$. We would like to understand the following question for small values of k:

> How does the satisfiability threshold r_3 of random 3-SAT formulas behave under any k-clause independent distribution $\mathcal{D} \in \mathcal{F}_k(n, m)$?

As the distribution $\mathcal{D}$ is not unique, there might be a large gap between lower and upper estimates of r_3. To this end, we formally define the *lower satisfiability threshold* $\texttt{LST}_k(n)$ as an upper bound on m, such that a random formula ϕ drawn from a distribution in $\mathcal{F}_k(n, m)$ with $m \leq \texttt{LST}_k(n)$ clauses has $\mathbf{Pr}[\phi \text{ is satisfiable}] \geq \frac{2}{3}$. Similarly, the *upper satisfiability threshold* $\texttt{UST}_k(n)$ is a lower bound on m, such that the random formula ϕ with $m \geq \texttt{UST}_k(n)$ clauses has $\mathbf{Pr}[\phi \text{ is satisfiable}] \leq \frac{1}{3}$. What kind of bounds on upper $\texttt{UST}_k(n)$ and lower $\texttt{LST}_k(n)$ thresholds should we expect?

Reasonable expectations

The condition $\mathcal{D} \in \mathcal{F}_k(n, m)$ only says something about configurations of at most k clauses and does not put any other restrictions on the random formula $\phi \sim \mathcal{D}$. As the degree of independence k is a small constant, any argument that gives bounds on $\texttt{LST}_k$ or $\texttt{UST}_k$ can only rely on statistics of at most a constant number of clauses. Hence, it is rather likely that bounds on $\texttt{LST}_k$ and $\texttt{UST}_k$ come together with efficient procedures of, respectively, finding a satisfying assignment for a random formula ϕ, or certifying that ϕ is not satisfiable. Hence, given the prior work on random 3-SAT for $\mathcal{D}_{\text{Ind.}} \in \mathcal{F}_k(n, m)$, we get the following picture:

---

producing certain certificates, or says that the formula might be correct.

Upper satisfiability threshold. The best known result for refuting 3-CNF formulas efficiently is due to Feige and Ofek [25], who show how to do it only for a large number of clauses $m = c \cdot n^{3/2}$. Furthermore, for any smaller number of clauses $m = O(n^{3/2-\varepsilon})$, a random 3-CNF formula is likely to have only exponential in n proof size for any unsatisfiability proof in resolution [8]. Hence, it is out of reach to aim for a better bound on $\mathtt{UST}_k(n)$ than $O(n^{3/2})$ while relying only on k-wise independence for some constant k. In fact, the best known positive result on *efficiently computable proofs* of unsatisfiability in resolution is due to Beame et al. [7], who show that an ordered DLL algorithm executed on a random 3-SAT instance with $m = \Omega(n^2/\log n)$ clauses terminates in polynomial time.

Lower satisfiability threshold. As the proofs for the lower bounds on r_3 often establish simple procedures that find satisfying assignments with high probability, it is still possible that $\mathtt{LST}_k(n)$ is of similar order $\Theta(n)$ as the lower bounds on r_3 for $\mathcal{D}_{\text{Ind.}}$. Thus, the most ambitious result would be to show that $\mathtt{LST}_k(n) \le c_k \cdot n$ for constant c_k that increases with k. A more modest goal is to aim for $\mathtt{LST}_k(n) = o(n)$ for a constant k, where $\mathtt{LST}_k(n) \to \Theta(n)$ as $k \to +\infty$.

1.2 Our results

We obtain the following bounds on the upper and lower satisfiability thresholds $\mathtt{UST}_k(n)$ and $\mathtt{LST}_k(n)$ for various values of k.

Upper satisfiability thresholds

We first consider small degrees of independence, i.e., $k \in \{2, 3\}$. In both cases, we show that $\mathtt{UST}_k(n) = \Theta(n^3)$, meaning that one needs almost all possible clauses in a 3-clause (as well as 2-clause) independent formula to ensure that it is unsatisfiable (see **Theorem 4** and **Theorem 7**). The most nontrivial part is to construct the distribution $\mathcal{D} \in \mathcal{F}_3(n, m)$ with $m = \Theta(n^3)$ and $\mathbf{Pr}[\phi \text{ is satisfiable}] \ge \frac{2}{3}$. Our construction is based on "3-XOR formulas" (i.e., OR-clauses that have either one or three literals that are satisfied by a randomly planted truth assignment), which aligns well with the intuition developed in previous work [24, 25]. The main technical difficulty is to ensure k-clause independence by adding a small fraction of unsatisfiable instances and checking all k-wise statistics.

Our most exciting and technically involved result (see **Theorem 8**) is our proof that $\mathtt{UST}_4(n) = O(n^2)$, i.e., a random formula $\phi \sim \mathcal{D}$ with $m = \Omega(n^2)$ clauses is unsatisfiable with large probability for any 4-clause independent distribution $\mathcal{D} \in \mathcal{F}_4(n, m)$. It is worth noting that such a bound is much harder to get under the 4-wise independence assumption than in the case of a mutually independent distribution $\mathcal{D}_{\text{Ind.}}$. Indeed, Feige and Ofek [25] describe a very simple refutation algorithm for $m = \Theta(n^2)$ that fixes a variable x and considers all clauses containing x or $\overline{x}$ (there will be $\Theta(n)$ such clauses in expectation). Then, after deleting x (or $\overline{x}$), one can reduce the problem to the refutation of the respective random 2-CNF sub-instance, which can be easily verified in polynomial time and has a low satisfiability threshold of $r_2 = 1$. This simple approach obviously fails for 4-clause independent distributions. We instead construct a bipartite multigraph $G(\phi)$ between pairs of distinct literals on one side and all singleton literals on the other, in which every OR-clause in ϕ corresponds to three different edges. We then carefully examine the statistic $\kappa(\phi)$ that counts $K_{2,2}$ subgraphs in $G(\phi)$ for a random $\phi \sim \mathcal{D}$. We find that the expected value of $\kappa(\phi)$ for random ϕ is only slightly larger than its absolute minimal value, while at the same time $\kappa(\phi)$ is significantly larger than its expectation when ϕ is satisfiable. Our argument

bears certain similarities with the argument in [25], which also looked at intersections of two literals between pairs of clauses but used the 3-XOR principle and a differently constructed non-bipartite graph.

Lower satisfiability thresholds

We show (in **Theorem 14**) that $\texttt{LST}_k(n) \geq \Omega(n^{1-1/k})$ for any $k \geq 2$. I.e., any k-clause independent random formula is satisfiable with high probability if it contains at most $O(n^{1-1/k})$ clauses. The argument is simple: for any k-clause independent distribution, we look at the 3-uniform hypergraph that corresponds to the variables of a random formula ϕ produced according to this distribution, and argue that this graph does not have Berge-cycles, with high probability. We also provide an informal justification that this bound is asymptotically tight, i.e., that $\texttt{LST}_k(n) = O(n^{1-1/k})$. Specifically, we outline a plausible approach for constructing a k-clause independent distribution with $m = O(n^{1-1/k})$ clauses such that most of its formulas are unsatisfiable. Our approach is built upon existing constructions of dense hyper-graphs with large girth. It is interesting to note that, in both the proof of the $\texttt{LST}_k(n) = \Omega(n^{1-1/k})$ result and the approach for showing that $\texttt{LST}_k(n) = O(n^{1-1/k})$, we only need to consider variables and can completely ignore the distribution over the literals.

2 Preliminaries

Let $x_1, x_2, \cdots, x_n$ be n boolean variables. A literal ℓ is a boolean variable or the negation of it. For convenience, we usually represent a literal as a variable-sign pair, i.e., $\ell = (x_i, s)$, where s is the positive sign $\texttt{+}$ if the literal is a boolean variable and the negative sign $\texttt{-}$ if it is its negation. We define $\Sigma(n) = \{\texttt{+}, \texttt{-}\}^n$. An instance of the *Satisfiability* problem in conjunctive normal form (or *SAT instance*, for short) is a boolean formula over a subset of the n variables which is a conjunction of disjunctive clauses, each clause containing literals with different variables. In a 3-SAT instance, every clause has exactly 3 literals. Each SAT instance C can be described as a multiset of clauses. We denote the size of an instance C as $m = |C|$.

Given the number of variables n, let $X(n)$ be the set of variables. Let $T(n)$ be the set of all unordered triplets of variables $T(n) = \{(x_i, x_j, x_k) \mid x_i, x_j, x_k \in X(n), i < j < k\}$ and $\mathcal{T}(n)$ be the set of all possible clauses, i.e., all triplets with literals of different variables. We will usually omit the dependency of X, T, and $\mathcal{T}$ on n, when the value of n is clear from the context. To refer to the variable (the set of variables) or the sign (the set of signs) of a literal (a clause $c \in C$), we define operators $\text{Var}(\cdot)$ and $\text{Sign}(\cdot)$, so that if, e.g., $c = ((x_1, \texttt{+}), (x_2, \texttt{-}), (x_3, \texttt{+}))$, then $\text{Var}(c) = (x_1, x_2, x_3)$ and $\text{Sign}(c) = (\texttt{+}, \texttt{-}, \texttt{+})$.

A *truth assignment* is a vector $\sigma \in \{0, 1\}^n$, where 1 or 0 at the σ_i coordinate corresponds to the "true" or "false" value of x_i, respectively. A truth assignment σ is *satisfying* for an instance C when each clause $c \in C$ has at least one true literal under σ. An instance C is *satisfiable* if there exists a satisfying assignment; otherwise, C is *unsatisfiable*.

The random 3-SAT model assumes that the instance is drawn from a probability distribution over all possible 3-SAT instances with m clauses and n variables. The main object of study in such a model is the probability of such a random instance being satisfiable as a function of n and m.

2.1 Random 3-SAT without mutual independence

In the standard random 3-SAT model, each clause is drawn uniformly at random among all clauses in $\mathcal{T}(n)$, independently from the other clauses. A constructive definition is given by Model 1. We denote the distribution over instances generated by Model 1 as $\mathcal{D}_{\text{Ind.}}$.

MODEL 1 Selects a 3-SAT instance uniformly at random from all possible instances.

Input: Integers $n \geq 3$ and $m \geq 1$
Output: A 3-SAT instance with n variables and m clauses
$C \leftarrow \emptyset$;
for $l \leftarrow 1, \ldots, m$ **do**
  pick a literal triplet (ℓ_1, ℓ_2, ℓ_3) uniformly at random from $\mathcal{T}(n)$;
  $c \leftarrow (\ell_1, \ell_2, \ell_3)$;
  $C \leftarrow C \cup \{c\}$;
end
return C;

Our main focus will be on k-wise relaxations of the independent distribution.

▶ **Definition 1** (k-clause independent random SAT). *A distribution $\mathcal{D}$ for selecting a random SAT instance is k-clause independent if the set of any fixed k clauses $S \subseteq C$ is distributed uniformly at random over all k-tuples of possible clauses, i.e.,*

$$\mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[c_i = t_i, \forall i \in [k]] = \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}_{Ind.}}[c_i = t_i, \forall i \in [k]] = \prod_{i\in[k]} \mathbf{Pr}[c_i = t_i].$$

for every $c_1, \ldots, c_k \in C$ and $t_1, \ldots, t_k \in \mathcal{T}$. We denote the family of all k-clause independent distributions with m clauses over n variables as $\mathcal{F}_k(n, m)$.

We remark that, in contrast to the random 3-SAT model (Model 1), which defines a single distribution for given n and m, the family $\mathcal{F}_k(n, m)$ contains many different distributions.

By definition, the probability of any event A_S that depends only on a subset S of at most k clauses in the k-clause independent distribution $\mathcal{D}$ is the same as for $\mathcal{D}_{\text{Ind.}}$. I.e., the expectations of the indicator function $\mathbb{I}[A_S]$ are the same for $\mathcal{D}$ and $\mathcal{D}_{\text{Ind.}}$. Hence, by linearity of expectation, any statistic that involves only $k' \leq k$ clauses must be the same for $\mathcal{D}$ and $\mathcal{D}_{\text{Ind.}}$, i.e.,

$$\mathop{\mathbf{E}}_{C\sim\mathcal{D}}\left[\sum_{\substack{S\subseteq C,\\ |S|=k'}} \mathbb{I}\left[A_S\right]\right] = \mathop{\mathbf{E}}_{C\sim\mathcal{D}_{\text{Ind.}}}\left[\sum_{\substack{S\subseteq C,\\ |S|=k'}} \mathbb{I}\left[A_S\right]\right]. \tag{1}$$

We would like to understand what are the largest/smallest possible number of clauses for a random 3-SAT formula to be almost surely satisfiable/unsatisfiable under any k-clause independent distribution. Using $SAT(C)$ to denote the event that the SAT instance C is satisfiable, we define the following satisfiability "thresholds".

▶ **Definition 2** (Upper satisfiability threshold). *The upper satisfiability threshold $\texttt{UST}_k(n)$ is defined as follows. For integer $n \geq 3$, $\texttt{UST}_k(n)$ is the minimum integer m such that*

$$\mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[SAT(C)] \leq 1/3, \quad \forall \mathcal{D} \in \mathcal{F}_k(n, m).$$

▶ **Definition 3** (Lower satisfiability threshold). *The lower satisfiability threshold $\mathtt{LST}_k(n)$ is defined as follows. For integer $n \geq 3$, $\mathtt{LST}_k(n)$ is the maximum integer m such that*

$$\mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[\neg SAT(C)] \leq 1/3, \quad \forall \mathcal{D} \in \mathcal{F}_k(n,m).$$

Extending the line of research on the standard random 3-SAT model, we would like to have as tight estimates of $\mathtt{UST}_k(n)$ and $\mathtt{LST}_k(n)$ as possible.

3 Tight bounds for the upper satisfiability threshold $\mathtt{UST}_2(n)$

We begin with a technical warm up and prove asymptotically tight bounds on the upper satisfiability threshold of 2-clause independent random 3-SAT.

▶ **Theorem 4.** $\mathtt{UST}_2(n) = \Theta(n^3)$.

Theorem 4 follows by the next two lemmas. Lemma 5 provides an upper bound on $\mathtt{UST}_2(n)$. In the proof, we introduce the technique that we will use later in Section 5 to get a much more involved upper bound on $\mathtt{UST}_4(n)$.

▶ **Lemma 5.** *For any $\mathcal{D} \in \mathcal{F}_2(n,m)$ with $m \geq 56\binom{n}{3}$, $\mathbf{Pr}_{C\sim\mathcal{D}}[SAT(C)] \leq 56\binom{n}{3}/m$.*

Proof. Let $\xi(C) \stackrel{\text{def}}{=\!=} \sum_{c_i,c_j\in C} \mathbb{I}[c_i = c_j]$ be the number of identical clause pairs in the instance C. On the one hand, Equation (1) implies that the expectation of $\xi(C)$ is the same when C is drawn from a 2-clause independent distribution and $\mathcal{D}_{\text{Ind.}}$. On the other hand, if an instance C has a satisfying assignment, at least $1/8$ of possible clauses from $\mathcal{T}$ must not appear in C, which means that the value of $\xi(C)$ is significantly higher than its expectation. These two observations will allow us to get the desired bound on the probability $\mathbf{Pr}_{C\sim\mathcal{D}}[SAT(C)]$.

Let $\mathcal{D} \in \mathcal{F}_2(n,m)$ and define $p \stackrel{\text{def}}{=\!=} \mathbf{Pr}_{C\sim\mathcal{D}}[SAT(C)]$. We shall derive two lower bounds on the value the random variable $\xi(C)$ can get: an unconditional lower bound $\mathtt{LB}[\xi]$ (not far from $\mathbf{E}_{C\sim\mathcal{D}}[\xi(C)]$) and a lower bound $\mathtt{LB}[\xi \mid SAT]$ on $\xi(C)$ when C is satisfiable (this will be significantly larger than $\mathbf{E}_{C\sim\mathcal{D}}[\xi(C)]$). We note that

$$\begin{aligned}\mathbf{E}_{C\sim\mathcal{D}}[\xi(C)] &= \mathbf{Pr}[SAT(C)] \cdot \mathbf{E}[\xi(C) \mid SAT(C)] \\ &\quad + \mathbf{Pr}[\neg SAT(C)] \cdot \mathbf{E}[\xi(C) \mid \neg SAT(C)] \\ &\geq p \cdot \mathtt{LB}[\xi \mid SAT] + (1-p) \cdot \mathtt{LB}[\xi]. \end{aligned} \tag{2}$$

We next derive $\mathbf{E}_{C\sim\mathcal{D}}[\xi(C)]$, $\mathtt{LB}[\xi]$, and $\mathtt{LB}[\xi \mid SAT]$. We denote as $M \stackrel{\text{def}}{=\!=} |\mathcal{T}| = 8\binom{n}{3}$ the number of possible clauses and $\lambda \stackrel{\text{def}}{=\!=} m/M > 1$. First, we observe that

$$\mathbf{E}_{C\sim\mathcal{D}}[\xi(C)] = \sum_{c_i,c_j\in C} \mathbf{E}\left[\mathbb{I}\left[c_i = c_j\right]\right] = \frac{m^2-m}{2} \cdot \mathbf{Pr}[c_i = c_j] = \frac{m^2-m}{2\cdot M} = m \cdot \frac{\lambda \cdot M - 1}{2 \cdot M}. \tag{3}$$

To derive the lower bounds $\mathtt{LB}[\xi]$ and $\mathtt{LB}[\xi \mid SAT]$, we observe that any given clause type $t \in \mathcal{T}$ contributes $\binom{d_t}{2}$ pairs to $\xi(C)$, where $d_t \stackrel{\text{def}}{=\!=} \sum_{c\in C} \mathbb{I}[c = t]$ is the number of type t clauses in C. I.e.,

$$\xi(C) = \sum_{t\in\mathcal{T}} \frac{d_t^2 - d_t}{2}, \qquad \text{where } \sum_{t\in\mathcal{T}} d_t = m. \tag{4}$$

The minimum of $\xi(C) = \frac{1}{2}\sum d_t^2 - \frac{m}{2}$ under the constraint $\sum d_t = m$ is achieved when all the d_t variables are equal, i.e., equal to $\frac{m}{|\mathcal{T}|} = \lambda$. Thus, we get the following lower bound $\texttt{LB}[\xi]$ on $\xi(C)$:

$$\xi(C) \geq \texttt{LB}[\xi] \stackrel{\text{def}}{=\!=} \frac{1}{2}\sum_{t\in\mathcal{T}} \lambda^2 - \frac{m}{2} = m\cdot\frac{\lambda-1}{2}. \tag{5}$$

To derive the lower bound $\texttt{LB}[\xi \mid SAT]$ on $\xi(C)$ for a satisfiable formula C, we note that at least a $\frac{1}{8}$-fraction of the clause types are not present in C. I.e., at least $\frac{M}{8}$ of the d_t variables have value equal to 0 in (4). Similarly to (5), the minimum value of $\xi(C)$ is achieved when all the remaining $\frac{7M}{8}$ d_t variables are equal to each other, getting the value $\frac{8\cdot m}{7\cdot M}$. That is,

$$\xi(C) \geq \texttt{LB}[\xi \mid SAT] \stackrel{\text{def}}{=\!=} \frac{7\cdot M}{8}\cdot\left(\frac{8\lambda}{7}\right)^2 - \frac{m}{2} = m\cdot\frac{\frac{8}{7}\lambda-1}{2}. \tag{6}$$

We plug the bounds (3), (5), and (6) into (2) to get

$$m\cdot\frac{\lambda\cdot M-1}{2\cdot M} \geq p\cdot m\cdot\frac{\frac{8}{7}\lambda-1}{2} + (1-p)\cdot m\cdot\frac{\lambda-1}{2}.$$

After simple algebraic transformation, this is equivalent to the inequality $1-\frac{1}{M} \geq \frac{p}{7}\lambda$, which implies that $p \leq \frac{7\cdot M}{m} = 56\binom{n}{3}/m$. ◀

To lower-bound $\texttt{UST}_2(n)$, we use a specific 2-clause independent distribution, which is depicted as Model 2. The next lemma proves the correctness of our construction. The proof is deferred to the full version.

▶ **Lemma 6.** *Model 2 defines a 2-clause independent probability distribution that generates 3-SAT instances of size $\Omega(n^3)$ that are satisfiable with probability at least $1-O(n^{-3})$.*

MODEL 2 Selects a 3-SAT instance according to a 2-clause independent distribution.

Input: Integer $n \geq 3$
Output: A 3-SAT instance C with n variables and $m = \binom{n}{3}$ clauses

- With probability $1-\binom{n}{3}^{-1}$, construct a satisfiable instance C:
  1. Match $m = \binom{n}{3}$ clauses to all different $T(n)$ variable triplets uniformly at random;
  2. Pick a random truth assignment $\sigma \sim \texttt{Uni}[\{0,1\}^n]$;
  3. For each clause $c \in C$ matched to the variable triplet $(x_i, x_j, x_k) \in T(n)$
     - Pick a random sign triplet (s_1, s_2, s_3) of the same parity with $(\sigma_i, \sigma_j, \sigma_k)$ (i.e., $\mathbb{I}[s_1 = \texttt{-}] + \mathbb{I}[s_2 = \texttt{-}] + \mathbb{I}[s_3 = \texttt{-}] + \sigma_i + \sigma_j + \sigma_k = 0 \mod 2$);
     - Let clause $c \leftarrow ((x_i, s_1), (x_j, s_2), (x_k, s_3))$;
- With probability $\binom{n}{3}^{-1}$, sample C from distribution $\mathcal{D}_{\texttt{uni-var}}$ defined as follows:
  1. Pick a random **single** variable triplet $(x_i, x_j, x_k) \sim \texttt{Uni}[T(n)]$;
  2. For each clause $c \in C$
     - Pick a random sign triplet $(s_1, s_2, s_3) \sim \texttt{Uni}[\{\texttt{+},\texttt{-}\}^3]$;
     - Let clause $c \leftarrow ((x_i, s_1), (x_j, s_2), (x_k, s_3))$;

4 A tight lower bound for $\texttt{UST}_3(n)$

Clearly, Lemma 5 also provides an upper bound on $\texttt{UST}_3(n)$. The proof of the next statement follows by presenting a matching lower bound through Model 3.

▶ **Theorem 7.** *$\texttt{UST}_3(n) = \Theta(n^3)$.*

MODEL 3 Selects a 3-SAT instance according to a 3-clause independent distribution.

Input: Integer $n \geq 3$ and even integer $m \leq \frac{1}{3} \cdot \binom{n}{3}$.
Output: A 3-SAT instance C with n variables and m clauses.
Set $p \stackrel{\text{def}}{=\!=} (m-1) \cdot \left(\binom{n}{3}^{-1} - \frac{1}{3}\binom{n}{3}^{-2} \right)$ and $q \stackrel{\text{def}}{=\!=} \binom{n}{3}^{-2}$;

- With probability $1-p-q$, construct a satisfiable instance C:
  1. Match C clauses to m different variable triplets in $T(n)$ uniformly at random;
  2. Pick a random truth assignment $\sigma \sim \texttt{Uni}[\{0,1\}^n]$;
  3. For each clause $c \in C$ matched to the variable triplet $(x_i, x_j, x_k) \in T(n)$
     - Pick a random sign triplet (s_1, s_2, s_3) of the same parity with $(\sigma_i, \sigma_j, \sigma_k)$ (i.e., $\mathbb{I}\left[s_1 = \texttt{-}\right] + \mathbb{I}\left[s_2 = \texttt{-}\right] + \mathbb{I}\left[s_3 = \texttt{-}\right] + \sigma_i + \sigma_j + \sigma_k = 0 \mod 2$);
     - Let clause $c \leftarrow ((x_i, s_1), (x_j, s_2), (x_k, s_3))$;
- With probability p, construct an instance C with $m/2$ different variable triplets:
  1. Uniformly at random match C to $\frac{m}{2}$ different variable triplets in $T(n)$ (exactly 2 clauses in C per one variable triplet);
  2. For each clause $c \in C$ assigned to the variable triplet $(x_i, x_j, x_k) \in T(n)$
     - Pick a random sign triplet $(s_1, s_2, s_3) \sim \texttt{Uni}[\{\texttt{+}, \texttt{-}\}^3]$;
     - Let clause $c \leftarrow ((x_i, s_1), (x_j, s_2), (x_k, s_3))$;
- With probability q, sample C from distribution $C \sim \mathcal{D}_{\texttt{uni-var}}$ as follows:
  1. Pick a random single variable triplet $(x_i, x_j, x_k) \sim \texttt{Uni}[T(n)]$;
  2. For each clause $c \in C$
     - Pick a random sign triplet $(s_1, s_2, s_3) \sim \texttt{Uni}[\{\texttt{+}, \texttt{-}\}^3]$;
     - Let clause $c \leftarrow ((x_i, s_1), (x_j, s_2), (x_k, s_3))$;

We give an explicit construction (see Model 3) of a 3-clause independent distribution $\mathcal{D} \in \mathcal{F}_3(n, m)$ with n variables and (an even number of) $m \leq \frac{1}{3} \cdot \binom{n}{3}$ clauses, such that $\mathbf{Pr}_{C \sim \mathcal{D}}[SAT(C)] \geq 2/3 - O(n^{-6})$. Our construction in Model 3 follows the same pattern as in Model 2, but uses an additional step and minor modifications to ensure 3-clause independence.[2] The proof is deferred to the full version.

5 An upper bound for $\texttt{UST}_4(n)$

Our next result is rather surprising as it indicates that 4-wise independence allows for a steep decrease in the upper satisfiability threshold compared to 2- and 3-wise independence.

▶ **Theorem 8.** $\mathit{UST}_4(n) = O(n^2)$.

Proof. We will prove the theorem by showing that for any positive integers n and m and any 4-clause independent probability distribution $\mathcal{D} \in \mathcal{F}_4(n, m)$, it holds $\mathbf{Pr}_{C \sim \mathcal{D}}[SAT(C)] \leq O\left(\max\left\{\frac{n^2}{m}, \frac{1}{\sqrt{n}}\right\}\right)$. The claim is obvious for $n < 10$. We will assume that $n \geq 10$ and $m \leq \sqrt{10} \cdot n^{5/2}$, and will show that $\mathbf{Pr}_{C \sim \mathcal{D}}[SAT(C)] \leq \frac{4288 \cdot n^2}{m}$ for every 4-clause independent probability distribution $\mathcal{D} \in \mathcal{F}_4(n, m)$. Note that for $m > \sqrt{10} \cdot n^{5/2}$, the probability bound of $O\left(\frac{1}{\sqrt{n}}\right)$ follows by selecting uniformly at random a subset of $\sqrt{10} \cdot n^{5/2}$ clauses.

We will use a graph representation of 3-SAT instances defined as follows. Given a 3-SAT instance C consisting of m clauses over n variables, the bipartite multi-graph $G(C) = (L \cup R, E)$ has a node corresponding to each (unordered) pair of literals $\{\ell_1, \ell_2\}$ from different variables at the left node side L and a node corresponding to each literal ℓ at the right node

[2] The first and third block of Model 3 correspond to the two blocks of Model 2. The only difference in the first block is that the matching of C is not to all the variable triplets in $T(n)$ but only to m of them.

side R. Hence, $|L| = 4\binom{n}{2}$ and $|R| = 2n$. For every clause $c = (\ell_1, \ell_2, \ell_3)$ of C, $G(C)$ has the three edges between the node corresponding to the pair of literals (ℓ_i, ℓ_j) and the node corresponding to literal ℓ_{6-i-j} for $(i,j) \in \{(1,2),(1,3),(2,3)\}$.

The main proof idea of Theorem 8 is to analyze the statistic $\kappa(C)$ defined as the number of distinct $K_{2,2}$ subgraphs in graph $G(C)$. Namely, we first derive an upper bound on the expectation $\mathbf{E}_{C\sim\mathcal{D}}[\kappa(C)]$ (see Lemma 9) when C is drawn from the 4-clause independent probability distribution $\mathcal{D} \in \mathcal{F}_4(n,m)$. We then give two lower bounds on the value of the random variable $\kappa(C)$ by considering an underlying simple subgraph of $G(C)$. Note that the underlying simple subgraph corresponds to a smaller instance $\widetilde{C}$ with $\widetilde{m}$ distinct clauses, in which we remove all duplicated clauses in C. As we show in Lemma 10, this does not significantly reduce the size of the instance. Our first lower bound (Lemma 11) on $\kappa(\widetilde{C})$ holds for any instance $\widetilde{C}$ and is very close to the upper bound on the expectation of $\kappa(C)$. On the other hand, when $\widetilde{C}$ is satisfiable, we manage to give a significantly stronger lower bound on $\kappa(\widetilde{C})$ in Lemma 12. We then conclude the proof of Theorem 8 by relating all these upper and lower bounds with $\mathbf{E}_{C\sim\mathcal{D}}[\kappa(C)]$ in Lemma 13. We defer the proofs of Lemmas 9, 10, 11, and 12 to the full version.

▶ **Lemma 9.** *For any $\mathcal{D} \in \mathcal{F}_4(n,m)$, it holds that $\mathbf{E}_{C\sim\mathcal{D}}[\kappa(C)] \le \frac{81m^4}{64n^6} + \frac{729m^3}{32n^4}$.*

▶ **Lemma 10.** $\mathbf{E}_{C\sim\mathcal{D}}[\widetilde{m}^4] \ge m^4 - \frac{125}{6}\cdot m^3 \cdot n^2$.

▶ **Lemma 11.** *Let C be an instance with n variables and $\widetilde{m}$ distinct clauses. Then,*

$$\kappa(\widetilde{C}) \ge \frac{81\widetilde{m}^4}{64n^6} - \frac{27\widetilde{m}^3}{8n^4}.$$

▶ **Lemma 12.** *Let C be a statisfiable instance with n variables and $\widetilde{m}$ distinct clauses. Then*

$$\kappa(\widetilde{C}) \ge \frac{82\widetilde{m}^4}{64n^6} - \frac{123\widetilde{m}^3}{16n^4}.$$

▶ **Lemma 13.** *For any $\mathcal{D} \in \mathcal{F}_4(n,m)$, we have $\mathbf{E}_{C\sim\mathcal{D}}[\kappa(C)] \ge \frac{81m^4}{64n^6} - \frac{1215m^3}{32n^4} + \frac{m^4}{64n^6} \cdot \mathbf{Pr}_{C\sim\mathcal{D}}[SAT(C)]$.*

Proof. We prove the lemma with the following derivation:

$$\begin{aligned}
\mathop{\mathbf{E}}_{C\sim\mathcal{D}}[\kappa(C)] &\ge \mathop{\mathbf{E}}_{C\sim\mathcal{D}}\left[\kappa(\widetilde{C})\right] \\
&= \mathop{\mathbf{E}}_{C\sim\mathcal{D}}\left[\kappa(\widetilde{C})|SAT(C)\right]\cdot \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[SAT(C)] \\
&\quad + \mathop{\mathbf{E}}_{C\sim\mathcal{D}}\left[\kappa(\widetilde{C})|\neg SAT(C))\right]\cdot \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[\neg SAT(C)] \\
&\ge \mathop{\mathbf{E}}_{C\sim\mathcal{D}}\left[\frac{82\widetilde{m}^4}{64n^6} - \frac{123\widetilde{m}^3}{16n^4}\middle| SAT(C)\right]\cdot \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[SAT(C)] \\
&\quad + \mathop{\mathbf{E}}_{C\sim\mathcal{D}}\left[\frac{81\widetilde{m}^4}{64n^6} - \frac{27\widetilde{m}^3}{8n^4}\middle|\neg SAT(C)\right]\cdot \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[\neg SAT(C)] \\
&\ge \mathop{\mathbf{E}}_{C\sim\mathcal{D}}\left[\frac{82\widetilde{m}^4}{64n^6} - \frac{123\widetilde{m}^3}{16n^4}\right] - \mathop{\mathbf{E}}_{C\sim\mathcal{D}}\left[\frac{\widetilde{m}^4}{64n^6}\middle|\neg SAT(C)\right]\cdot \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[\neg SAT(C)] \\
&\ge \frac{82}{64n^6}\mathop{\mathbf{E}}_{C\sim\mathcal{D}}[\widetilde{m}^4] - \frac{123m^3}{16n^4} - \frac{m^4}{64n^6}\cdot \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[\neg SAT(C)] \\
&\ge \frac{82m^4}{64n^6} - \frac{861m^3}{32n^4} - \frac{123m^3}{16n^4} - \frac{m^4}{64n^6}\cdot \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[\neg SAT(C)]
\end{aligned}$$

$$= \frac{81m^4}{64n^6} - \frac{1107m^3}{32n^4} + \frac{m^4}{64n^6} \cdot \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[SAT(C)]\,,$$

as desired. The second inequality follows by Lemmas 12 and 11, the fourth one by the fact $\widetilde{m} \le m$, and the fifth one by Lemma 10. ◀

Now, Lemmas 9 and 13 yield

$$\frac{81m^4}{64n^6} - \frac{1215m^3}{32n^4} + \frac{m^4}{64n^6} \cdot \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[SAT(C)] \le \mathop{\mathbf{E}}_{C\sim\mathcal{D}}[\kappa(C)] \le \frac{81m^4}{64n^6} + \frac{729m^3}{32n^4},$$

which implies the desired bound $\mathbf{Pr}_{C\sim\mathcal{D}}[SAT(C)] \le \frac{4288\cdot n^2}{m}$. Theorem 8 follows. ◀

6 Bounds on the lower satisfiability threshold

In this section, we present our upper bound on the lower satisfiability threshold and explain why we believe that it is the tight bound for every degree of independence.

▶ **Theorem 14.** *For every integer $k \ge 2$, $\mathtt{LST}_k(n) = \Omega(n^{1-1/k})$.*

Proof. We prove the theorem by showing that for every integers $n \ge 3$ and $m \le \frac{1}{12}\cdot n^{1-1/k}$, any k-clause independent distribution $\mathcal{D} \in \mathcal{F}_k(n,m)$ satisfies $\mathbf{Pr}_{C\sim\mathcal{D}}[\neg SAT(C)] \le 1/3$.

Let $G(C) = (V,E)$ be a 3-uniform hypergraph defined by an instance C drawn from a k-clause independent distribution $\mathcal{D}$, $V = X(n)$ and $E = \{\mathrm{Var}(c)\}_{c\in C}$, i.e., every node in G represents a variable and every hyperegde in G represents the variable triplet of a clause. We consider simple *Berge-cycles* (or, simply, cycles) of length $\ell \ge 2$. A cycle of length $\ell > 2$ is defined as a set of ℓ distinct hyperedges $e_1, e_2, \ldots, e_\ell \in E$ such that pairs of consecutive edges share exactly one vertex ($|e_i \cap e_{i+1}| = 1$ and $|e_\ell \cap e_1| = 1$) and all other pairs of hyperedges are disjoint. In a cycle of length $\ell = 2$, the two hyperedges e_1 and e_2 have at least two vertices in common. We similarly define a path of length $\ell \ge 2$, where pairs of consecutive edges share exactly one vertex ($|e_i \cap e_{i+1}| = 1$ for $i \in [\ell-1]$) and all other pairs of hyperedges are disjoint. We observe that any unsatisfiable instance must contain a subgraph H in G such that every variable in H appears in at least two hyperedges of H, which in turn implies that G has a cycle. That translates into the following upper bound on the probability that instance $C \sim \mathcal{D}$ is not satisfiable:

$$\begin{aligned}
\mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[\neg SAT(C)] &\le \mathop{\mathbf{Pr}}_{C\sim\mathcal{D}}[\exists \text{ cycle in } G(C)] \le \sum_{\ell\ge 2} \mathop{\mathbf{E}}_{C\sim\mathcal{D}}\left[\mathsf{Cycle}_\ell(G(C))\right] \\
&\le \mathop{\mathbf{E}}_{C\sim\mathcal{D}}[\mathsf{Path}_k(G(C))] + \sum_{\ell=2}^{k-1} \mathop{\mathbf{E}}_{C\sim\mathcal{D}}\left[\mathsf{Cycle}_\ell(G(C))\right] \\
&= \mathop{\mathbf{E}}_{C\sim\mathcal{D}_{\mathrm{Ind.}}}[\mathsf{Path}_k(G(C))] + \sum_{\ell=2}^{k-1} \mathop{\mathbf{E}}_{C\sim\mathcal{D}_{\mathrm{Ind.}}}\left[\mathsf{Cycle}_\ell(G(C))\right]. \qquad (7)
\end{aligned}$$

In the above derivation, $\mathsf{Cycle}_\ell(G(C))$ counts the number of distinct cycles of length ℓ in G, and $\mathsf{Path}_k(G(C))$ counts the number of distinct paths of lengths k in G. For each ordered tuple of k hyperedges in G (there are $k!\cdot\binom{m}{k}$ such orderings), we can easily calculate the probability that they form a path. Thus,

$$\mathop{\mathbf{E}}_{C\sim\mathcal{D}_{\mathrm{Ind.}}}[\mathsf{Path}_k(G(C))] = k!\cdot\binom{m}{k}\cdot\frac{3\cdot\binom{n-3}{2}}{\binom{n}{3}}\cdot\prod_{t=3}^{k}\frac{2\cdot\binom{n-2t+1}{2}}{\binom{n}{3}} \le m^k\cdot\frac{9}{n}\cdot\left(\frac{6}{n}\right)^{k-2}. \qquad (8)$$

For each ordering of k edges, the first edge e_1 can be chosen arbitrarily; the second edge has exactly one vertex in common with e_1 and the remaining two vertices are chosen from $[n] \setminus e_1$ vertices; each of the remaining edges e_t for $t \geq 3$ has exactly one of the two vertices of e_{t-1} different from $e_{t-1} \cap e_{t-2}$ and has two other vertices chosen from $[n] \setminus (e_1 \cup \ldots \cup e_{t-1})$. The upper bound follows after simplifying each of the terms. We similarly derive an upper bound on $\mathbf{E}[\mathsf{Cycle}_\ell(G(C))]$ for $\ell \geq 3$ as follows:

$$\mathop{\mathbf{E}}_{C \sim \mathcal{D}_{\text{Ind.}}} \left[\mathsf{Cycle}_\ell(G(C))\right] \leq \frac{1}{6} \cdot \ell! \cdot \binom{m}{\ell} \cdot \left[\frac{3 \cdot \binom{n-3}{2}}{\binom{n}{3}} \cdot \prod_{t=3}^{\ell-1} \frac{2 \cdot \binom{n-2t+1}{2}}{\binom{n}{3}}\right] \cdot \frac{4 \cdot (n-2\ell+1)}{\binom{n}{3}},$$

where for each ordering of ℓ edges (now $2\ell \geq 6$ orderings correspond to the same cycle), the probabilities to choose the first $\ell - 1$ edges can be computed in the same way as for Path_k; for the last hyper-edge e_ℓ, there are $4 = 2 \cdot 2$ ways to select a vertex of $e_{\ell-1}$ together with a vertex of e_1, and $n - 2\ell + 1$ choices for the new vertex in $[n] \setminus (e_1 \cup \ldots \cup e_{\ell-1})$. Hence,

$$\mathop{\mathbf{E}}_{C \sim \mathcal{D}_{\text{Ind.}}} \left[\mathsf{Cycle}_\ell(G(C))\right] \leq m^\ell \cdot \left[\frac{9}{n} \cdot \left(\frac{6}{n}\right)^{\ell-3}\right] \cdot \frac{4}{n^2} = \frac{1}{6}\left(\frac{6m}{n}\right)^\ell. \tag{9}$$

For $\ell = 2$ we have

$$\mathop{\mathbf{E}}_{C \sim \mathcal{D}_{\text{Ind.}}} \left[\mathsf{Cycle}_2(G(C))\right] = \binom{m}{2} \cdot \frac{3 \cdot (n-3) + 1}{\binom{n}{3}} = \frac{3m \cdot (m-1) \cdot (3n-8)}{n \cdot (n-1) \cdot (n-2)} \leq \left(\frac{3m}{n}\right)^2. \tag{10}$$

We conclude the proof by plugging estimates (8),(9), and (10) into (7).

$$\begin{aligned}\mathbf{Pr}\left[\neg SAT(C)\right] &\leq \frac{2}{3}\frac{(6m)^k}{n^{k-1}} + \left(\frac{3m}{n}\right)^2 + \frac{1}{6}\sum_{\ell=3}^{k-1}\left(\frac{6m}{n}\right)^\ell \\ &\leq \frac{2}{3 \cdot 2^k} + \frac{1}{16} + \frac{1}{6} \cdot \left(\frac{1}{2^3} + \frac{1}{2^4} + \ldots + \frac{1}{2^{k-1}}\right) < \frac{1}{3}.\end{aligned}$$

The second inequality follows since $m \leq \frac{1}{12} \cdot n^{1-1/k}$. ◀

6.1 On the tightness of the $O(n^{1-1/k})$ bound for $\mathtt{LST}_k(n)$

Theorem 14 says that k-wise independence of $\mathcal{D} \in \mathcal{F}_k(n, m)$ is enough to guarantee satisfiability of a random formula $C \sim \mathcal{D}$ for the number of clauses m of order $n^{1-1/k}$. On the other hand, for the mutually independent distribution $\mathcal{D}_{\text{Ind.}}$ a random formula is unsatisfiable with high probability for $m = O(n)$. Furthermore, our analysis in Theorem 14 is essentially tight and it seems unlikely that there is a better bound than $m = O(n^{1-1/k})$. We discuss below why this is the case, by outlining a plausible way for constructing k-clause independent distribution $\mathcal{D} \in \mathcal{F}_k(n, m)$ with $\mathbf{Pr}_{C \sim \mathcal{D}}[\neg SAT(C)] \geq \frac{2}{3}$ and $m = \Theta(n^{1-1/k})$.

Informal outline of the construction

In our construction of the k-clause independent distribution $\mathcal{D}$ we are only concerned with the distribution of clauses over variables, as all literals will be assigned uniformly at random and independently. Then, a sufficient condition for a random formula $C \sim \mathcal{D}$ to be unsatisfiable with large probability is that the 3-uniform hypergraph $G(C)$ constructed in Theorem 14 has a dense subgraph, i.e., a subgraph on $|V(H)|$ vertices (corresponding to variables in C) with at least $|E(H)| \geq 100 \cdot |V(H)|$ hyperedges. Indeed, one can simply count the expected number of satisfying assignments of a random formula ϕ_H on $V(H)$ variables and

$|E(H)|$ fixed clauses with randomly assigned literals: initially, all $2^{|V(H)|}$ assignment are satisfying, but then, as we add $|E(H)|$ clauses one-by-one, the expected number of satisfying assignments reduces each time exactly by a factor 7/8 (each satisfying assignment disappears with probability 1/8). At the end, we get that the expected number of satisfying assignments is $\left(\frac{7}{8}\right)^{100|V(H)|} \cdot 2^{|V(H)|} < 0.01$, which means that ϕ is unsatisfiable most of the times.

Now, we want to make sure that our hypergraph $G(C)$ often has such a "dense" subgraph H, to get an unsatisfiable random formula. Notice that H needs only have a constant number of vertices. Also, note that according to the analysis in Theorem 14, we need to avoid any cycles of length smaller than or equal to k, as the probability of having such a cycle is of order $o(1)$ in $G(C)$ for $m = c \cdot n^{1-1/k}$ and any k-clause independent distribution $C \sim \mathcal{F}_k(n, m)$. Luckily, there are many construction of such 3-uniform hypergraphs H on $|V(H)| = O(1)$ vertices with large number of hyperedges $|E(H)| \geq 100|V(H)|$, and also of large girth $g(H) \geq k + 1$ (e.g., see [23]). We shall "plant" H (essentially insert H as a connected component into G) with large probability in our construction. However, we need to make sure that by inserting H into our graph G, we still have enough room to match $\mathbf{Pr}[c_i = t_i, \forall i \in S] = \prod_{i \in S} \mathbf{Pr}[c_i = t_i]$ for all $S : |S| \leq k$. Next, we give a high level idea how one could achieve this.

As usual for the construction of distributions with identical marginals, we symmetrize $\mathcal{D}$ over all possible permutations of variables in $C \sim \mathcal{D}$. Then, our goal is to match the expected numbers for each isomorphism class of configurations of k hyperedges in $G(C)$ for $C \sim \mathcal{D}$ with $C \sim \mathcal{D}_{\text{Ind.}}$. It is useful to take note of the structure of hypergraph $G(C)$ for $C \sim \mathcal{D}_{\text{Ind.}}$ when $m = c \cdot n^{1-1/k}$: it consists of connected components, each of which is a tree of size at most k; the number of connected components of size k is a constant that grows slightly faster than linearly in c and, more generally, the number of connected components of size $k - j$ is $\Theta(n^{j/k})$. There is also a $o(1)$ probability event of having a Berge-cycle or a connected component of size at least $k + 1$ in $G(C)$ for $C \sim \mathcal{D}_{\text{Ind.}}$. We can ignore in our construction of $\mathcal{D}$ those events, by adding small probability mass to $\mathcal{D}$ that consists of $\mathcal{D}_{\text{Ind.}}$ conditional on any of these rare events (can be easily achieved via rejection sampling from $\mathcal{D}_{\text{Ind.}}$). In this way, we only need to worry about matching expected numbers of forest configurations of size k in $G(C)$ between $C \sim \mathcal{D}$ and $C \sim \mathcal{D}_{\text{Ind.}}$. We can pick the constant in $m = c \cdot n^{1-1/k}$ sufficiently large, so that the constant size subgraph H contributes fewer trees of each type than their respective expected numbers in $G(C)$ for $C \sim \mathcal{D}_{\text{Ind.}}$. Then, we can add a few more connected components that are trees of size k to $C \sim \mathcal{D}$, so that we match k-wise statistics on all tree configurations with $C \sim \mathcal{D}_{\text{Ind.}}$. By having H and a few trees of size k in the graph $G(C)$ for $C \sim \mathcal{D}$, we only utilize a constant number of variables. Then, we would like to keep adding smaller connected components that consist of trees with strictly less than k hyperedges and eventually match k-wise statistics on all forest-like configurations of size k.

It seems plausible that the approach outlined above should work and yield a k-clause independent distribution $\mathcal{D}$. Apart from heavy notation that would be needed to formalize all steps in the above outline, the main technical hurdle is to ensure that statistics for all "forest-like" configurations of k hyperedges with more than one connected component are perfectly matched with $G(C)$ for $C \sim \mathcal{D}_{\text{Ind.}}$. Note, however, that even if our approach fails and there is a stronger version of Theorem 14 with $m = \omega(n^{1-1/k})$ and $\mathbf{Pr}_{C \sim \mathcal{D}}[SAT(C)] \geq \frac{2}{3}$ for any k-clause independent distribution $\mathcal{D}$, such a theorem would require a rather nontrivial argument that relies on subtle dependencies between multiple k-wise statistics.

References

1 Dimitris Achlioptas. Setting 2 variables at a time yields a new lower bound for random 3-SAT. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 28–37, 2000. doi:10.1145/335305.335309.

2 Dimitris Achlioptas. Random satisfiabiliy. In *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 437–462. IOS Press, 2nd edition, 2021. doi:10.3233/FAIA200993.

3 Dimitris Achlioptas and Yuval Peres. The threshold for random k-sat is $2klog2 - o(k)$. *Journal of the American Mathematical Society*, 17:947–973, 2004. URL: http://www.jstor.org/stable/20161221.

4 Noga Alon, Alexandr Andoni, Tali Kaufman, Kevin Matulef, Ronitt Rubinfeld, and Ning Xie. Testing k-wise and almost k-wise independence. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 496–505, 2007. doi:10.1145/1250790.1250863.

5 Noga Alon and Asaf Nussboim. k-wise independent random graphs. In *Proceedings of 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 813–822, 2008. doi:10.1109/FOCS.2008.61.

6 Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The community structure of SAT formulas. In *Proceedings of 15th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 410–423, 2012. doi:10.1007/978-3-642-31612-8_31.

7 Paul Beame, Richard Karp, Toniann Pitassi, and Michael Saks. The efficiency of resolution and davis–putnam procedures. *SIAM Journal on Computing*, 31(4):1048–1075, April 2002. doi:10.1137/S0097539700369156.

8 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001. doi:10.1145/375827.375835.

9 Itai Benjamini, Ori Gurel-Gurevich, and Ron Peled. On k-wise independent distributions and boolean functions. *arXiv:math*, abs/1201.3261, 2012. doi:10.48550/arXiv.1201.3261.

10 Béla Bollobás, Christian Borgs, Jennifer T. Chayes, Jeong Han Kim, and David Bruce Wilson. The scaling window of the 2-SAT transition. *Random Structures and Algorithms*, 18(3):201–256, 2001. doi:10.1002/RSA.1006.

11 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of 55th IEEE Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

12 Andrei Z. Broder, Alan M. Frieze, and Eli Upfal. On the satisfiability and maximum satisfiability of random 3-CNF formulas. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 322–330, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313794.

13 Ioannis Caragiannis, Nick Gravin, Pinyan Lu, and Zihe Wang. Relaxing the independence assumption in sequential posted pricing, prophet inequality, and random bipartite matching. In *Proceedings of 17th International Conference on Web and Internet Economics (WINE)*, pages 131–148, 2021. doi:10.1007/978-3-030-94676-0_8.

14 Peter C. Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 331–340, 1991. URL: http://ijcai.org/Proceedings/91-1/Papers/052.pdf.

15 Vasek Chvátal and Bruce A. Reed. Mick gets some (the odds are on his side). In *Proceedings of 33rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 620–627, 1992. doi:10.1109/SFCS.1992.267789.

16 Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, 1988. doi:10.1145/48014.48016.

17 Amin Coja-Oghlan. The asymptotic k-sat threshold. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 804–813, 2014. doi:10.1145/2591796.2591822.

18 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Transactions on Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

19 Josep Díaz, Lefteris M. Kirousis, Dieter Mitsche, and Xavier Pérez-Giménez. On the satisfiability threshold of formulas with three literals per clause. *Theoretical Computer Science*, 410(30-32):2920–2934, 2009. doi:10.1016/J.TCS.2009.02.020.

20 Jian Ding, Allan Sly, and Nike Sun. Proof of the satisfiability conjecture for large k. *Annals of Mathematics*, 196(1):1–388, 2022. doi:10.4007/annals.2022.196.1.1.

21 Olivier Dubois, Yacine Boufkhad, and Jacques Mandler. Typical random 3-SAT formulae and the satisfiability threshold. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 126–127, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338243.

22 Shaddin Dughmi, Yusuf Hakan Kalayci, and Neel Patel. Limitations of stochastic selection problems with pairwise independent priors. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 479–490, 2024. doi:10.1145/3618260.3649718.

23 David Ellis and Nathan Linial. On regular hypergraphs of high girth. *The Electronic Journal of Combinatorics*, 21(1):1, 2014. doi:10.37236/3851.

24 Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 534–543, 2002. doi:10.1145/509907.509985.

25 Uriel Feige and Eran Ofek. Easily refutable subformulas of large random 3cnf formulas. *Theory of Computing*, 3(1):25–43, 2007. doi:10.4086/TOC.2007.V003A002.

26 Abraham Flaxman. Random planted 3-sat. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1728–1732. Springer New York, New York, NY, 2016. doi:10.1007/978-1-4939-2864-4_330.

27 John Franco and Marvin C. Paull. Probabilistic analysis of the davis putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5(1):77–87, 1983. doi:10.1016/0166-218X(87)90032-1.

28 Joel Friedman, Andreas Goerdt, and Michael Krivelevich. Recognizing more unsatisfiable random k-SAT instances efficiently. *SIAM Journal on Computing*, 35(2):408–430, 2005. doi:10.1137/S009753970444096X.

29 Tobias Friedrich, Anton Krohmer, Ralf Rothenberger, Thomas Sauerwald, and Andrew M. Sutton. Bounds on the satisfiability threshold for power law distributed random SAT. In *Proceedings of 25th Annual European Symposium on Algorithms (ESA)*, pages 37:1–37:15, 2017. doi:10.4230/LIPICS.ESA.2017.37.

30 Tobias Friedrich and Ralf Rothenberger. Sharpness of the satisfiability threshold for non-uniform random k-SAT. In *Proceedings of 21st International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 273–291, 2018. doi:10.1007/978-3-319-94144-8_17.

31 Tobias Friedrich and Ralf Rothenberger. The satisfiability threshold for non-uniform random 2-SAT. In *Proceedings of 46th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 61:1–61:14, 2019. doi:10.4230/LIPICS.ICALP.2019.61.

32 Alan M. Frieze and Stephen Suen. Analysis of two simple heuristics on a random instance of k-SAT. *Journal of Algorithms*, 20(2):312–355, 1996. doi:10.1006/JAGM.1996.0016.

33 Nick Gravin and Zhiqi Wang. On robustness to k-wise independence of optimal bayesian mechanisms. In *Proceedings of the 65th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1275–1293, 2024. doi:10.1109/FOCS61266.2024.00084.

34 Anupam Gupta, Jinqiao Hu, Gregory Kehne, and Roie Levin. Pairwise-independent contention resolution. In *Proceddings of 25th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 196–209, 2024. doi:10.1007/978-3-031-59835-7_15.

35 Mohammad Taghi Hajiaghayi and Gregory B Sorkin. The satisfiability threshold of random 3-SAT is at least 3.52. *arXiv:math*, abs/0310193, 2003. doi:10.48550/arXiv.math/0310193.

36 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.

37 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

38 Anil Kamath, Rajeev Motwani, Krishna V. Palem, and Paul G. Spirakis. Tail bounds for occupancy and the satisfiability threshold conjecture. *Random Structures and Algorithms*, 7(1):59–80, 1995. doi:10.1002/RSA.3240070105.

39 Alexis C. Kaporis, Lefteris M. Kirousis, and Efthimios G. Lalas. The probabilistic analysis of a greedy satisfiability algorithm. *Random Structures and Algorithms*, 28(4):444–480, 2006. doi:10.1002/RSA.20104.

40 Marc Mezard, Giorgio Parisi, and Riccardo Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297(5582):812–815, 2002. doi:10.1126/science.1073287.

41 David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI)*, pages 459–465, 1992. URL: http://www.aaai.org/Library/AAAI/1992/aaai92-071.php.

42 Oleksii Omelchenko and Andrei A. Bulatov. Satisfiability threshold for power law random 2-sat in configuration model. *Theoretical Computer Science*, 888:70–94, 2021. doi:10.1016/J.TCS.2021.07.028.

43 Ronitt Rubinfeld and Ning Xie. Testing non-uniform k-wise independent distributions over product spaces. In *Proceedings of 37th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 565–581, 2010. doi:10.1007/978-3-642-14165-2_48.

44 Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2013.

Optimal Antimatroid Sorting

Benjamin Aram Berendsohn
Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

The classical *comparison-based sorting* problem asks us to find the underlying total ordering of a given set of elements, where we can only access the elements via comparisons. In this paper, we study a *restricted* version, where, as a hint, a set T of possible total orderings is given, usually in some compressed form.

Recently, an algorithm called *topological heapsort* with optimal running time was found for case where T is the set of topological orderings of a given directed acyclic graph, or, equivalently, T is the set of linear extensions of a partial ordering [Haeupler et al. 2024]. We show that a simple generalization of topological heapsort is applicable to a much broader class of restricted sorting problems, where T corresponds to a given *antimatroid*.

As a consequence, we obtain optimal algorithms for the following restricted sorting problems, where the allowed total orders are ...

- ... restricted by a given set of *monotone precedence formulas*;
- ... the *perfect elimination orders* of a given chordal graph; or
- ... the possible *vertex search orders* of a given connected rooted graph.

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

Keywords and phrases sorting, working-set heap, greedy, antimatroid

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.104

1 Introduction

One of the fundamental problems in theoretical computer science is *sorting*. It has a wide variety of applications in practice [24] and is frequently used as an introduction to design and analysis of algorithms.

Perhaps the most intensely studied variant is *comparison-based sorting*, where one can only access the input elements via comparisons (otherwise, we assume the standard word RAM model). This excludes bucket sort, radix sort, and other integer-sorting techniques [10, 16, 15]. The well-known *information-theoretic bound (ITB)* states that comparison-based sorting requires $\log(n!) \approx n \log n$ comparisons in the worst case, and many algorithms asymptotically matching this bound are known.

In this paper, we study a restricted variant of comparison-based sorting. In addition to the n elements to be sorted, we are give a set T of possible total orders, with the guarantee that the actual order of the input is contained in T. Call this the *T-sorting problem*. Note that T may be very large compared to n, so it makes sense to consider variants of the problem where T is given in some compressed form. Regardless of that, the ITB for the T-sorting problem is $\log |T|$, so perhaps the first question to ask is: Can we solve the T-sorting problem with only $\mathcal{O}(\log |T|)$ comparisons?

In turns out that the answer is *yes* if $|T| \geq 2^{\Omega(n)}$. Fredman [9] showed how to solve the T-sorting problem with $\mathcal{O}(n + \log |T|)$ comparisons. This algorithm assumes that T is given explicitly and is thus entirely impractical. Fredman also showed that there exist sets T with $|T| \in \mathcal{O}(n)$ where the T-sorting problem requires $\Omega(n)$ comparisons, which means the ITB is not always tight.

Most later work on this problem has been focused on the special case where T is the set of linear extensions of a partial order P, given by its Hasse diagram. It is known that $\mathcal{O}(\log |T|)$ comparisons are sufficient, that is, the ITB is tight (even when $\log T \leq n$). After

© Benjamin Aram Berendsohn;
licensed under Creative Commons License CC-BY 4.0

decades of research [21, 9, 29, 19, 2, 5, 33], eventually, two algorithms were found that are simultaneously comparison-optimal and running-time optimal [14, 32]. Both algorithms actually solve the slightly more general *DAG sorting* problem: Given is a directed acyclic graph G, and T is taken as the set of topological orderings of G. Note that each DAG G defines a partial order P (though G may contain some "unnecessary" edges), and T is the set of linear extensions of P.

Most important for our purposes is the *topological heapsort* algorithm of Haeupler, Hladík, Iacono, Rozhon, Tarjan, and Tětek [14]. This was the first DAG-sorting algorithm with optimal running time $\mathcal{O}(|V(G)| + |E(G)| + \log|T|)$, though it makes $\Theta(|V(G)| + \log|T|)$ comparisons, which is non-optimal if $|T|$ is small. Achieving comparison optimality requires further modifications, which we discuss later.

We briefly describe topological heapsort. Let G be the given DAG. At the beginning, identify the set S of sources (vertices of in-degree 0) of G, and insert them into a priority queue Q. In each following step, we first find and remove the minimum element x from Q. Then, we remove x from G. This may create several new sources, which we all insert into Q. Repeat until Q is empty. It is easy to see that this algorithm is *correct* if the underlying total order of the vertices is indeed a topological ordering of G. Haeupler et al. achieve optimal running time by using a special priority queue with the so-called *working-set property*.

Let us now try to generalize topological heapsort to solve the general T-sorting problem. Suppose that instead of the graph G, we are given a black-box data structure D that reports all elements that are the minimum of at least one ordering $\tau \in T$. After "removing" the first k elements $x_1, x_2, \ldots, x_k$, let D report all elements y such that $x_1x_2\ldots x_ky$ is a prefix of some ordering $\tau \in T$ (the *available* elements). Again, it is easy to see that topological heapsort with this data structure is *correct*. But is it optimal?

The main contribution of this paper (Section 3) is to show that topological heapsort has optimal running time if T corresponds to a class of structures called *antimatroids*, which are a generalization of partial orders, when we disregard the running time for the data structure D. We additionally show that, for several possible representations of antimatroids, we can implement D such that its total running time is linear in the size of the input, so the running time stays optimal (Section 4). Finally, we show how to additionally achieve comparison optimality (Section 5). The full version of the paper contains various omitted proofs, as well as a discussion of several generalizations of antimatroids where topological heapsort is *not* optimal. In the remainder of this section, we give an informal definition of antimatroids, and discuss some related work.

Antimatroids. Fix a set T of total orderings, and consider the black-box data structure D mentioned above. Recall that, for each sequence $x_1, x_2, \ldots, x_k$ of $k \geq 0$ elements, D tells us the set of *available* elements y such that $x_1x_2\ldots x_ky$ is a prefix of some $\tau \in T$. It is clear that any set T can be described in this way. Intuitively, T is an *antimatroid* if the following conditions apply. First, once an element becomes available, it stays available until it is chosen (availability is *monotone*); and second, availability of an element can only depend on the *set* of elements chosen so far (but not their order).

The first condition can be nicely related to topological heapsort: It means that our priority queue Q at no point contains unavailable elements. The second condition may seem more surprising, but it turns out to be also necessary for optimality of topological heapsort and tightness of the ITB (for details, see the full version of the paper).

Antimatroids were first studied by Dilworth [7] in lattice theory. Later, they were found to be special cases of so-called *greedoids* [25, 26], which are structures admitting a certain greedy optimization procedure. A common alternative definition of antimatroids treats them as *set systems*; this definition can be related to *matroids* (hence the name), which are also special cases of greedoids. We refer to the surveys of Björner and Ziegler [1] and Korte, Schrader, and Lovász [27] for more.

Related work. Optimal algorithms are known [17, 4, 29] for *merging* two sorted lists; this corresponds to a partial order consisting of exactly two disjoint chains. Merging can be generalized to *multiway merging*, where the input is split into any number of sorted lists; here, the ITB is known to be related to the *Shannon entropy* of the list lengths. Multiway merging is important for pratical sorting algortihms that exploit *runs* in a given unsorted sequence (see, e.g., Gelling, Nebel, Smith, and Wild [11]).

A different special case of the T-sorting problem concerns the case where the set T is the permutations that *avoid* a certain pattern π.[1] It is known that $T \leq c_\pi^n$ for some constant c_π depending on π, so the ITB is linear in this case. Opler [31] gave an optimal linear-time algorithm for the pattern-avoiding sorting problem, which works even when the pattern π is unknown. We refer to his paper for more information on that problem.

The $X + Y$*-sorting problem* is a restricted sorting problem where, interestingly, the ITB is *not* tight. Here, two sets $X, Y \subset \mathbb{R}$ of size k each are given, and we need to sort the set $\{x + y \mid x \in X, y \in Y\}$ of size $n = k^2$. If only comparisons of the form $x + y < x' + y'$ with $x, x' \in X$, $y, y' \in Y$ are allowed, then this is a restricted sorting problem by our definition. It is known that the number of possible orderings is only $k^{\Theta(k)}$ [9], which makes the ITB $\Omega(k \log k) = \Omega(\sqrt{n} \log n)$; however, Fredman [9] showed that $\Omega(k^2)$ comparisons are needed. It is currently unknown if an algorithm with optimal running time $\Theta(k^2)$ exists.

Finally, a question closely related to restricted sorting is finding *balanced pairs*, which are comparisons that split the set of possible total orders into two parts of approximately equal size. There is a body of work on balanced pairs in partial orders [21, 9, 29, 2, 3], and the concept has been generalized to antimatroids [8].

2 Preliminaries

All the following definitions are taken from Björner and Ziegler [1] with only slight changes.

Words and languages. Let an *alphabet* Σ be a set of *letters*. A *word* on Σ is a finite sequence of letters. We write words without commas as $\alpha = x_1 x_2 \dots x_n$, where $x_i \in \Sigma$, and we write $\alpha\beta$ for the concatenation of two words α and β. We denote the length of a word α by $|\alpha|$. The *support* $\tilde{\alpha}$ of a word α is the set of letters contained in it, and the empty word is denoted by ε.

A word α is *simple* if each letter occurs at most once in it (i.e., $|\tilde{\alpha}| = |\alpha|$), and α is a *permutation* of Σ if each letter occurs precisely once in it. The set of all simple words on Σ is denoted by Σ_s^*, and a *simple language* is a subset $L \subseteq \Sigma_s^*$. We denote the set of permutations contained in a simple language by $\mathcal{P}(L)$, and we call L *full* if $|\mathcal{P}(L)| > 0$.

[1] A permutation τ of a set X *avoids* a permutation π of $[k]$ if τ contains no subsequence that is order-isomorphic to π, according to some fixed total order on X. This restricted sorting problem is usually formulated in a different, but equivalent way where X is given as a sequence that is known to avoid a pattern.

Partial orders. A partial order on a set Σ is denoted with a capital letter, like P, and we write $\prec_P$ for the actual partial order relation. Total orders are treated as permutations of the underlying set Σ. Whenever necessary, we write $\prec_\pi$ for the total order relation corresponding to the permutation π, i.e., we have $x \prec_\pi y$ if x precedes y in π. An *oracle* for a total order π is a black-box function that answers queries of the form $x \prec_\pi y$ in constant time.

Antimatroids. An *antimatroid* (sometimes called *antimatroid language*) is a simple language $A \subseteq \Sigma_s^*$ that satisfies the following two axioms:

- **(i)** For all $\alpha \in \Sigma_s^*$ and $x \in \Sigma \setminus \tilde{\alpha}$, if $\alpha x \in A$, then $\alpha \in A$. (A is closed under taking prefixes.)
- **(ii)** For each $\alpha, \beta \in A$ with $\tilde{\alpha} \not\subseteq \tilde{\beta}$, there is some $x \in \tilde{\alpha} \setminus \tilde{\beta}$ such that $\beta x \in A$.

Axiom (i) is called *accessibility* and implies that every word in L can be *built* by starting with ε and successively appending letters, without ever leaving L. Axiom (ii) ensures the two "availability properties" of antimatroids mentioned in the introduction.

In this paper, we only consider *full* antimatroids (recall that this means $\mathcal{P}(A) \neq \emptyset$).[2] A full antimatroid A is completely determined by $\mathcal{P}(A)$; in fact, A is exactly the set of prefixes of $\mathcal{P}(A)$. However, not every set of permutations can be described by an antimatroid; for example, any antimatroid on $\Sigma = \{x, y, z\}$ that contains xyz and zyx also contains xzy.

Throughout this paper, we use a different formalism, which is closer to the informal definition of antimatroids given in the introduction. For an alphabet Σ, a *precedence function* for a letter $x \in \Sigma$ is a function $p_x \colon 2^{\Sigma \setminus \{x\}} \to \{0, 1\}$ that is *monotone*, i.e., we have $p_x(X) \leq p_x(Y)$ if $X \subseteq Y \subseteq \Sigma$. A *monotone precedence system (MPS)* on Σ is a collection $S = \{p_x\}_{x \in \Sigma}$ of precedence functions. The *language* of S is the set

$$\mathcal{L}(S) = \{x_1 x_2 \dots x_k \mid \forall i \in [k] : p_{x_i}(\{x_1, x_2, \dots, x_{i-1}\}) = 1\}.$$

The full version of the paper contains a proof that monotone precedence systems indeed characterize (full) antimatroids. In the following, we write $\mathcal{P}(S) = \mathcal{P}(\mathcal{L}(S))$ for convenience.

▶ **Example 1.** Let P be a partial order on a set Σ. Define $S = \{p_x\}_{x \in \Sigma}$ with $p_x(X) = 1$ if and only if $\{y \in \Sigma \mid y \prec_P x\} \subseteq X$. It is easy to see that $\mathcal{P}(S)$ is exactly the set of linear extensions of P. Thus, antimatroids generalize partial orders.

▶ **Example 2.** Let $\Sigma = \{a, b, c\}$, and consider the MPS S with $p_a(X) = p_b(X) = 1$ for all $X \subseteq \Sigma$, and $p_c(X) = 1$ iff $a \in X$ or $b \in X$. Then $\mathcal{P}(S) = \{abc, bac, acb, bca\}$. Since $\mathcal{P}(S)$ contains both acb and its reverse bca, but not all possible orderings, it is *not* the set of linear extensions of any partial order. Thus, antimatroids *strictly* generalize partial orders.

Priority queues. The central data structure of topological heapsort is the *working-set priority queue*. This is a priority queue (like the well-known *binary heap* [34]) with the so-called *working-set property*. Informally, this property ensures that extracting an element is fast if that element has been inserted only shortly before. We omit the precise definitions here, since they are not needed for our proofs, and refer to Haeupler et al. [14] for more details.

[2] Some authors define antimatroids to be always full [27].

3 Generalized topological heapsort

Let us now formally state the antimatroid-sorting meta-problem. Given is a set Σ, an antimatroid A on Σ, and a comparison oracle for a permutation $\pi \in \mathcal{P}(A)$. The task is to output π (or, equivalently, output Σ in sorted order according to the oracle comparisons $\prec_\pi$). Note the assumption $\pi \in \mathcal{P}(A)$ in particular implies that A is full.

We call this a meta-problem since the precise representation of A is unspecified. We discuss some possible representations in Section 4. For now, we give a meta-algorithm for the problem (Algorithm 1), using the characterization of A as a monotone precedence system S.

Algorithm 1 Topological heapsort for antimatroids.

Input: Set Σ, MPS $S = \{p_x\}_{x\in\Sigma}$, comparison oracle for some $\pi \in \mathcal{P}(S)$.
Initialize working-set priority queue Q that contains each $x \in \Sigma$ with $p_x(\emptyset) = 1$
$\alpha \leftarrow \varepsilon$
while Q is not empty **do**
  Delete the minimum from Q and append it to α
  For each $x \in \Sigma \setminus (\tilde{\alpha} \cup Q)$, add it to Q if $p_x(\tilde{\alpha}) = 1$
return α

We can already show correctness of our meta-algorithm regardless of implementation details.

▶ **Lemma 3.** *Topological heapsort is correct for every input* (Σ, S, π).

Proof. Let $S = \{p_x\}_{x\in\Sigma}$ and $\pi = x_1, x_2, \ldots, x_n$. We need to show that topological heapsort produces $\alpha = \pi$.

Since $\pi \in \mathcal{P}(S)$, we have $p_{x_i}(\{x_1, x_2, \ldots, x_{i-1}\}) = 1$ for each $i \in [n]$. In particular, this means that $p_{x_1}(\emptyset) = 1$, so x_1 is contained in Q at the start. Since also $x_1 \prec_\pi x_i$ for all $i > 1$, the first element added to α is indeed x_1. Suppose now that we have $\alpha = x_1 x_2 \ldots x_{i-1}$ at the start of some iteration. Since $p_{x_i}(\tilde{\alpha}) = 1$, we have $x_i \in Q$. Since $x_i \prec_\pi x_j$ for all $j > i$, the algorithm next appends x_i to α. By induction, the algorithm indeed returns π. ◀

Two parts of the algorithm are left unspecified: How to identify the initial elements in Q, and the elements inserted in each iteration. We now define an abstract data structure for this task. A *candidate data structure* C for a monotone precedence system $S = \{p_x\}_{x\in\Sigma}$ maintains a set $X \subseteq \Sigma$, initially $X = \emptyset$, and supports the following two operations:

- C.`init`(): Set $X \leftarrow \emptyset$ and report all $x \in \Sigma$ with $p_x(\emptyset) = 1$.
- C.`step`(x): Given $x \in \Sigma \setminus X$ with $p_x(X) = 1$, report all $y \in \Sigma$ with $p_y(X) = 0$ and $p_y(X \cup \{x\}) = 1$. Then, add x to X.

Observe that C allows at most $|\Sigma|$ calls to `step` after `init`, one for each $x \in \Sigma$. Further, parameters of `step` calls form a sequence $x_1 x_2 \ldots x_k \in \mathcal{L}(S)$. The *total time* for C, written $t^{\text{total}}(C)$, is the overall time for one call to `init` and a valid sequence of `step` calls, in the worst case.

Clearly, any correct candidate data structure can be used to finish the implementation of topological heapsort. The actual implementation will heavily depend on the given representation of the antimatroid; see Section 4.

a	b	c	d	e	
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	
x_1	x_2	x_3	x_4	x_5	
Q_0	Q_1	Q_2	Q_3	Q_4	Q_5
$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$
$\{a, c\}$	$\{b, c\}$	$\{c\}$	$\{d, e\}$	$\{e\}$	$\emptyset$

Figure 1 An example transcript of topological heapsort for some antimatroid on $\Sigma = \{a, b, c, d, e\}$.

3.1 Running time analysis

Since we want our analysis to be independent of the candidate data structure, we split the work done by topological heapsort into two essential parts.

- The total time $t^{\text{total}}(C)$ spend by the candidate data structure, as defined above.
- The time required to initialize Q (given the initial elements), delete the minimum from Q, and insert given elements into Q in each step. Call this the *queue time* t^{Q}.

Since the loop performs at most $n = |\Sigma|$ iterations, the running time of topological heapsort is $t^{\text{total}}(C) + t^{\text{Q}} + \mathcal{O}(n)$.

Note that the queue time does *not* depend on the representation of S at all. In the following, we show that t^{Q} is always $\mathcal{O}(n + \log|\mathcal{P}(S)|)$, by reducing to the special case where S is a *partial order*, which has been solved by Haeupler et al. [14]. The number of comparisons can be as large as the queue time, which is not optimal if $\log(|\mathcal{P}(S)|) \ll n$. In Section 5, we improve this bound to the optimal $\mathcal{O}(\log|\mathcal{P}(S)|)$ by modifying the algorithm.

Partial orders. Let G be a directed acyclic graph with n vertices and m edges, and consider the set $\mathcal{T}(G)$ of topological orderings of G. Clearly, $\mathcal{T}(G)$ is the set of linear extensions of a partial order on $V(G)$. Thus, $\mathcal{T}(G)$ is an also antimatroid in $V(G)$ (see Example 1).

It is easy to implement a candidate data structure for $\mathcal{T}(G)$ via the standard topological sorting algorithm [18, 23], with total running time $\mathcal{O}(m + n)$. Haeupler et al. [14] showed that the queue time of topological heapsort is optimal for partial orders:

▶ **Lemma 4** (Haeupler et al. [14]). *Let S be a monotone precedence system corresponding to a partial order on a set Σ, and let $\pi \in \mathcal{P}(S)$ be a total order on Σ. Then, for topological heapsort with input (Σ, S, π), we have $t^{\text{Q}} \in \mathcal{O}(|\Sigma| + \log|\mathcal{P}(S)|)$.*

General antimatroids. We now generalize Lemma 4 to antimatroids. We need the following definition. The *transcript* of a run of topological heapsort on an input (Σ, S, π) is the sequence $(Q_0, Q_1, \ldots, Q_n)$, where Q_0 is the set of elements initially in the priority queue, and Q_i for $i \in [n]$ is the set of elements in the priority queue after the i-th step. Observe that, if $\pi = x_1 x_2 \ldots x_n$, then $x_i \in Q_{i-1}$, and $x_i \notin Q_j$ for each $j \geq i$. See Figure 1 for an example.

A crucial fact that we use in the proof below is that the queue time t^{Q} *only* depends on the transcript of the run. In particular, two runs on inputs (Σ, S, π) and (Σ, S', π) that have the same transcript also have the same queue time, even if $S \neq S'$.

▶ **Theorem 5.** *Let S be a monotone precedence system on Σ, and let $\pi \in \mathcal{P}(S)$ be a total order on Σ. Then, for topological heapsort with input (Σ, S, π), we have $t^{\text{Q}} \in \mathcal{O}(|\Sigma| + \log|\mathcal{P}(S)|$.*

Proof. Consider a run of topological heapsort on (Σ, S, π) with transcript $(Q_0, Q_1, \ldots, Q_n)$, and let $\pi = x_1 x_2 \ldots x_n$.

We now define a partial order P on Σ. For each $i \in [n]$, let $x_i \prec_P y$ for each $y \in \Sigma \setminus (Q_0 \cup Q_1 \cup \dots Q_{i-1})$. Informally, we have $x \prec_P y$ if x is deleted from the priority queue before y is inserted. Let S_P be the monotone precedence system corresponding to P. Clearly, $\pi \in \mathcal{P}(S_P)$. We claim the following:

(i) The transcript of running topological heapsort on (Σ, S_P, π) is again $(Q_0, Q_1, \dots, Q_n)$.

(ii) $\mathcal{P}(S_P) \subseteq \mathcal{P}(S)$.

We first explain how this implies the theorem. By Lemma 4, running topological heapsort on (Σ, S_P, π) has queue time $\mathcal{O}(|\Sigma| + \log|\mathcal{P}(S_P)|)$. By claim (i), running topological heapsort on (Σ, S, π) has the same queue time (since the transcript is the same), which, by claim (ii), is at most $\mathcal{O}(|\Sigma| + \log|\mathcal{P}(S)|)$, as required.

We now prove claim (i). Let $(Q'_0, Q'_1, \dots, Q'_n)$ be the transcript of running topological heapsort on (Σ, S_P, π). Each $y \in Q_0$ is a minimal element of P by definition, so $Q_0 \subseteq Q'_0$. On the other hand, each $y \notin Q_0$ satisfies $x_1 \prec_P y$ by definition and thus $y \notin Q'_0$, so $Q_0 = Q'_0$.

Now take some $j \in [n]$ and assume that by induction, we have $Q_i = Q'_i$ for all $i < j$. For each $y \in Q_j$, we have $\{z \in \Sigma \mid z \prec_P y\} \subseteq \{x_1, x_2, \dots, x_j\}$, and $y \notin \{x_1, x_2, \dots, x_j\}$, so $y \in Q'_j$ by definition. On the other hand, each $y \notin Q_j$ satisfies either (a) $y \in \{x_1, x_2, \dots, x_j\}$ or (b) $y \notin Q_0 \cup Q_1 \cup \dots \cup Q_j$. (a) directly implies $y \notin Q'_j$ (y is removed earlier). (b) implies $x_{j+1} \prec_P y$, which again implies $y \notin Q'_j$.

We finish the proof with claim (ii). Let $\alpha = x'_1 x'_2 \dots x'_n \in \mathcal{P}(S_P)$. We show that $\alpha \in \mathcal{P}(S)$, i.e., that for each $i \in [n]$, we have $p_{x'_i}(\{x'_1, x'_2, \dots, x'_{i-1}\}) = 1$.

Let $y \in \Sigma$, and let k be minimal such that $y \in Q_k$. Let $X_k = \{x_1, x_2, \dots, x_k\}$. First, by definition of Q_k, we have $p_y(X_k) = 1$. Second, we have $\{x \in \Sigma \mid x \prec_P y\} = X_k$ by minimality of k. Now suppose $y = x'_i$, and let $X'_{i-1} = \{x'_1, x'_2, \dots, x'_{i-1}\}$. Then $p'_y(X'_{i-1}) = 1$, implying $X'_{i-1} \supseteq X_k$. By monotonicity, we have $p_y(X'_{i-1}) \geq p_y(X_k) = 1$, as desired. ◀

Remark. The technique of constructing an auxiliary partial order is inspired by the work of Haeupler, Hladík, Rozhoň, Tarjan, and Tětek [12] (we discuss further connections in Section 4.2). In fact, the exact partial order P in the proof of Theorem 5 appears in the revised version[3] of the paper that introduced topological heapsort [14]. They observe that P is an *interval order*, which means $|\mathcal{P}(P)|$ is relatively easy to bound and can be related to the working-set bound.

The algorithm of van der Hoog, Rotenberg, and Rutschmann [32] is also analyzed with the help of interval orders, but seemingly cannot be easily generalized to antimatroids.

4 Antimatroid representations

In this section, we consider several representations of antimatroids (some only covering special cases), and discuss how to implement candidate data structures (CDSs) for these representations. If C is a CDS for an animatroid A with some representation, then we call C *efficient* if $t^{\text{total}}(C) \in \mathcal{O}(n)$, where n is the size of the representation (in machine words). Theorem 5 implies that topological heapsort with an efficient CDS has optimal running time.

4.1 Precedence formulas

We first consider an explicit representation of general full antimatroids based on monotone precedence systems. The *precedence formula representation* of a full antimatroid A on Σ is a collection $\{F_x\}_{x \in \Sigma}$, where F_x is a monotone boolean formula on the variable set $V_x = \Sigma \setminus \{x\}$. (A *monotone* boolean formula allows only the operators $\vee$ and $\wedge$ and the constants 0 and 1, notably prohibiting the negation $\neg$.)

[3] An earlier preprint [13] used a different proof technique.

Each formula F_x represents a precedence function p_x as follows: For $Y \subseteq \Sigma$, we let $p_x(Y)$ be the value of F_x when setting each $y \in Y$ to 1, and each $y \in X \setminus (Y \cup \{x\})$ to 0. Since any monotone function f can be represented as a monotone boolean formula, this representation is suitable for every antimatroid. In the following, we assume each formula is given as a linked list of symbols with each variable, operator, constant, or parenthesis in the formula occupying one machine word. Other representations, like parsing trees and circuits, are also possible.

An CDS C for $\{F_x\}_{x\in\Sigma}$ is implemented as follows. We need two preprocessing steps. First, for each formula, we simplify it. We repeatedly replace substrings $(1 \wedge x) \to x$, $(1 \vee x) \to 1$, $(0 \wedge x) \to 0$, and so on, in linear overall time. Afterwards, we have $F_x = 1$ for each x with $p_x(\emptyset) = 1$.

Second, compute a directed graph G, where $V(G) = \Sigma$, and $(x, y) \in E(G)$ if F_y contains the variable x. For each such edge, we store a list of pointers to each occurrence of x within F_y. Note that the size of G (including pointers) is at most the total size of all formulas.

The initial candidate set reported by `init()` is the set of sources in G. `step`(x) is implemented as follows: For each out-neighbor y of x, replace each occurrence of x in F_y with 1. Then, simplify F_y as outlined above, and report y if $F_y = 1$ after the simplification. Finally, remove x from the graph. To see that the data structure is correct, observe that after calling `step`(x) on each x in some set $X \subset \Sigma$, we have $F_y = 1$ for each y with $p_y(X) = 1$.

The total running time is linear, since each edge is visited once, and simplification is linear overall. Thus, the data structure is efficient, and so:

▶ **Theorem 6.** *Given an antimatroid A on Σ in a precedence formula representation of length n, we can solve the $\mathcal{P}(A)$-sorting problem in optimal time $\mathcal{O}(n + \log|\mathcal{P}(A)|)$.*

There are two representations of antimatroids that are similar to precedence formulas: *rooted set collections* and *alternative precedence structures* [27]. These roughly correspond to precedence formulas in conjuctive normal form and disjunctive normal form, respectively. A third related characterization with so-called *elementary ranking conditions* [30] is used in linguistics (see the full version of the paper for details). In the following, we discuss two more examples that only apply to subclasses of antimatroids.

4.2 Vertex search and distance orderings

Given a connected graph G and a designated root $r \in V(G)$, the *vertex search antimatroid* on (G, r), written $A^{\mathrm{VS}}_{G,r}$, contains the empty word ε as well as each simple word $v_1 v_2 \dots v_k$ where $v_1 = r$ and each prefix $v_1 v_2 \dots v_i$ induces a connected subgraph of G. The words in $A^{\mathrm{VS}}_{G,r}$ model each way of traversing the graph in a manner similar to BFS or DFS. The basic idea is easily extended to directed graphs; here we require that each prefix $v_1 v_2 \dots v_i$ induces a subgraph of G where every vertex is reachable from $r = v_1$. (We assume that each vertex is reachable from r in G.)

It is easy to see that $A^{\mathrm{VS}}_{G,r}$ is indeed an antimatroid; for example, as a precedence formula for each vertex $v \neq r$, take $F_v = u_1 \vee u_2 \vee \dots \vee u_k$, where $u_1, u_2, \dots, u_k$ are the neighbors (resp. in-neighbors) of v. However, not every antimatroid is a vertex search antimatroid.

Again, the restricted sorting problem for vertex search antimatroids is solved in optimal time with the precedence formula representation given above.

The vertex search antimatroid plays a role in the *distance ordering* problem, a variant of single-source shortest path (SSSP) problem. Given is an edge-weighted directed graph (G, w) with a designated root r, and the task is to order the vertices by distance from r. Recently, Haeupler et al. [12] showed that Dijkstra's classical SSSP is *universally optimal* when equipped with a certain working-set heap (which is more powerful than the one presented in Section 2).

Universally optimal means, informally, that for every fixed graph, the algorihtm is worst-case optimal w.r.t. the weight function. In fact, they essentially match the information-theoretic lower bound with a running time of $\mathcal{O}(\log |T| + |V(G)| + |E(G)|)$, where T is the set of possible distance orderings for the input graph G with root r, with any weight function.

We now demonstrate a strong connection between antimatroid sorting and the distance ordering problem.[4] First off, the set T of possible distance orderings is precisely the vertex search antimatroid (see the full version of the paper for a proof). Thus, we can run topological heapsort to find a distance ordering with optimal running time – *if* we are allowed to directly compare two vertices. That means comparing the distances of two vertices to r, which we cannot do in the distance ordering problem (without computing the distances first).

Still, it can be seen that the *transcript* (see the proof of Theorem 5) of a run of Dijsktra's algorithm on an input (G, w, r) is the same as running topological heapsort on the corresponding vertex search antimatroid. This gives an alternative proof of universal optimality; details are found in the full version of the paper. It should be noted that Haupler et al. also give an algorithm that is universally optimal in both time and number of comparisons, which is harder (c.f. Section 5).

4.3 Elimination orderings

An undirected graph is *chordal* if has no induced cycles of length four or more. It is well-known that chordal graphs can be characterized by the existence of *perfect elimination orderings (PEOs)*. A PEO is obtained by successively removing *simplicial* vertices, that is, vertices whose neighborhood form a clique.

The set of elimination orderings form an antimatroid, also known as the *simplicial vertex pruning* [1] or *simplicial shelling* [27] antimatroid. Given a graph G, we can define the simplicial vertex pruning antimatroid A_G^{SVP} via the MPS $\{p_v\}_{v \in V(G)}$ with

$$p_v(U) = \begin{cases} 1, & \text{if } v \text{ is simplicial in } G - U; \\ 0, & \text{otherwise.} \end{cases}$$

A candidate data structure for A_G is essentially a data structure that maintains the set of simplicial vertices in G under simplicial vertex deletion.

There are known static and fully-dynamic algorithms to compute simplicial vertices in *arbitrary* graphs [22, 28]. Since we only care about chordal graphs and only need vertex deletion, we can use a relatively simple data structure based on *clique trees*. Details are found in the full version of the paper. The total running time of removing all vertices is $\mathcal{O}(m + n)$. Thus, we can solve the corresponding restricted sorting problem in optimal time.

5 Comparison optimality

In this section, we modify the topological heapsort algorithm to only use an optimal $\mathcal{O}(\log |\mathcal{P}(S)|)$ comparisons, instead of $\mathcal{O}(|\Sigma| + \log |\mathcal{P}(S)|)$, while keeping the optimal running time. The special case of partial orders was again solved by Haeupler et al. [14]. Our algorithm is similar, but some non-trivial adaptation is required.

First of all, note that the additional $\mathcal{O}(n)$ term is only relevant when $|\mathcal{P}(S)| \leq 2^{o(n)}$. In the partial order/DAG-sorting case, Haeupler et al. [14] showed that this condition implies the existence of a long directed path in the DAG. Observe that such a path is pre-sorted. We prove a corresponding fact for antimatroids (Section 5.1).

[4] Such a connection is not surprising, since the technique used in the analysis of Dijkstra's algorithm [12] inspired the original analysis of topological heapsort [14] and large parts of this paper.

Haeupler et al. then proceed roughly as follows. They first compute a set of *marked* vertices on the pre-sorted path. Intuitively, the marked vertices represent the interaction with the remainder of the graph: The removal of unmarked vertices does not affect "availability" of vertices outside of the path. This means that often, a large set of unmarked vertices can be removed at once, without any additional comparisons.

This strategy cannot be adapted to general antimatroids in a straight-forward way, since the availability constraints are too complex to preprecompute a small set of marked vertices. We instead use the following three-step algorithm (given is an antimatroid A on Σ):

1. Compute a long pre-sorted sequence β (Section 5.1).
2. Sort the set $\Gamma = \Sigma \setminus \tilde{\beta}$, using topological heapsort on the sub-antimatroid of A *induced* by Γ (Section 5.2). This gives us a sorted sequence γ.
3. Merge the two sequences β and γ, utilizing the antimatroid constraints (Section 5.3).

All three steps can be implemented efficiently using a candidate data structure for A. We obtain:

▶ **Theorem 7.** *Given a set Σ, a candidate data structure C for an antimatroid A on σ, and an oracle for a total order $\pi \in \mathcal{P}(A)$, we can sort Σ w.r.t. $\prec_\pi$ in time $\mathcal{O}(t^{\text{total}}(C) + \log|\mathcal{P}(A)|)$ and with $\mathcal{O}(\log|\mathcal{P}(A)|)$ comparisons.*

This means that each efficient candidate data structure presented in Section 4 yields a comparison-optimal sorting algorithm.

5.1 Bottlenecks in antimatroids

Let $S = \{p_x\}_{x\in\Sigma}$ be a monotone precedence system on Σ. The *layer sequence* of S is a partition of Σ into nonempty *layers* $L_1, L_2, \ldots, L_k$ inductively defined as follows:

- $L_1 = \{x \in \Sigma \mid p_x(\emptyset) = 1\}$.
- $L_i = \{x \in \Sigma \setminus \bar{L}_{i-1} \mid p_x(\bar{L}_{i-1}) = 1\}$, where $\bar{L}_{i-1} = L_1 \cup L_2 \cup \cdots \cup L_{i-1}$, for each $2 \le i \le k$.

Observe that a layer sequence exists if and only if $\mathcal{P}(S) \neq \emptyset$, and that a layer sequence is always unique. If a candidate data structure C is available, we can compute the layer sequence in $\mathcal{O}(t^{\text{total}}(C))$ time as follows: Call $C.\texttt{init}()$ and add the reported elements to L_1. Then, call $C.\texttt{step}(x)$ for each $x \in L_1$, add the reported elements to L_2, and so on.

▶ **Lemma 8.** *Let S be a monotone precedence system with k layers. Then $|\mathcal{P}(S)| \ge 2^{n-k}$.*

Proof. We essentially follow Haeupler et al. [14]. Let $L_1, L_2, \ldots, L_k$ be the layer sequence for S. For each $i \in [k]$, let α_i be an arbitrary permutation of L_i. We claim that the word $\alpha = \alpha_1\alpha_2\ldots\alpha_k$ is contained in $\mathcal{P}(S)$. Indeed, if $x \in L_i$, then all $y \in \bar{L}_{i-1}$ precede x in α. Since $p_x(\bar{L}_{i-1}) = 1$ by definition of L_i, we have $\alpha \in \mathcal{P}(S)$.

There are $\prod_{i=1}^{k} |L_i|!$ ways of constructing α. Writing $n_i = |L_i|$ and $n = |\Sigma|$, we have

$$|\mathcal{P}(S)| \ge \prod_{i=1}^{k} n_i! \ge \prod_{i=1}^{k} 2^{n_i-1} = 2^{n-k}. \qquad \blacktriangleleft$$

We now proceed to show how to find a long sequence $x_1x_2\ldots x_k \in \Sigma_{\text{s}}^*$ such that for all $i \in [k-1]$, we have $x_i \prec_\alpha x_{i+1}$ for all $\alpha \in \mathcal{P}(A)$; in other words, the sequence is pre-sorted. In the DAG-sorting case, we can simply take a longest path in the DAG. Here, we need a different approach, which is also related to the technique of Haeupler et al. [14].

▶ **Lemma 9.** *Let $L_1, L_2, \ldots, L_k$ be the layers of a monotone precedence system S, and let $\pi \in \mathcal{P}(S)$. Then, for all $1 \le i < j \le k$ and $y \in L_j$, there exists an $x \in L_i$ such that $x \prec_\pi y$.*

Proof. Suppose the lemma does not hold for some pair $i < j$, i.e., there exists some $y' \in L_j$ such that $y' \prec_\pi x$ for all $x \in L_i$. Let $y = \min_\pi(\Sigma \setminus \bar{L}_i) = \min_\pi(L_{i+1} \cup L_{i+2} \cup \cdots \cup L_k)$. Since $y \preceq_\pi y'$, we have $y \prec_\pi x$ for all $x \in L_i$, implying that actually $y = \min_\pi(\Sigma \setminus \bar{L}_{i-1})$. This means that $p_y(\bar{L}_{i-1}) = 1$, implying that $y \in L_i$, a contradiction. ◀

An element $x \in \Sigma$ is a called a *bottleneck* if it is the only element in its layer. Let t be the number of bottlenecks and let k be the number of layers. Since each non-bottleneck layer contains at least two vertices, we have $n - t \geq 2(k - t)$, which implies $n - k \geq \frac{n-t}{2}$. By Lemma 8, we have

▶ **Corollary 10.** *Let S be an MPS with t bottleneck elements. Then $|\mathcal{P}(S)| \geq 2^{(n-t)/2}$.*

The *bottleneck sequence* $\beta = b_1 b_2 \dots b_t$ of S contains all bottlenecks, ordered by the layer they appear in. Lemma 9 implies that, for every permutation $\pi \in \mathcal{P}(S)$, we have $b_1 \prec_\pi b_2 \prec_\pi \cdots \prec_\pi b_t$. Overall, we obtain

▶ **Lemma 11.** *Let S be an MPS on Σ, and write $n = |\Sigma|$. Then there exists a sequence $\beta = b_1 b_2 \dots b_t$ that appears as a subsequence in every $\pi \in \mathcal{P}(S)$, and $|\mathcal{P}(S)| \geq 2^{(n-t)/2}$. Moreover, given a candidate data structure C for S, we can compute β in time $\mathcal{O}(t^{\text{total}}(C))$.*

5.2 Sorting subsets

In this section, we show how to sort the non-bottleneck elements $\Sigma \setminus \tilde{\beta}$ of an antimatroid A. Recall that a candidate data structure C is given, and we want to sort in overall time $\mathcal{O}(t^{\text{total}}(C) + \log|\mathcal{P}(A)|)$, with $\mathcal{O}(\log|\mathcal{P}(A)|)$ comparisons (see Theorem 7). We show how to sort any subset $\Gamma \subseteq \Sigma$ within this budget.

We first need some definitions. Let Σ be an alphabet. Given a word α and a subset $\Gamma \subseteq \Sigma$, the *restriction* of α to Γ, written $\alpha|_\Gamma$, is obtained by removing all letters in $\Sigma \setminus \Gamma$ from α. For a simple language $L \subseteq \Sigma_s^*$, let $L|_\Gamma = \{\alpha|_\Gamma \mid \alpha \in L\}$. It is known that if $A \subseteq \Sigma_s^*$ is an antimatroid, then the restriction $A|_\Gamma$ is also an antimatroid (called the *trace* [27, 1]).

Let us now assume we have a candidate data structure C_Γ for $A|_\Gamma$. Then we can use topological heapsort to sort Γ. The running time is $t^{\text{total}}(C_\Gamma) + \mathcal{O}(\log|\mathcal{P}(A|_\Gamma)|)$, and we need $\mathcal{O}(|\Gamma| + \log|\mathcal{P}(A|_\Gamma)|)$ comparisons. First, observe that $|\mathcal{P}(A|_\Gamma)| \leq |\mathcal{P}(A)|$, since the restriction operation surjectively maps $\mathcal{P}(A|_\Gamma)$ to $\mathcal{P}(A)$. Second, recall that $|\Gamma| \leq 2\log|\mathcal{P}(A)|$ if Γ is obtained by removing all bottleneck elements (Corollary 10). If we further assume that $t^{\text{total}}(C_\Gamma) \leq \mathcal{O}(t^{\text{total}}(C))$, then we are within the budget of Theorem 7.

It remains to build a candidate data structure C_Γ for $A|_\Gamma$ with $t^{\text{total}}(C_\Gamma) \leq \mathcal{O}(t^{\text{total}}(C))$. Suppose C is given. Then C_Γ.`init`() is implemented as follows. First, call C.`init`(), and store the result in a set Y. Then, we repeat the following as long as there is an element $y \in Y \setminus \Gamma$: Discard y, call C.`step`(y), and add all reported elements to Y. At the end, report the final set Y.

To implement C_Γ.`step`(y), we similarly first call C.`step`(y), store the result in a set Y, and perform the repeated replacement as above.

The (rather technical) correctness proof for C_Γ is found in the full version of the paper. The running time is clearly $t^{\text{total}}(C) + \mathcal{O}(|\Sigma|) = \mathcal{O}(t^{\text{total}}(C))$. Thus, we have:

▶ **Lemma 12.** *Given a set Σ, a subset $\Gamma \subseteq \Sigma$, a candidate data structure C for an antimatroid A on Σ, and an oracle for a total order $\pi \in \mathcal{P}(A)$, we can sort Γ w.r.t. $\prec_\pi$ in time $\mathcal{O}(t^{\text{total}}(C) + \log|\mathcal{P}(A)|)$ and with $\mathcal{O}(|\Gamma| + \log|\mathcal{P}(A)|)$ comparisons.*

5.3 Merging

We now want to optimally merge under antimatroid constraints:

▶ **Theorem 13.** *Let A be an antimatroid on Σ, let $\pi \in \mathcal{P}(A)$, and let Γ, Δ be a partition of Σ. Let $\gamma = \pi|_\Gamma$ and $\delta = \pi|_\Delta$. Let $R \subseteq \mathcal{P}(A)$ contain each permutation $\rho \in \mathcal{P}(A)$ such that $\rho|_\Gamma = \gamma$ and $\rho|_\Delta = \delta$.*

Given Σ, γ, δ, an oracle for π, and a candidate data structure C for A, we can compute π in time $\mathcal{O}(t^{\text{total}}(C) + \log|R|)$ and with $\mathcal{O}(|\Delta| + \log|R|)$ comparisons.

A proof of Theorem 13 is given in the full version of the paper. The algorithm used can be seen as a simplified version of Haeupler et al.'s *heapsort with lookahead* algorithm [14]; in fact, if the antimatroid A is given as a collection of precedence formulas, it simplifies to a partial order under the assumption that γ and δ are sorted. (However, our proof works with general candidate data structures.)

5.4 Putting things together

We now prove Theorem 7. First, we compute the bottleneck sequence β with Lemma 11. Then, we use Lemma 12 to sort the subset $\Gamma = \Sigma \setminus \tilde{\beta}$. In other words, if π is the underlying total order, we compute $\gamma = \pi|_\Gamma$. Finally, we merge β and γ using Theorem 13. The overall running time is $\mathcal{O}(t^{\text{total}}(C) + \log|\mathcal{P}(A)| + \log|R|)$, and we use $\mathcal{O}(\log|\mathcal{P}(A)| + |\Gamma| + \log|R|)$ comparisons, where R is as defined in Theorem 13. Note that $R \subseteq \mathcal{P}(A)$. Moreover, we have $|\Gamma| = n - |\beta| \leq 2\log|\mathcal{P}(A)|$ by Lemma 11. Thus, we need $\mathcal{O}(t^{\text{total}}(C) + \log|\mathcal{P}(A)|)$ time and $\mathcal{O}(\log|\mathcal{P}(A)|)$ comparisons, as desired.

6 Conclusion

In this paper, we have shown that the basic variant of *topological heapsort* nicely generalizes to the problem of sorting antimatroids, and its running time stays optimal. We also showed how the algorithm can be modified to be comparison-optimal in the antimatroid case, which in particular implies that the information-theoretic bound is tight for antimatroids. Since topological heapsort is not optimal for further generalizations like *greedoids* (see the full version of the paper), perhaps antimatroids are the most general structure that can be sorted "greedily". The question whether the information-theoretic bound for greedoids holds, and whether greedoids can be sorted efficiently in some other way, is left open.

For another interesting open question, let us consider an important class of antimatroids that was omitted from Section 4. *Convex shelling antimatroids* [6, 27, 1] are obtained by progressively removing points from the convex hull of a given point set. A candidate data structure for such an antimatroid would be a certain kind of *decremental convex hull* data structure. The total running time of this data structure cannot be linear as in the other examples, since computing the convex hull can take superlinear time [20]. Is there a candidate data structure for the convex shelling antimatroid that is fast enough to sort optimally?

References

1 Anders Björner and Günter M. Ziegler. Introduction to Greedoids. In Neil White, editor, *Matroid Applications*, volume 40 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 1992. doi:10.1017/CBO9780511662041.009.

2 G. R. Brightwell, S. Felsner, and W. T. Trotter. Balancing pairs and the cross product conjecture. *Order*, 12(4):327–349, 1995. doi:10.1007/bf01110378.

3 Graham R. Brightwell. Balanced pairs in partial orders. *Discret. Math.*, 201(1-3):25–52, 1999. doi:10.1016/S0012-365X(98)00311-2.

4 Mark R. Brown and Robert Endre Tarjan. A fast merging algorithm. *J. ACM*, 26(2):211–226, 1979. doi:10.1145/322123.322127.

5 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M. Jungers, and J. Ian Munro. Sorting under partial information (without the ellipsoid algorithm). In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 359–368. ACM, 2010. doi:10.1145/1806689.1806740.

6 Brenda L. Dietrich. A circuit set characterization of antimatroids. *Journal of Combinatorial Theory, Series B*, 43(3):314–321, 1987. doi:10.1016/0095-8956(87)90007-4.

7 R. P. Dilworth. Lattices with unique irreducible decompositions. *Annals of Mathematics*, 41(4):771–777, 1940. doi:10.1007/978-1-4899-3558-8_10.

8 David Eppstein. Antimatroids and balanced pairs. *Order*, 31(1):81–99, 2014. doi:10.1007/S11083-013-9289-1.

9 Michael L. Fredman. How good is the information theory bound in sorting? *Theoretical Computer Science*, 1(4):355–361, 1976. doi:10.1016/0304-3975(76)90078-5.

10 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, December 1993. doi:10.1016/0022-0000(93)90040-4.

11 William Cawley Gelling, Markus E. Nebel, Benjamin Smith, and Sebastian Wild. Multiway powersort. In Gonzalo Navarro and Julian Shun, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2023, Florence, Italy, January 22-23, 2023*, pages 190–200. SIAM, 2023. doi:10.1137/1.9781611977561.CH16.

12 Bernhard Haeupler, Richard Hladík, Václav Rozhon, Robert E. Tarjan, and Jakub Tetek. Universal optimality of dijkstra via beyond-worst-case heaps. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2099–2130. IEEE, 2024. doi:10.1109/FOCS61266.2024.00125.

13 Bernhard Haeupler, Richard Hladík, John Iacono, Vaclav Rozhon, Robert Tarjan, and Jakub Tětek. Fast and simple sorting using partial information, 2024. arXiv:2404.04552v1.

14 Bernhard Haeupler, Richard Hladík, John Iacono, Václav Rozhoň, Robert E. Tarjan, and Jakub Tětek. Fast and simple sorting using partial information. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3953–3973. SIAM, 2025. doi:10.1137/1.9781611978322.134.

15 Yijie Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, January 2004. doi:10.1016/j.jalgor.2003.09.001.

16 Yijie Han and Mikkel Thorup. Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In *Proceedings of the 43 rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02)*, pages 135–144. IEEE Comput. Soc, 2002. doi:10.1109/SFCS.2002.1181890.

17 Frank K. Hwang and Shen Lin. A simple algorithm for merging two disjoint linearly-ordered sets. *SIAM J. Comput.*, 1(1):31–39, 1972. doi:10.1137/0201004.

18 Arthur B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962. doi:10.1145/368996.369025.

19 Jeff Kahn and Jeong Han Kim. Entropy and sorting. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing - STOC '92*, STOC '92, pages 178–187. ACM, 1992. doi:10.1145/129712.129731.

20 David G. Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15(1):287–299, 1986. doi:10.1137/0215021.

21 S. S. Kislitsyn. A finite partially ordered set and its corresponding set of permutations. *Mathematical Notes of the Academy of Sciences of the USSR*, 4(5):798â€“801, November 1968. doi:10.1007/bf01111312.

22 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000. doi:10.1016/S0020-0190(00)00047-8.

23 Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 1968.

24 Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1973.

25 B. Korte and L. Lovász. Mathematical structures underlying greedy algorithms. In Ferenc Gécseg, editor, *Fundamentals of Computation Theory*, volume 117 of *Lecture Notes in Computer Science*, pages 205–209, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg. doi:10.1007/3-540-10854-8_22.

26 Bernhard Korte and László Lovász. Greedoids - a structural framework for the greedy algorithm. In William R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 221–243. Elsevier, 1984. doi:10.1016/b978-0-12-566780-7.50019-2.

27 Bernhard Korte, Rainer Schrader, and László Lovász. *Greedoids*. Springer Berlin Heidelberg, 1991. doi:10.1007/978-3-642-58191-5.

28 Min Chih Lin, Francisco J. Soulignac, and Jayme Luiz Szwarcfiter. Arboricity, h-index, and dynamic algorithms. *Theor. Comput. Sci.*, 426:75–90, 2012. doi:10.1016/J.TCS.2011.12.006.

29 Nathan Linial. The information-theoretic bound is good for merging. *SIAM J. Comput.*, 13(4):795–801, 1984. doi:10.1137/0213049.

30 Nazarré Merchant and Jason Riggle. OT grammars, beyond partial orders: ERC sets and antimatroids. *Natural Language & Linguistic Theory*, 34(1):241–269, August 2015. doi:10.1007/s11049-015-9297-5.

31 Michal Opler. An optimal algorithm for sorting pattern-avoiding sequences. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 689–699. IEEE, 2024. doi:10.1109/FOCS61266.2024.00049.

32 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. Simpler optimal sorting from a directed acyclic graph. In Ioana Oriana Bercea and Rasmus Pagh, editors, *2025 Symposium on Simplicity in Algorithms (SOSA)*, pages 350–355. SIAM, January 2025. doi:10.1137/1.9781611978315.26.

33 Ivor van der Hoog and Daniel Rutschmann. Tight bounds for sorting under partial information. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2243–2252. IEEE, 2024. doi:10.1109/FOCS61266.2024.00131.

34 J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964.

Parameterized Algorithms for Computing Pareto Sets

Joshua Marc Könen
Institute of Computer Science, University of Bonn, Germany

Heiko Röglin
Institute of Computer Science, University of Bonn, Germany

Tarek Stuck
Institute of Computer Science, University of Bonn, Germany

Abstract

The problem of computing the set of Pareto-optimal solutions has been studied for a variety of multiobjective optimization problems. For many such problems, algorithms are known that compute the Pareto set in (weak) output-polynomial time. These algorithms are often based on dynamic programming and by weak output-polynomial time, we mean that the running time depends polynomially on the size of the Pareto set but also on the sizes of the Pareto sets of the subproblems that occur in the dynamic program. For some problems, like the multiobjective minimum spanning tree problem, such algorithms are not known to exist and for other problems, like multiobjective versions of many NP-hard problems, such algorithms cannot exist, unless $\mathcal{P} = \mathcal{NP}$.

Dynamic programming over tree decompositions is a common technique in parameterized algorithms. In this paper, we study whether this technique can also be applied to compute Pareto sets of multiobjective optimization problems. We first derive an algorithm to compute the Pareto set for the multicriteria s-t cut problem and show how this result can be applied to a polygon aggregation problem arising in cartography that has recently been introduced by Rottmann et al. (GIScience 2021). We also show how to apply these techniques to also compute the Pareto set of the multiobjective minimum spanning tree problem and for the multiobjective TSP. The running time of our algorithms is $\mathcal{O}(f(w) \cdot \text{poly}(n, p_{\max}))$, where f is some function in the treewidth w, n is the input size, and $p_{\max}$ is an upper bound on the size of the Pareto sets of the subproblems that occur in the dynamic program. Finally, we present an experimental evaluation of computing Pareto sets on real-world instances of polygon aggregation problems. For this matter we devised a task-specific data structure that allows for efficient storage and modification of large sets of Pareto-optimal solutions. Throughout the implementation process, we incorporated several improved strategies and heuristics that significantly reduced both runtime and memory usage, enabling us to solve instances with treewidth of up to 22 within reasonable amount of time. Moreover, we conducted a preprocessing study to compare different tree decompositions in terms of their estimated overall runtime.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases parameterized algorithms, treewidth, multicriteria optimization problems, multicriteria MST, multicriteria TSP, polygon aggregation

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.105

Related Version *Full Version*: https://arxiv.org/abs/2509.06124

Supplementary Material *Software*: https://github.com/Tarek-pub/Bicriteria_Aggregation [19]
archived at swh:1:dir:b6eacf189792239e9115cf288be27c6753b8df17
InteractiveResource: https://github.com/Tarek-pub/Bicriteria_Aggregation_plotting [20]

Funding This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 390685813; 459420781 and by the Lamarr Institute for Machine Learning and Artificial Intelligence lamarr-institute.org.



© Joshua Marc Könen, Heiko Röglin, and Tarek Stuck;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Multiobjective optimization problems arise naturally in many contexts. When booking a train ticket, one might be interested in minimizing the travel time, the price, and the number of changes. Similarly when planning a new infrastructure, companies often have to find a compromise between reliability and costs, to name just two illustrative examples. For multiobjective optimization problems there is usually not a single solution that is optimal for all criteria simultaneously, and hence, one has to find a trade-off between the different criteria. Since it is not a priori clear which trade-off is desired, one often studies the set of *Pareto-optimal solutions* (also known as *Pareto set*), where a solution is Pareto-optimal if it is not dominated by any other solution, i.e., there does not exist another solution that is better in at least one criterion and at least as good in all other criteria. The reasoning behind this is that a dominated solution cannot be a reasonable compromise of the different criteria and it makes sense to restrict one's attention to only the Pareto-optimal solutions.

A lot of research in multiobjective optimization deals with algorithms for computing the Pareto set and with studying the size of this set for various optimization problems. If this set is not too large then it can be presented to a (human) decision maker who can select a solution. The Pareto set also has the important property that any monotone function that combines the different objectives into a single objective is optimized by a Pareto-optimal solution. This means, in order to optimize such a function, one could first generate the Pareto set and then pick the best solution from this set.

This approach is only reasonable if there are not too many Pareto-optimal solutions. While in the worst case the Pareto set can be of exponential size for almost all multiobjective optimization problems, it has been observed that the Pareto set is often small in practice (see, e.g., [16, 23]). It has also been shown in the probabilistic model of smoothed analysis that for many problems the expected size of the Pareto set is polynomial if the input is subject to a small amount of random noise [10, 5]. This motivates the development of algorithms for computing the Pareto set that have polynomial running time with respect to the output size.

In the literature such output-sensitive algorithms have been developed for different multiobjective optimization problems (e.g., for the multiobjective shortest path problem [12, 17, 29], multiobjective flow problems [15, 24], and the knapsack problem when viewed as a bicriteria optimization problem [25]). There is, however, a small caveat. Since these algorithms are usually based on dynamic programming, they are not output-polynomial in the strict sense but they solve certain subproblems of the given instance and their running times depend not only polynomially on the size of the Pareto set of the entire instance but also polynomially on the sizes of the Pareto sets of the subproblems. While in theory it could be the case that some of the subproblems have Pareto sets of exponential size while the Pareto set of the entire instance is small (see, e.g., [9]), this behavior is neither observed in experiments nor does it occur in probabilistic input models. Hence, algorithms that are output-polynomial in the weak sense that their running time depends polynomially on the sizes of the Pareto sets of all subproblems that occur in the dynamic program are useful and state-of-the-art for many multiobjective optimization problems. For some problems such output-sensitive algorithms are not known to exist. In particular, for the multiobjective spanning tree problem no algorithm is known that computes the Pareto set in output-polynomial time (not even in the weak sense).

Dynamic programming is a common technique in parameterized algorithms. For many graph problems, dynamic programming on tree decompositions is applied to obtain fixed-parameter tractable (FPT) algorithms with respect to the treewidth of the input graph.

In this paper, we explore for the first time the potential of dynamic programming on tree decompositions for computing Pareto sets. First we design an algorithm for the multicriteria s-t cut problem and apply it on a problem from cartography that has recently been introduced by Rottmann et al. [28] and that was the original motivation for our research.

Rottmann et al. study maps with building footprints and present a new model how less detailed maps can be derived from a given map. Their method is based on viewing the problem as a bicriteria optimization problem. It is assumed that the plane containing the building footprints is triangulated and a set of triangles to glue together some of the buildings is to be selected that on the one hand minimizes the total area and on the other hand minimizes the total perimeter. Rottmann et al. present an algorithm that computes the set of extreme nondominated solutions (i.e., the set of solutions that optimize a linear combination of the objectives), which is a subset of the Pareto set. They ask if it is also possible to compute the entire Pareto set in some output-efficient way.

We show how a treewidth-based algorithm can be used to compute the Pareto set for the s-t cut problem. The running time of this algorithm is FPT in the treewidth and output-polynomial in the weak sense, i.e., it is of the form $\mathcal{O}(f(w) \cdot \text{poly}(n, p_{\max}))$ where f is some function in the treewidth w, n denotes the input size, and $p_{\max}$ denotes an upper bound on the size of the Pareto sets of the subproblems that occur in the dynamic program.

As a special case, the results for the multiobjective s-t cut algorithm directly translate to the cartography problem by computing the whole Pareto set in FPT time and being output-polynomial in the weak sense. We experimentally analyze the computation of Pareto sets for this cartography problem on real-world datasets by designing a specialized data structure, developing heuristics, and integrating additional implementation-specific techniques tailored to the task.

1.1 Our Results

We first consider the multiobjective s-t cut problem and its special case, the triangle aggregation problem [28]: Given some polygons P and triangles T, the goal is to find all Pareto-optimal subsets $T' \subseteq T$ minimizing both the total area and perimeter of $P \cup T'$. This problem arises in cartography, where the goal is to compute a less detailed map from a given map. Here the polygons are building footprints and the space between these footprints is triangulated. By including triangles, one can glue together these building footprints, leading to a less detailed version of the map. We show that the multiobjective s-t cut problem implies an algorithm to compute the Pareto set in time $\mathcal{O}(nw \cdot 2^w \cdot p_{\max}^2 \log(p_{\max}))$ for this problem, where w is the treewidth of the triangle adjacency graph.

In the full version of the paper, we show that this method for computing Pareto sets can also be applied for other multiobjective optimization problems, such as the multiobjective minimum spanning tree problem (MST) and the multiobjective traveling salesman problem (TSP). For both, we present algorithms with running time $\mathcal{O}(nw^{\mathcal{O}(w)} \cdot p_{\max}^2 \log^{d-2}(w^{\mathcal{O}(w)} \cdot p_{\max}^2))$ where w is the treewidth of the input graph, n is the number of vertices in the graph, $p_{\max}$ is the size of the largest Pareto set computed in one of the subproblems of the dynamic program, and $d \geq 2$ is the (constant) number of different objectives. This shows that the Pareto set can be efficiently computed for graphs with small treewidth if the Pareto sets of the subproblems are not too large, which is often the case in realistic inputs.

Finally, we experimentally evaluate our algorithm on real-world datasets. We introduce new heuristic pruning techniques, runtime estimates for tree decompositions and their root, memory optimization, multi-threading, and other improvements for reducing runtime and memory usage.

1.2 Related Work

Output-polynomial time algorithms (at least in the weak sense) are known for many problems. Examples include the multiobjective shortest path problem [12, 17, 29], multiobjective flow problems [15, 24], and the knapsack problem when viewed as a bicriteria optimization problem [25]. For the multiobjective spanning tree problem such algorithms are not known and there are only results on the set of extreme nondominated solutions. This is a subset of the Pareto set and it contains all solutions that optimize some linear combination of the different objectives. For the multiobjective spanning tree problem it is known that there are only polynomially many extreme nondominated solutions [14], and efficient algorithms for enumerating them exist [2].

Rottmann et al. [28] introduce the triangle aggregation problem under the objective of map simplification by grouping multiple building footprints together. They show that, similar to the multiobjective minimum spanning tree problem, there exist only a polynomial number of nondominated extreme solutions and they present an algorithm to compute this set in polynomial time. They also provide an extensive experimental study of their algorithms on real-world data sets showing that the model captures nicely the intention to aggregate building footprints to obtain less detailed maps. They put it as an open question whether or not it is possible to also compute the set of Pareto-optimal solutions in an output-efficient way.

Motivated by the observation that for many problems the Pareto set is often small in applications, the number of Pareto-optimal solutions has been studied in the probabilistic framework of smoothed analysis in a sequence of articles [27, 22, 10, 5]. It has been shown for a large class of linear integer optimization problems (which contains in particular the multiobjective MST problem and the multiobjective TSP) that the expected number of Pareto-optimal solutions is polynomially bounded if the coefficients (in our case, the edge weights) are independently perturbed by random noise. Formally, the coefficients of $d-1$ out of the d objectives are assumed to be independent random variables with adversarially chosen distributions with bounded density. It is shown that not only the expected number of Pareto-optimal solutions is bounded polynomially in this setting but also all constant moments, so in particular the expected squared number of Pareto-optimal solutions is also polynomially bounded. Combined with this, our results imply that the Pareto set for the multiobjective MST problem, the multiobjective TSP and the multiobjective s-t cut problem can be computed in expected FPT running time $\mathcal{O}(f(w) \cdot \text{poly}(n))$ in the model of smoothed analysis.

While dynamic programming over tree decompositions is a well-established technique, using it effectively in practice comes with more difficulties than simply minimizing the width of the decomposition. In experiments the runtime of an algorithm can vary largely for different tree decompositions with the same width.

Bodlaender and Fomin [8] introduced the concept of the *f-cost* of a tree decomposition, which sums up a cost function f applied to the bag sizes across all nodes. This framework formalizes the observation that not all nodes contribute equally to runtime or memory, and that practical efficiency can benefit from structurally favorable decompositions.

The f-cost approach was later extended and evaluated experimentally by Abseher et al. [1] using machine learning to select a tree decomposition based on different features such as bag sizes, branching factors, and node depths. With their method they could often select a decomposition that performs well in practice. Kangas et al. [18] used the same model for the problem of counting linear extensions, confirming the value of such estimators, and suggesting the average join-node depth as a good single feature for estimating runtime.

Beyond selection heuristics, tree decompositions have also been studied with respect to memory efficiency. Charwat et al. [11] explored compressed representations of intermediate states using decision diagrams, and Betzler et al. [6] proposed anchor-based compression techniques to reduce hard drive memory usage.

In our case, we not only solve an optimization problem, but compute the entire set of Pareto-optimal solutions. This makes memory usage more sensitive to the structure of join operations and limits the applicability of pointer-based or diagram-compressed representations. To address this, we develop a custom estimation strategy for predicting both runtime and memory usage, based on predicting the number of Pareto-optimal solutions at each node and the runtime impact of join operations. We then select a decomposition based on a combination of both runtime and memory usage. Details of this procedure are provided in the full version of the paper.

2 Preliminaries

Let $[i] := \{1, \dots, i\}$. We consider an arbitrary optimization problem with a constant number $d \in \mathbb{N}$ of objectives to be minimized. Each (feasible) solution s is mapped to a cost vector $f(s) = (f(s)_1, \dots, f(s)_d) \in \mathbb{R}^d$ where f is our objective function. A solution s_1 is said to *dominate* another solution s_2 if $f(s_1)_i \leq f(s_2)_i$ for all $i \in [d]$ and there exists some $j \in [d]$ such that $f(s_1)_j < f(s_2)_j$. We write $s_1 \prec s_2$ to denote that s_1 dominates s_2. The set of all non-dominated solutions is called the *Pareto set*. These definitions extend naturally to optimization problems where a subset of objectives is to be maximized instead of minimized. In the following we assume that all objectives are to be minimized, but all arguments and results can be adapted analogously when some (or all) objectives are to be maximized.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two sets of Pareto-optimal solutions. We define their combined Pareto set as $\mathcal{P}_1 + \mathcal{P}_2 := \{p \in \mathcal{P}_1 \cup \mathcal{P}_2 \mid p \text{ is Pareto-optimal in } \mathcal{P}_1 \cup \mathcal{P}_2\}$, their union as $S^{\mathcal{P}_1,\mathcal{P}_2} := \{p_1 \cup p_2 \mid p_1 \in \mathcal{P}_1, p_2 \in \mathcal{P}_2\}$, which contains all pairwise unions of solutions from $\mathcal{P}_1$ and $\mathcal{P}_2$, and the Pareto set of $S^{\mathcal{P}_1,\mathcal{P}_2}$ as $\mathcal{P}_1 \oplus \mathcal{P}_2 := \{s \in S^{\mathcal{P}_1,\mathcal{P}_2} \mid s \text{ is Pareto-optimal in } S^{\mathcal{P}_1,\mathcal{P}_2}\}$.

2.1 Treewidth

Our algorithms rely on dynamic programming over a *tree decomposition*. The concept of tree decomposition and *treewidth*, introduced by Robertson and Seymour [26], provide a measure of how similar a graph is to a tree. Many $\mathcal{NP}$-hard problems become tractable on graphs with small treewidth. Computing the optimal treewidth is $\mathcal{NP}$-hard [3], but it is FPT with respect to the treewidth [7], and a tree decomposition of width at most $2 \cdot w(G)$ can be computed in linear time using approximation algorithms [21]. Hence, we assume that our algorithm starts from such an approximate decomposition if no optimal one is available. In the following, we briefly introduce the notation used for tree decompositions.

▶ **Definition 1** (Tree decomposition). *A tree decomposition of a graph $G = (V, E)$ is a pair $\mathcal{T} = (\mathbb{T} = (\{t_1, \dots, t_m\}, E), \{X_t\}_{t \in V(\mathbb{T})})$ where $\mathbb{T}$ is a tree and each node $t \in V(\mathbb{T})$ is associated with a bag $X_t \subseteq V$, such that the following conditions hold:*

- $\bigcup_{t \in V(\mathbb{T})} X_t = V(G)$*, i.e., every vertex of G appears in at least one bag,*
- $\forall \{u, v\} \in E(G)\ \exists t \in V(\mathbb{T}) : u \in X_t \wedge v \in X_t$*, i.e., for every edge $\{u, v\} \in E(G)$ the vertices u and v must appear together in at least one bag,*
- $T_u = \{t \in V(\mathbb{T}) \mid u \in X_t\}$ *is a connected subtree of $\mathbb{T}$ for every $u \in V(G)$.*

The *width* of a tree decomposition is $w(\mathcal{T}) := \max_{t \in V(\mathbb{T})} |X_t| - 1$, and the *treewidth* $w(G)$ is the smallest possible *width* of a tree decomposition. We denote with V_t the union of vertices that were introduced at t or some child of t. In the following, we write w for the width of the tree decomposition used by our algorithm, which may be larger than $w(G)$ if an approximation is used.

We assume we have a *nice tree decomposition*, which is a binary tree decomposition rooted at some node $r \in V(\mathbb{T})$ with $X_r = \emptyset$ and with four specific node types:

- **Leaf node**: Let t be a node with no child nodes. Then t is a *Leaf node* iff $X_t = \emptyset$.
- **Introduce node**: Let $t_{\text{parent}}, t_{\text{child}}$ be two nodes in $V(\mathbb{T})$, where t_{parent} has exactly one child t_{child}. Then t_{parent} is an *Introduce node* iff $X_{t_{\text{parent}}} = X_{t_{\text{child}}} \cup \{v\}$ for some $v \in V(G)$.
- **Forget node**: Let $t_{\text{parent}}, t_{\text{child}}$ be two nodes in $V(\mathbb{T})$, where t_{parent} has exactly one child t_{child}. Then t_{parent} is a *Forget node* iff $X_{t_{\text{parent}}} \cup \{v\} = X_{t_{\text{child}}}$ for some $v \in V(G)$.
- **Join node**: Let $t_{\text{parent}}, t^1_{\text{child}}, t^2_{\text{child}}$ be three nodes in $V(\mathbb{T})$, where t_{parent} has exactly two children t^1_{child} and t^2_{child}. Then t_{parent} is a *Join node* iff $X_{t_{\text{parent}}} = X_{t^1_{\text{child}}} = X_{t^2_{\text{child}}}$.

From any tree decomposition of width w, a nice tree decomposition of the same width with $\mathcal{O}(|V(G)| \cdot w)$ nodes can be computed in polynomial time [13].

The algorithm works as follows: Given an instance $G = (V, E)$ and a cost function $c : E \to \mathbb{R}^d$, it first computes a nice tree decomposition $(\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$. In each leaf node the Pareto sets are initialized as empty sets. Then, for each node t, depending on its type, the algorithm recursively computes the set of Pareto-optimal solutions for each subproblem at this node. We distinguish between the functions `computeLeafNode`, `computeIntroduceNode`, and `computeForgetNode`, corresponding to their respective node types, and show correctness assuming that the Pareto sets for all child nodes are correctly computed.

3 Multiobjective s-t cut problem

We consider the problem of computing all Pareto-optimal s-t cuts in a graph. Given a graph $G' = (V \cup \{s, t\}, E')$ with $n = |V|$ and a cost function $c' : E' \to \mathbb{R}^d$, a cut corresponds to a subset $S \subseteq V$. The cost of a cut S is defined as the sum of costs of all edges crossing from $S \cup \{s\}$ to $(V \setminus S) \cup \{t\}$. Let $\delta(S) := \{\{u, v\} \in E' \mid u \in S \cup \{s\}, v \in (V \setminus S) \cup \{t\}\}$ be the set of edges cut by S. The cost of S is given by $\sum_{e \in \delta(S)} c'(e)$. We now wish to compute all s-t cuts that are Pareto-optimal. Without loss of generality, we assume $\{s, t\} \notin E'$, since such an edge only introduces a constant shift in cost to every Pareto-optimal solution.

We denote $\delta(A, B) := \{\{u, v\} \in E' \mid u \in A, v \in B\}$ for $A, B \subseteq V(G')$ as the set of edges between vertices in A and B that are being cut. We define $c : 2^{V(G')} \times 2^{V(G')} \to \mathbb{R}^d$ with $c(A, B) = \sum_{e \in \delta(A,B)} c'(e)$ for any subsets $A, B \subseteq V(G')$. For simplicity, for a single vertex $p \in V(G')$ and subset $A \subseteq V(G')$, we write $c(p, A)$ instead of $c(\{p\}, A)$.

Let $G = G'[V]$ be the subgraph induced by V. Given a nice tree decomposition $(\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ for G, we compute for each node $t \in V(\mathbb{T})$ and subset $S \subseteq X_t$ the Pareto set $\mathcal{P}_t^S$ consisting of all Pareto-optimal solutions $p \subseteq V_t$ with $p \cap X_t = S$. We refer to such a subset S as a *selection*. Every such solution p has cost $c'(\delta(p)) = c(p, V \setminus p) + c(p, t) + c(V \setminus p, s)$. We refer to such an instance by a pair (t, S).

▶ **Theorem 2.** *For an arbitrary s-t cut instance $(G' = (V \cup \{s, t\}, E'), c')$ with cost function $c' : E \to \mathbb{R}^d$, we can compute the set of Pareto-optimal s-t cuts in time $\mathcal{O}(nw \cdot 2^w \cdot p_{\max}^2 \log^{\max\{d-2,1\}}(p_{\max}^2))$ if a nice tree decomposition of graph $G = G'[V]$ with width w is provided.*

In the following we describe how to compute the Pareto set for each node type. For each instance (t, S), the set of Pareto-optimal solutions is stored in an entry $D[t, S]$. For simplicity, we assume that $D[t, S]$ contains the actual corresponding Pareto set. Since the algorithm only depends on the cost vectors and selection S, in the actual implementation we will only save their cost vectors and reconstruct each solution by additionally storing pointers to the solutions from which it originated. Since we assume d as a constant, the encoding length of every Pareto-optimal solution is constant as well. Proofs of the following lemmas are provided in the full version of the paper.

`computeLeafNode`(t, S): For a leaf node t, we have only one solution $D[t, \emptyset] = \{\emptyset\}$.

`computeIntroduceNode`(t, S): Assume we have $X_t = X_{t'} \cup \{v\}$. If $v \notin S$, then $D[t, S] = D[t', S]$. If $v \in S$, then v is only adjacent to vertices in $X_{t'}$ and $V \setminus V_t$. If $X_{t'}$ is fixed, placing v on the same side of the cut as s only changes the cost of every solution by a fixed amount. Therefore we have $D[t, S] = D[t', S \setminus \{v\}]$, and their costs change by $c(v, X_{t'} \cup (V \setminus V_t) \cup \{t\}) - 2c(S \setminus \{v\}, v) - c(s, v)$.

▶ **Lemma 3.** *If node t introduces some vertex v and has a child t' for which $D[t', S]$ has been computed, then* `computeIntroduceNode`(t, S) *computes $D[t, S]$ in time $\mathcal{O}(p_{\max})$ for any partition S.*

`computeForgetNode`(t, S): Assume we have $X_t = X_{t'} \setminus \{v\}$. To obtain $D[t, S]$, we compute the union of the sets $D[t', S]$ and $D[t', S \cup \{v\}]$ and then remove all dominated solutions.

▶ **Lemma 4.** *If node t forgets some vertex v and has a child t' for which all possible sets $D[t', S']$ have been computed, then* `computeForgetNode`(t, S) *computes $D[t, S]$ in time $\mathcal{O}(p_{\max} \log^{d-2}(p_{\max}))$.*

For the correctness of `computeJoinNode`(t, S), the following lemma will be useful:

▶ **Lemma 5.** *For each Pareto-optimal solution $p \subseteq V$ and any node $t \in V(\mathbb{T})$, the solution $p \cap V_t$ is Pareto-optimal in the instance $(t, X_t \cap p)$.*

`computeJoinNode`$(t, \mathcal{S})$: Assume we have $X_t = X_{t_1} = X_{t_2}$. Since X_t separates $V_{t_1} \setminus X_t, V_{t_2} \setminus X_t$ and $V \setminus V_t$, we can combine every solution from $D[t_1, S]$ with every solution from $D[t_2, S]$ and then remove all dominated ones.

▶ **Lemma 6.** *If node $t \in V(\mathbb{T})$ is a join node with children $t_1, t_2 \in V(\mathbb{T})$ and sets $D[t_1, S]$ and $D[t_2, S]$ have been correctly computed, then* `computeJoinNode`(t, S) *computes $D[t, S]$ in time $\mathcal{O}(p_{\max}^2 \log^{\max\{d-2,1\}}(p_{\max}^2))$.*

Proof of Theorem 2. The correctness follows directly from the previous lemmas for the different node types, and from $D[r, \emptyset]$ being the Pareto set for the entire instance.

The runtime is dominated by the join nodes. Our nice tree decomposition contains $\mathcal{O}(nw)$ nodes, and for each node t the number of subsets $S \subseteq X_t$ is bounded by 2^{w+1}. Computing $D[t, S]$ at a join node takes time $\mathcal{O}(p_{\max}^2 \log^{\max\{d-2,1\}}(p_{\max}^2))$. Hence, the total running time is bounded by $\mathcal{O}(nw \cdot 2^w \cdot p_{\max}^2 \log^{\max\{d-2,1\}}(p_{\max}^2))$. ◀

The technique of computing the Pareto set for the s-t cut problem can also be applied to other multiobjective optimization problems as well. To illustrate this, we show how to compute the Pareto set for the multiobjective MST problem and the multiobjective TSP problem. Proofs of the following theorems are provided in the full version of the paper.

▶ **Theorem 7.** *For an arbitrary multiobjective spanning tree instance $(G = (V, E), c)$ with $d \geq 2$ objectives and cost function $c : E \to \mathbb{R}^d$, we can compute the set of Pareto-optimal spanning trees in time $\mathcal{O}(n(2w)^{\mathcal{O}(w)} \cdot p_{\max}^2 \log^{\max\{d-2,1\}}((2w)^{\mathcal{O}(w)} \cdot p_{\max}^2))$, if a nice tree decomposition of graph G with width w is provided.*

▶ **Theorem 8.** *For an arbitrary multiobjective traveling salesman instance $(G = (V, E), c)$ with $d \geq 2$ objectives and cost function $c : E \to \mathbb{R}^d$, we can compute the set of all Pareto-optimal tours in time $\mathcal{O}(n(3w+3)^{\mathcal{O}(w)} \cdot p_{\max}^2 \cdot \log^{\max\{d-2,1\}}((3w+3)^{\mathcal{O}(w)} \cdot p_{\max}^2))$, if a nice tree decomposition of graph G with width w is provided.*

4 Experiments

In this section we present the experimental evaluation of our proposed algorithm for a special case of the multiobjective s-t cut problem: the bicriteria polygon aggregation problem, introduced by Rottmann et al. [28]. An instance (T, P) consists of a set $P = \{p_1, \ldots, p_m\}$ of polygons and a set $T = \{t_1, \ldots, t_n\}$ of triangles. For any polygon $s \in T \cup P$, let $A(s) > 0$ denote its area and $P(s) > 0$ its perimeter. Given a set of triangles $S \subseteq T$, we define $A(S)$ as the total area of $S \cup P$ and $P(S)$ as the total perimeter of $S \cup P$. The goal is to compute the Pareto set $\mathcal{P}$ of subsets $S \subseteq T$ minimizing both $A(S)$ and $P(S)$.

As with many multiobjective optimization problems, the set of Pareto-optimal solutions for the bicriteria polygon aggregation problem can be of exponential size. In fact, one can more generally show that the bicriteria polygon aggregation problem can be seen as a generalization of the bicriteria knapsack problem. The statement on the size of the Pareto set follows as a corollary. The proof of the following theorem is provided in the full version of the paper.

▶ **Theorem 9.** *For every instance I of the bicriteria knapsack problem, there exists an instance I' of the bicriteria triangle aggregation problem such that there is a one-to-one correspondence between the Pareto-optimal solutions in I and I'.*

Our implementation follows the methodology outlined in Section 3, is written in Java, and was evaluated on the datasets used by Rottmann et al.[28], which consist of triangle aggregation problems derived from various cities and villages in Germany. The source code is available at `https://github.com/Tarek-pub/Bicriteria_Aggregation`.

We use the s-t cut construction from Rottmann et al. [28] to generating each instance. Additionally, we remove vertices corresponding to polygons from the graph G, resulting in a more compact, yet equivalent graph structure. To reduce complexity, weights (i.e. area and perimeter) are rounded to one decimal place (perimeter in decimals), and only unique-weight solutions are retained. Tree decompositions are computed using the JDrasil framework [4]. Several observations during development led to significant improvements in runtime and memory usage, which we integrated iteratively into the implementation and quantify through an analysis on the Ahrem dataset (Table 1). Experiments were run using 16 threads of an Intel Core i9-13900 with 100GB RAM. Multiple tree decompositions were generated for exactly 1 hour using 16 threads.

Further implementation details are provided in the full version of the paper, including an extended description of optimizations and a complete list of datasets with corresponding runtimes and memory usage.

As noted in Section 3, it suffices to store only the overall cost of a solution, rather than the full set of triangles included. The actual solutions can be reconstructed later by maintaining, additionally to the cost, pointers to the solutions from which the current one originated.

Table 1 Performance comparison of different algorithmic improvements on the dataset Ahrem.
*: These values are extrapolated based on sampled subproblems and scaled proportionally to estimate full runtimes (more details are provided in the full version of the paper).
**: For the versions before choosing a good root or tree decomposition, we chose the median rated root / tree decomposition in this table for a fair comparison.

Algorithm improvement	Runtime [h]	Storage usage [GiB]
Outsourcing & Pruning**	13767.8*	3707
Choosing a good root**	8571.6* (−38%)	3991 (+8%)
Choosing from multiple tree decompositions	7129.6* (−17%)	3647 (−9%)
Join node heuristic	162.0 (−98%)	3647 (−0%)
Join-forget nodes	8.2 (−95%)	1499 (−59%)
Introduce-join-forget nodes	7.0 (−15%)	122 (−92%)
Summary	**−99.95%**	**−96.71%**

We briefly describe the data structure used in our implementation, as well as a rough description of the algorithm adaptations. Each solution p is represented as a weighted pair $(A(p), P(p))$ for area and perimeter.

4.1 Outsourcing and Pruning

Initially, the implementation was unable to solve even small datasets, such as Osterloh (treewidth 14), due to RAM exhaustion reaching up to 100GB. To solve this problem, we implemented an efficient outsourcing strategy that stores all currently unneeded solutions onto a hard drive, keeping only the necessary solutions in RAM. Each solution is stored as a 16-byte entry in a dedicated *origin-pointer* file. Each DP table $D[t, S]$ is stored in a separate *surface-pointer* file, containing solution IDs and their weights. These files are kept on disk and only loaded into RAM when needed. This data structure allowed solving Osterloh without exceeding RAM limits, but other datasets remained unsolvable due to hard drive usage growing up to 2TB. To free up disk space, we remove obsolete surface-pointer files and, when necessary, run a slow in-place pruning step to clean the origin-pointer file. This involves recursively marking only reachable entries by following their pointers and compacting the file accordingly. Pruning is only triggered when space is critically low, based on conservative estimates of future storage use. These strategies prevented further storage issues, but many datasets were still unsolvable due to excessive runtime, particularly in join nodes. For example, the runtime required to solve the Ahrem dataset at this stage was estimated at approximately 13,768 hours, or one and a half year worth of CPU.

4.2 Choosing an appropriate tree decomposition

When generating multiple tree decompositions for the same graph, we observed that both the decomposition structure and the choice of the root node can significantly affect performance. Similar to Abseher et al. [1], we therefore generate multiple tree decompositions for the same graph and select one that is expected to yield good performance. However, our approach differs in two important aspects: First, since we consider a concrete algorithm rather than a general class of dynamic programming approaches, we can simulate its execution to approximate actual performance. Second, we do not only estimate runtime but also account for memory consumption, selecting the decomposition that minimizes a weighted combination of both. To this end, each decomposition is traversed in postorder, and we estimate the number of solutions at each node to predict resource usage.

Earlier versions of our approach focused solely on minimizing runtime, which did not always lead to reduced memory usage (see Table 1). Only in the final version did we optimize both runtime and memory simultaneously. Incorporating this performance estimator into the algorithm led to a significant improvement in runtime. For the Ahrem dataset we estimated a runtime decrease to around 7,130 hours, a 48% decrease in the estimated runtime.

In an experiment with 100 decompositions for the instance Osterloh, our estimator selected the decomposition that ranked first in both runtime and storage. The correlation between the predicted score and actual performance was 0.91 for runtime and 0.97 for storage. We obtained similarly strong results when estimating performance for different root node choices, again observing a high correlation between predicted and actual runtime and memory usage. These results suggest that our estimator reliably selects either the best tree decomposition or at least one that still performs well with regards to runtime and memory consumption.

4.3 Join node algorithm

Let t be a join node with children t_1 and t_2. For each subset $S \subseteq X_t$, we compute $\mathcal{P}_t^S = \mathcal{P}_{t_1} \oplus \mathcal{P}_{t_2}$ by combining all solutions $p_1 \in \mathcal{P}_{t_1}^S$ with all solutions $p_2 \in \mathcal{P}_{t_2}^S$ and subsequently remove all dominated solutions. To do this efficiently, we employ a lexicographical ordering of the input lists and process the combinations using a min-heap. Assume $|\mathcal{P}_{t_1}^S| \leq |\mathcal{P}_{t_2}^S|$. We initialize a min-heap with one heap node for each $p_1 \in \mathcal{P}_{t_1}^S$, each pointing to the first entry in $\mathcal{P}_{t_2}^S$. We then repeatedly extract the root of the min-heap, process the corresponding solution pair (p_1, p_2), and increment its pointer to reference the next solution $p_2' \in \mathcal{P}_{t_2}^S$ that follows p_2 in lexicographical order. If such a solution p_2' exists, we update the heap accordingly, otherwise, we remove the heap node from the data structure. This process continues until the heap is empty, ensuring that every valid solution pair has been considered. This min-heap mechanism ensures that all candidate pairs are enumerated in lexicographical order, which enables an efficient check for Pareto-optimality.

4.4 Optimizing join node computations

As with many dynamic programming algorithms over a nice tree decomposition, join nodes are the main computational bottleneck. At each join node t we needed to compute $\mathcal{P} = \mathcal{P}_1 \oplus \mathcal{P}_2$ for two Pareto sets $\mathcal{P}_1, \mathcal{P}_2$ exactly $2^{|X_t|}$ many times. Computing $\mathcal{P}_1 \oplus \mathcal{P}_2$ for $d \geq 2$ can be done in $\mathcal{O}(p_{\max}^2 \log^{\max\{d-2,1\}}(p_{\max}^2))$. In the worst case, $\mathcal{P}_1 \oplus \mathcal{P}_2$ can contain up to $|\mathcal{P}_1| \cdot |\mathcal{P}_2|$ many solutions if every combination is Pareto-optimal. By considering every combination $(p_1, p_2) \in \mathcal{P}_1 \times \mathcal{P}_2$ and filtering out all dominated ones, this becomes very time-consuming. However, in practice we observed only a roughly linear growth in size relative to the larger input Pareto set. This discrepancy indicated that many combinations were computed that would ultimately be discarded. To circumvent this problem, we utilize a heuristic pruning strategy to efficiently exclude many non-Pareto-optimal combinations at the same time.

We construct a heuristic set $\mathcal{H}$ for $\mathcal{P}$ using a small subset of $\mathcal{P}_1$ and $\mathcal{P}_2$. Afterwards we partition $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{H}$ into multiple sections and compute lower bounds for each section in $\mathcal{P}_1, \mathcal{P}_2$ and upper bounds for $\mathcal{H}$. We then iterate over all combinations of lower bounds and compare them with the upper bounds. If a combination of lower bounds is entirely dominated by all upper bounds, we can conclude that no solution in these sections will be Pareto-optimal (PO) in $\mathcal{P}$ and safely skip them in the min-heap. To construct $\mathcal{H}$, we apply the same optimization recursively until a base case is reached. Furthermore, we apply multi-threading for each join node t, enabling parallel computation of multiple entries $D[t, S]$ at the same time.

Table 2 Overview over some of the datasets we managed to solve. For all datasets see the full version of the paper.

Dataset	w	Time [h]	Storage [GIB]	#PO Solutions	Percentage nonextreme	Graph #Vertices	Graph #Edges	$\lvert V(\mathbb{T})\rvert$
Osterloh	14	0.15	5.8	83055	99.43	1717	1887	7586
Ahrem	17	3.1	122	219969	98.99	5280	5607	21778
Lottbek	19	10.5	355	316567	99.48	5595	6101	24426
Erlenbach	22	70.7	1754	303565	99.47	5644	6284	25682

The heuristic pruning process introduces a trade-off between its own runtime and the efficiency gains from skipping combinations. By selecting appropriate parameters, we achieved a favorable balance, reducing the runtime by nearly 98% and ultimately solving Ahrem in under seven days.

4.5 Join-forget nodes

To further improve runtime and reduce memory usage, we analyzed recurring patterns in the algorithm process that allowed for optimization. In many instances, after computing a join node, many of these solutions were immediately discarded in subsequent forget nodes. To address this inefficiency, we introduce a new node type, called *join-forget* node, which merges a join node with subsequent forget nodes into a single operation. Formally, for a join node t in the tree decomposition, if its unique parent and all ancestors up to the next non-forget node are forget nodes, we replace t and all these forget nodes by a single join-forget node t'. This new node retains the original bag X_t and additionally stores the set F of forgotten vertices from the removed forget nodes.

We consider which lists would be merged in the upcoming forget nodes of the original join node while combining solutions in the min-heap. More specifically, for a join-forget node with forgotten vertices F, we only create a min-heap for each subset $S' \subseteq X_t \setminus F$. Each such heap allows us to efficiently iterate over all combinations from the union $\bigcup_{S \in \{S' \cup F' \mid F' \subseteq F\}} \{p_1 \cup p_2 \mid p_1 \in \mathcal{P}^S_{t_1}, p_2 \in \mathcal{P}^S_{t_2}\}$.

This setup also enhances our join node heuristic. Instead of computing a heuristic set $\mathcal{H}$ and upper bounds for each pair $\mathcal{P}^S_{t_1}, \mathcal{P}^S_{t_2}$ separately, we reuse the join-forget structure. Specifically, for each subset $S' \subseteq X_t \setminus F$, we compute the same heuristic set $\mathcal{H}$ for all $S \in \{S' \cup F' \mid F' \subseteq F\}$. We do so by using subsets of $\mathcal{P}^S_{t_1}$ and $\mathcal{P}^S_{t_2}$ for all such S. As before, we recursively apply this heuristic until a small base case is reached. A side effect of this strategy is that it significantly increases RAM usage, as each thread now requires not just three lists in memory (namely $\mathcal{P}^S_{t_1}, \mathcal{P}^S_{t_2}$ and $\mathcal{P}^S_{t_1} \oplus \mathcal{P}^S_{t_2}$), but up to $2^{|F|+1} + 1$ lists.

This strategy led to another significant boost in runtime for many of our datasets. For the Ahrem dataset, using join-forget nodes reduced the runtime by 95%, allowing us to solve the instance in almost 8 hours. Moreover, storage usage also decreased by 59%. We emphasize that the improvements are not solely due to the use of join-forget nodes, but by adapting the tree decomposition performance estimator accordingly as well.

4.6 Introduce-join-forget nodes

Together with the newly defined join-forget node, we searched for new patterns in the algorithm computations to reduce the storage consumption further. To this end, we introduce a second node type called *introduce-join-forget* node, which further generalizes the concept

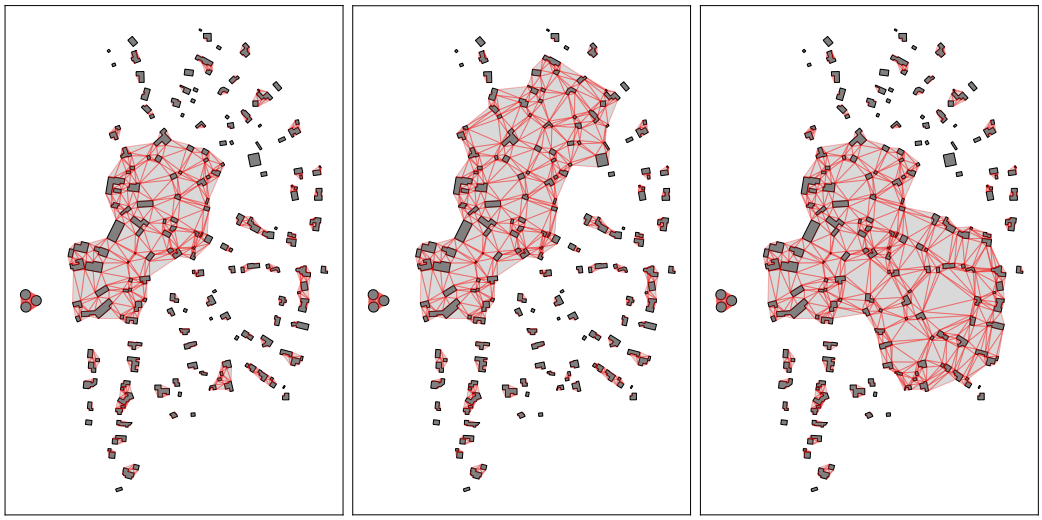


Figure 1 Three PO aggregations of the dataset Osterloh. Input polygons are filled dark gray, chosen triangles are filled light gray. Middle: nonextreme solution. Left/Right: Closest extreme solutions to the nonextreme solution.

of join-forget nodes. After replacing applicable structures with join-forget nodes, we consider each join-forget node t with child nodes t_1, t_2, where at least one child (t_1, t_2, or both) is an introduce node. We remove all consecutive introduce nodes among the children and store their introduced vertices as additional information in t.

A major inefficiency of standard introduce nodes arises from the exponential growth in possible subsets $S \subseteq X_t$. When a new vertex v is introduced, each existing solution $p \in \mathcal{P}_t^S$ for $S \subseteq X_t \setminus \{v\}$ is, with a constant weight adjustment, duplicated for both S and $S \cup \{v\}$. This significantly increases both memory usage and I/O overhead in the outsourcing step, making it a significant bottleneck. To resolve this, we avoid explicit duplication of solutions in introduce-join-forget nodes. Let $I \subseteq X_t$ be the set of vertices whose introduction was skipped on one side. If a set $\mathcal{P}^S$ is required for the min-heap computations and $S \cap I \neq \emptyset$, we instead load the solutions of the surface-pointer file of $\mathcal{P}^{S \setminus I}$ and add the constant weight adjustments according to $S \cap I$. Additionally, we create an origin-pointer entry for such a solution only if they are identified as Pareto-optimal in the min-heap. This strategy also reduces the growth of the origin-pointer file.

Together with adapting the performance estimator, this final improvement reduced the memory consumption by an additional 92%, resulting in a total of only 122GB of storage required for the Ahrem dataset, while also achieving a further 15% decrease in runtime, bringing the total runtime down to 7 hours.

5 Results

Using an efficient data structure and multiple algorithmic improvements described in Section 4, we were able to compute the full Pareto set for many datasets, including instances with treewidths of up to 22, within a reasonable amount of time. Table 2 shows a subset of datasets we successfully solved. All of these computations were performed on a high performance computing system. We used 96 threads of two Intel Xeon “Sapphire Rapids” 2.10GHz and about two terabytes of storage.

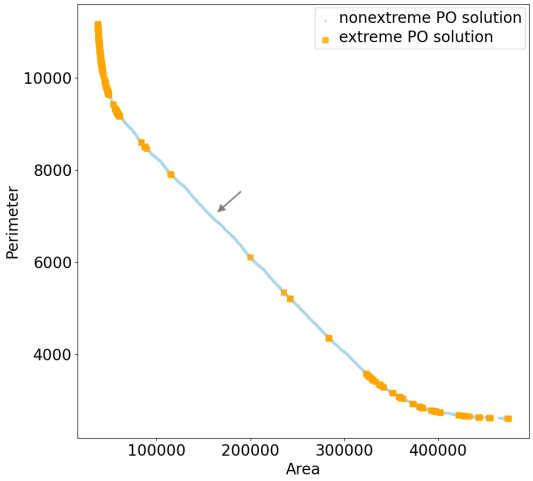


Figure 2 The weights of all PO solutions of Osterloh. The arrow indicates which nonextreme solution is shown in Figure 1.

For the polygon aggregation problem, it is known that all extreme solutions are hierarchically compatible [28]. In Figure 1 we illustrate this by showing two consecutive extreme solutions p_1, p_2 and one nonextreme solution p' that lies between them (i.e. $A(p_1) < A(p') < A(p_2)$ and $P(p_1) > P(p') > P(p_2)$). While the nonextreme solution p' is also a viable aggregation, it exhibits a significantly different structure compared to the extreme solutions. Focusing solely on extreme solutions may lead to large gaps in the solution space, as seen in Figure 2.

Across the datasets we solved, extreme solutions accounted for only less than one percent of the total PO solutions on average. As a result, computing the set of nonextreme solutions can therefore lead to a significant richer range of solutions for the user to choose from, while the extreme solutions are forced being hierarchical depending on each other.

We published a tool to interactively investigate all PO solutions for the dataset Osterloh. The tool is available at `https://github.com/Tarek-pub/Bicriteria_Aggregation_plotting`.

6 Conclusions

We presented the first algorithms for computing Pareto sets using dynamic programming over tree decompositions and showed that this framework can naturally be applied to various multiobjective optimization problems. The main motivation for our work was the article of Rottmann et al. [28], who raised the question of whether it is possible to compute the Pareto set in output-polynomial time. We also conducted an experimental analysis of the polygon aggregation problem on real-world instances and developed several techniques to improve both runtime and memory usage.

We want to highlight that the theoretical running times of our algorithms are only worst-case bounds. In practice, because not every bag in a tree decomposition has the maximum possible size and the number of join nodes is often relatively small, the actual running time will usually be smaller. Additionally, it is a common phenomenon for multiobjective optimization that Pareto sets are not too large for real-world inputs. Hence, the dependency on the size of the largest Pareto set is not prohibitive in practice. For problems like the multiobjective minimum spanning tree problem, the multiobjective TSP, and many other problems, this is further supported by theoretical results in smoothed analysis, which shows that Pareto sets are expected to be polynomially bounded when instances are randomly perturbed [10].

References

1 Michael Abseher, Frederico Dusberger, Nysret Musliu, and Stefan Woltran. Improving the efficiency of dynamic programming on tree decompositions via machine learning. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, IJCAI'15, pages 275–282. AAAI Press, 2015. doi:10.1613/jair.5312.

2 Pankaj K. Agarwal, David Eppstein, Leonidas J. Guibas, and Monika Rauch Henzinger. Parametric and kinetic minimum spanning trees. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 596–605. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743510.

3 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. doi:10.1137/0608024.

4 Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A modular library for computing tree decompositions. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK*, volume 75 of *LIPIcs*, pages 28:1–28:21, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SEA.2017.28.

5 René Beier, Heiko Röglin, Clemens Rösner, and Berthold Vöcking. The smoothed number of pareto-optimal solutions in bicriteria integer optimization. *Math. Program.*, 200(1):319–355, September 2023. doi:10.1007/s10107-022-01885-6.

6 Nadja Betzler, Rolf Niedermeier, and Johannes Uhlmann. Tree decompositions of graphs: Saving memory in dynamic programming. *Discret. Optim.*, 3(3):220–229, 2006. Graphs and Combinatorial Optimization. doi:10.1016/j.disopt.2006.05.008.

7 Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, STOC '93, pages 226–234, New York, NY, USA, 1993. ACM. doi:10.1145/167088.167161.

8 Hans L. Bodlaender and Fedor V. Fomin. Tree decompositions with small cost. *Discret. Appl. Math.*, 145(2):143–154, 2005. Structural Decompositions, Width Parameters, and Graph Labelings. doi:10.1016/j.dam.2004.01.008.

9 Fritz Bökler. *Output-sensitive complexity of multiobjective combinatorial optimization with an application to the multiobjective shortest path problem.* PhD thesis, Dortmund University, Germany, 2018. doi:10.17877/DE290R-19130.

10 Tobias Brunsch and Heiko Röglin. Improved smoothed analysis of multiobjective optimization. *J. ACM*, 62(1):4:1–4:58, 2015. doi:10.1145/2699445.

11 Günther Charwat and Stefan Woltran. Efficient problem solving on tree decompositions using binary decision diagrams. In Francesco Calimeri, Giovambattista Ianni, and Miroslaw Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings*, volume 9345 of *Lecture Notes in Computer Science*, pages 213–227, Cham, 2015. Springer. doi:10.1007/978-3-319-23264-5_19.

12 H. William Corley and I. Douglas Moon. Shortest paths in networks with vector weights. *Journal of Optimization Theory and Application*, 46(1):79–86, 1985. doi:10.1007/BF00938761.

13 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 1st edition, 2015. doi:10.1007/978-3-319-21275-3.

14 Tamal K. Dey. Improved bounds on planar k-sets and k-levels. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 156–161. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646104.

15 Matthias Ehrgott. Integer solutions of multicriteria network flow problems. *Investigacao Operacional*, 19:229–243, 1999. doi:10.1007/978-3-642-59179-2_2.

16 Matthias Ehrgott. *Multicriteria Optimization (2. ed.)*. Springer, 2005. doi:10.1007/3-540-27659-9.

17 Pierre Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making: Theory and Applications*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127, 1980. doi:10.1007/978-3-642-48782-8_9.

18 Kustaa Kangas, Mikko Koivisto, and Sami Salonen. A faster tree-decomposition based algorithm for counting linear extensions. *Algorithmica*, 82(8):2156–2173, 2020. doi:10.1007/s00453-019-00633-1.

19 Joshua Marc Könen, Heiko Röglin, and Tarek Stuck. Bicriteria Aggregation. Software, swhId: swh:1:dir:b6eacf189792239e9115cf288be27c6753b8df17 (visited on 2025-09-09). URL: https://github.com/Tarek-pub/Bicriteria_Aggregation, doi:10.4230/artifacts.24714.

20 Joshua Marc Könen, Heiko Röglin, and Tarek Stuck. Bicriteria Aggregation Plotting. InteractiveResource (visited on 2025-09-09). URL: https://github.com/Tarek-pub/Bicriteria_Aggregation_plotting, doi:10.4230/artifacts.24713.

21 Tuukka Korhonen. Single-exponential time 2-approximation algorithm for treewidth. *CoRR*, abs/2104.07463, 2021. doi:10.48550/arXiv.2104.07463.

22 Ankur Moitra and Ryan O'Donnell. Pareto optimal solutions for smoothed analysts. *SIAM J. Comput.*, 41(5):1266–1284, 2012. doi:10.1137/110851833.

23 Matthias Müller-Hannemann and Karsten Weihe. On the cardinality of the pareto set in bicriteria shortest path problems. *Ann. Oper. Res.*, 147(1):269–286, 2006. doi:10.1007/s10479-006-0072-1.

24 Adli Mustafa and Mark Goh. Finding integer efficient solutions for bicriteria and tricriteria network flow problems using DINAS. *Comput. Oper. Res.*, 25(2):139–157, 1998. doi:10.1016/S0305-0548(97)00027-0.

25 George L. Nemhauser and Zev Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969. doi:10.1287/mnsc.15.9.494.

26 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

27 Heiko Röglin and Shang-Hua Teng. Smoothed analysis of multiobjective optimization. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 681–690. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.21.

28 Peter Rottmann, Anne Driemel, Herman J. Haverkort, Heiko Röglin, and Jan-Henrik Haunert. Bicriteria aggregation of polygons via graph cuts. In Krzysztof Janowicz and Judith Anne Verstegen, editors, *11th International Conference on Geographic Information Science, GIScience 2021, September 27-30, 2021, Poznań, Poland (Virtual Conference) - Part II*, volume 208 of *LIPIcs*, pages 6:1–6:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.GIScience.2021.II.6.

29 Anders J. V. Skriver and Kim Allan Andersen. A label correcting approach for solving bicriterion shortest-path problems. *Comput. Oper. Res.*, 27(6):507–524, 2000. doi:10.1016/S0305-0548(99)00037-4.

On Estimating the Quantum ℓ_α Distance

Yupan Liu
Graduate School of Mathematics, Nagoya University, Japan

Qisheng Wang
School of Informatics, University of Edinburgh, UK

Abstract

We study the computational complexity of estimating the quantum ℓ_α distance $\mathrm{T}_\alpha(\rho_0, \rho_1)$, defined via the Schatten α-norm $\|A\|_\alpha := \mathrm{tr}(|A|^\alpha)^{1/\alpha}$, given $\mathrm{poly}(n)$-size state-preparation circuits of n-qubit quantum states ρ_0 and ρ_1. This quantity serves as a lower bound on the trace distance for $\alpha > 1$. For any constant $\alpha > 1$, we develop an efficient *rank-independent* quantum estimator for $\mathrm{T}_\alpha(\rho_0, \rho_1)$ with time complexity $\mathrm{poly}(n)$, achieving an *exponential* speedup over the prior best results of $\exp(n)$ due to Wang, Guan, Liu, Zhang, and Ying (*IEEE Trans. Inf. Theory* 2024). Our improvement leverages efficiently computable *uniform* polynomial approximations of *signed* positive power functions within quantum singular value transformation, thereby eliminating the dependence on the rank of the states.

Our quantum algorithm reveals a dichotomy in the computational complexity of the Quantum State Distinguishability Problem with Schatten α-norm (QSD_α), which involves deciding whether $\mathrm{T}_\alpha(\rho_0, \rho_1)$ is at least $2/5$ or at most $1/5$. This dichotomy arises between the cases of constant $\alpha > 1$ and $\alpha = 1$:

- For any $1 + \Omega(1) \leq \alpha \leq O(1)$, QSD_α is BQP-complete.
- For any $1 \leq \alpha \leq 1 + \frac{1}{n}$, QSD_α is QSZK-complete, implying that no efficient quantum estimator for $\mathrm{T}_\alpha(\rho_0, \rho_1)$ exists unless BQP = QSZK.

The hardness results follow from reductions based on new rank-dependent inequalities for the quantum ℓ_α distance with $1 \leq \alpha \leq \infty$, which are of independent interest.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Quantum query complexity; Theory of computation → Quantum information theory; Theory of computation → Quantum complexity theory

Keywords and phrases quantum algorithms, quantum state testing, trace distance, Schatten norm

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.106

Related Version *Full Version*: https://arxiv.org/abs/2505.00457v2 [52]

Funding *Yupan Liu*: Supported in part by the Ministry of Education, Culture, Sports, Science and Technology (MEXT) Quantum Leap Flagship Program (Q-LEAP) under Grant JPMXS0120319794, in part by Japan Science and Technology Agency (JST) Support for Pioneering Research Initiated by the Next Generation (SPRING) under Grant JPMJSP2125 and "THERS Make New Standards Program for the Next Generation Researchers", and in part by the Japan Society for the Promotion of Science (JSPS) Grants-in-Aid for Scientific Research (KAKENHI) under Grant 24H00071.
Qisheng Wang: Supported by the Engineering and Physical Sciences Research Council under Grant EP/X026167/1.

1 Introduction

Closeness testing of quantum states is a central topic in quantum property testing [58], which aims to develop (efficient) quantum testers for properties of quantum objects. This problem is also closely related to verifying the functionality of quantum devices, such as Q_0 and Q_1, which are commonly designed to prepare the respective n-qubit (mixed) quantum states ρ_0 and ρ_1. The goal of (tolerant) quantum state testing is to design efficient quantum


© Yupan Liu and Qisheng Wang;
licensed under Creative Commons License CC-BY 4.0



algorithms that test whether ρ_0 is 2/5-far from or 1/5-close to ρ_1 with respect to a given closeness measure. Notably, this problem generalizes classical (tolerant) distribution testing (see [20] and [35, Chapter 11]) from a non-commutative perspective.

When the "source codes" of distribution- or state-preparation circuits are given, a surprising correspondence was established between such closeness testing problems – measured by the ℓ_1 norm distance [66, 37] or entropy difference [36] – and interactive proof systems that admit statistical zero-knowledge (SZK). This correspondence links closeness testing problems to both complexity theory and cryptography. A similar correspondence was later identified in the quantum world: closeness testing of quantum states with respect to the trace distance (given by Schatten 1-norm) [77, 78], denoted by QSD, or the von Neumann entropy difference [10] was shown to be QSZK-complete.[1]

In contrast, when the closeness measure follows an ℓ_2-norm-like definition, such as the Hilbert-Schmidt distance or the quantum linear entropy, the corresponding closeness testing problems are in BQP using the SWAP test [17, 27]. Taken together, these results reveal a dichotomy in the complexity of closeness testing: when the measure is ℓ_1-norm-like, the problems are QSZK-hard and their query or sample complexities have *polynomial* dependence on the dimension or rank of the states; whereas for ℓ_2-norm-like measures, the problems are contained in BQP and their query or sample complexities are *rank-independent*.

What about the closeness testing problems with respect to generalizations that approximates the trace distance or the von Neumann entropy? The quantum ℓ_α distance, defined as $\mathrm{T}_\alpha(\rho_0, \rho_1) \coloneqq \frac{1}{2}\operatorname{tr}(|\rho_0 - \rho_1|^\alpha)^{1/\alpha}$ via the Schatten α-norm, generalizes both the trace distance ($\alpha = 1$) and the Hilbert-Schmidt distance ($\alpha = 2$). Similarly, the quantum q-Tsallis entropy $\mathrm{S}_q(\rho)$ extends both von Neumann entropy ($q = 1$) and quantum linear entropy ($q = 2$).

Interestingly, prior results show a divergence in behavior for closeness measures looser than the ℓ_2 norm: The closeness testing problem with respect to $\mathrm{T}_\alpha(\rho_0, \rho_1)$, denoted by QSD_α (see Definition 17), is in BQP only for *even* integer $\alpha \geq 2$ via the Shift test [27]; while for odd integers $\alpha \geq 3$, the query and sample complexities generally depend on the rank [71]. However, the techniques in [27] yield BQP algorithms for estimating $\mathrm{S}_q(\rho)$ for *all* integer $q \geq 2$. A recent work [53] further explored the closeness testing problem with respect to $\mathrm{S}_q(\rho_0) - \mathrm{S}_q(\rho_1)$, and extended the observed dichotomy from integers – where the transition occurs between $q = 1$ and $q \geq 2$ – to a continuous setting, showing a sharp distinction between $q = 1$ and any constant $q > 1$. These results naturally lead to an intriguing question:

▶ **Problem 1.** What is the computational complexity of the closeness testing problem with respect to $\mathrm{T}_\alpha(\rho_0, \rho_1)$? Does a similar dichotomy hold between $\alpha = 1$ and constants $\alpha > 1$, or does the complexity vary largely depending on whether α is even or odd?

Why study ℓ_α problems for possibly non-integer $\alpha > 1$? The trace distance ($\alpha = 1$) is a fundamental closeness measure of quantum states, capturing the maximum success probability of quantum state discrimination [41, 40] and playing a key role in applications such as the security of quantum key distribution [11, 63]. For $\alpha > 1$, such as $\alpha = 1.001$, the quantum ℓ_α distance provides a natural lower bound on the trace distance, and addressing Problem 1 could make this bound efficiently computable. Moreover, insights from ℓ_α problems have previously contributed to progress on well-studied ℓ_1 problems, as seen in [49].

Beyond their connections to ℓ_1 problems, ℓ_α problems for $\alpha > 1$ are of independent interest. In classical scenarios, they have applications in machine learning (e.g., [46]), as well as in streaming and sketching algorithms (e.g. [43]). In quantum scenarios, the Hilbert-

[1] The QSZK containment of the closeness testing problem with respect to the trace distance, denoted by $\mathrm{QSD}[a(n), b(n)]$, holds only in the polarizing regime $a(n)^2 - b(n) \geq 1/O(\log n)$, as shown in [77, 78]. A recent work [51] slightly improved the parameter regime for this containment.

Schmidt distance ($\alpha = 2$) is widely used in quantum information theory (e.g., [42, 61]), and more recently, has been leveraged in designing near-term (variational) quantum algorithms (e.g., [5, 28]). Consequently, positive answers to Problem 1 may offer new opportunities to refine, extend, or develop techniques relevant to these areas.

A classical counterpart to Problem 1 was investigated in [69] nearly a decade ago. The main takeaway aligns with [53]: For constant $\alpha > 1$, the sample complexity for distinguishing whether $\mathrm{TV}_\alpha(D_0, D_1)$ is at least $2/5$ or at most $1/5$ is *independent* of the dimension of the probability distributions D_0 and D_1,[2] fewer samples are needed as α increases. Classically, these upper bounds can be achieved by drawing a polynomial number of samples and computing the ℓ_∞ norm distance between the resulting empirical distributions. However, this approach does not directly extend to the quantum world: (1) quantum states ρ_0 and ρ_1 are generally not simultaneously diagonalizable; and (2) even when they are, estimating their eigenvalues associated with the unknown common eigenbasis remains challenging. Addressing these challenges is central to resolving Problem 1, which is the focus of our work.

1.1 Main results

We begin by stating our first main theorem, which provides a positive answer to Problem 1 when α lies in the range $1 + \Omega(1) \le \alpha \le O(1)$:

▶ **Theorem 2** (Quantum estimator for T_α, informal). *Given quantum query access to the state-preparation circuits of n-qubit quantum states ρ_0 and ρ_1, for any constant $\alpha > 1$, there is a quantum algorithm for estimating $\mathrm{T}_\alpha(\rho_0, \rho_1)$ within additive error $1/5$ and query complexity $O(1)$. If the state-preparation circuits have $\mathrm{poly}(n)$-size descriptions, then the algorithm' time complexity is $\mathrm{poly}(n)$. Thus, for any constant $\alpha > 1$, QSD_α is in* BQP.

More precisely, for a given additive error ϵ, the explicit query complexity of Theorem 2 is $O(1/\epsilon^{\alpha+1+\frac{1}{\alpha-1}})$ (see Theorem 14). In combination with the samplizer [74, 75], estimating $\mathrm{T}_\alpha(\rho_0, \rho_1)$ can be done using $\widetilde{O}(1/\epsilon^{3\alpha+2+\frac{2}{\alpha-1}})$ samples of ρ_0 and ρ_1 (see Theorem 16). Both upper bounds can be expressed as $\mathrm{poly}(1/\epsilon)$ for the regime $1 + \Omega(1) \le \alpha \le O(1)$. In addition, if the state-preparation circuits of ρ_0 and ρ_1 have size $L(n) = \mathrm{poly}(n)$, then Theorem 2 implies a quantum algorithm with time complexity $\widetilde{O}(L/\epsilon^{\alpha+1+\frac{1}{\alpha-1}})$, or equivalently, $\mathrm{poly}(n, 1/\epsilon)$.

Previous quantum algorithms for estimating the quantum ℓ_α distance for constant $\alpha > 1$ have all relied on its powered variant, specifically the powered quantum ℓ_α distance:

$$\Lambda_\alpha(\rho_0, \rho_1) \coloneqq \frac{1}{2}\,\mathrm{tr}(|\rho_0 - \rho_1|^\alpha) = 2^{\alpha-1} \cdot \mathrm{T}_\alpha(\rho_0, \rho_1)^\alpha.$$

Thus, for $1 < \alpha \le O(1)$, the estimates of $\mathrm{T}_\alpha(\rho_0, \rho_1)$ and $\Lambda_\alpha(\rho_0, \rho_1)$ are polynomially related.

When $\alpha > 1$ is an even integer, estimating $\mathrm{T}_\alpha(\rho_0, \rho_1)$ follows from a folklore result via the Shift test [27], using $O(1/\epsilon)$ queries or $O(1/\epsilon^2)$ samples.[3] However, for odd integer $\alpha > 1$, no efficient quantum algorithm is known in general. Closeness testing of quantum states with respect to $\mathrm{T}_\alpha(\rho_0, \rho_1)$ for $\alpha = 3$, with query complexity $O(1/\epsilon^{3/2})$, has been noted only in [32]. For general non-integer constants $\alpha > 1$, the quantum query complexity of estimating $\mathrm{T}_\alpha(\rho_0, \rho_1)$ was studied in [71], with polynomial dependence on the maximum rank r of ρ_0 and ρ_1. A technical comparison of our approach with this result is provided in Section 1.2.

[2] The closeness measure $\mathrm{TV}_\alpha(D_0, D_1)$ represents the classical ℓ_α distance based on the ℓ_α norm and generalizes the total variation distance, which is recovered at $\alpha = 1$.

[3] The sample complexity was noted in [62, Equations (83) and (84)].

By combining our efficient quantum estimator for $\mathrm{T}_\alpha(\rho_0,\rho_1)$ in the regime $1+\Omega(1) \le \alpha \le O(1)$ (Theorem 2) with our hardness results for QSD_α (Theorem 3), we identify a sharp phase transition between the case of $\alpha = 1$ and constant $\alpha > 1$, addressing Problem 1. For clarity, we summarize our main theorems and the quantitative bounds on quantum query and sample complexities, derived from both our results and prior work, in Table 1.

Table 1 Computational, query, and sample complexities of QSD_α for $1 \le \alpha \le O(1)$.

		$\alpha = 1$	$1 < \alpha \le 1+\frac{1}{n^{1+\delta}}$	$1+\frac{1}{n^{1+\delta}} < \alpha \le 1+\frac{1}{n}$	$1+\Omega(1) \le \alpha \le O(1)$
QSD_α		QSZK-complete$^{(*)}$ [77, 78]	QSZK-complete$^{(*)}$ Theorem 3(2)		BQP-complete Theorems 2 and 3(1)
Query	Upper Bound	$\widetilde{O}\left(r/\epsilon^2\right)$ [72]	$\widetilde{O}\left(r^{3+\frac{2}{\alpha}}/\epsilon^{4\alpha+2}\right)$ [71]		$O\left(1/\epsilon^{\alpha+1+\frac{1}{\alpha-1}}\right)$ Theorem 14
Compl.	Lower Bound	$\widetilde{\Omega}\left(r^{1/2}\right)$ [18]	$\widetilde{\Omega}\left(r^{1/2}\right)$ Theorem 27(2)	$\Omega\left(r^{1/3}\right)$ Theorem 27(1)	$\Omega(1/\epsilon)$ Theorem 22(1)
Sample	Upper Bound	$\widetilde{O}\left(r^2/\epsilon^5\right)$ [72]	$\mathrm{poly}(r,1/\epsilon)$ Noted in [71, Footnote 2]		$\widetilde{O}\left(1/\epsilon^{3\alpha+2+\frac{2}{\alpha-1}}\right)$ Theorem 16
Compl.	Lower Bound	$\Omega\left(r/\epsilon^2\right)$ [60]	$\Omega\left(r/\epsilon^2\right)$ Theorem 28		$\Omega\left(1/\epsilon^2\right)$ Theorem 22(2)

(*) For any $\alpha(n) \in \left[1, 1+\frac{1}{n}\right]$, the promise problem $\mathrm{QSD}_\alpha[a,b]$ is contained in QSZK only under the polarizing regime $a(n)^2 - b(n) \ge 1/O(\log n)$, which can be slightly improved when $\alpha = 1$ (see Footnote 1). However, establishing containment in a complexity class typically requires the natural regime $a(n) - b(n) \ge 1/\mathrm{poly}(n)$, as in Theorem 2.

Finally, we present our second main theorem, which addresses the computational hardness of QSD_α, as outlined in Theorem 3. In this context, PureQSD$_\alpha$ refers to a restricted variant of QSD_α (see also Definition 17), where the states of interest are pure.

▶ **Theorem 3** (Computational hardness of QSD_α)**.** *The promise problem* QSD_α *captures the computational power of the respective complexity classes in the corresponding regimes of* α*:*

(1) Easy regimes: *For any* $1 \le \alpha \le \infty$, PureQSD$_\alpha$ *is* BQP*-hard. As a corollary,* QSD_α *is* BQP*-complete for* $1+\Omega(1) \le \alpha \le O(1)$*.*

(2) Hard regimes: *For any* $1 \le \alpha \le 1+\frac{1}{n}$, QSD_α *is* QSZK*-complete, where the* QSZK *containment of* $\mathrm{QSD}_\alpha[a,b]$ *only holds for the polarizing regime* $a(n)^2 - b(n) \ge 1/O(\log n)$*.*

1.2 Proof techniques: BQP containment for α constantly above 1

At a high level, Quantum Singular Value Transformation (QSVT) [34] implies that the main challenge in designing a quantum algorithm based on a smooth function – e.g., Grover search [38] and the OR function, or the HHL algorithm [39] and the multiplicative inverse function (see [57] for more examples) – reduces to finding an efficiently computable polynomial approximation. Once such an approximation is obtained, the algorithm follows directly using techniques from [34], with its efficiency determined entirely by the polynomial's properties.

Now we focus on quantum algorithms for estimating the powered quantum ℓ_α distance. We begin by reviewing [71] and then provide a high-level overview of our approach.

The quantum query complexity of estimating the quantum ℓ_α distance for non-integer α was first considered in [71, Theorem IV.1]. Their approach begins with the identity

$$2\Lambda_\alpha(\rho_0,\rho_1) = \|\rho_0 - \rho_1\|_\alpha^\alpha = \mathrm{tr}\left(|\nu_-|^{\alpha/2}\Pi_{\nu_+}|\nu_-|^{\alpha/2}\right),$$

where $\nu_\pm = \rho_0 \pm \rho_1$ and Π_{ν_+} denotes the projector onto the support subspace of ν_+. According to this identity, they aim to prepare a quantum state that is a block-encoding of (normalized) $|\nu_-|^{\alpha/2}\Pi_{\nu_+}|\nu_-|^{\alpha/2}$.[4] To this end, they first prepare a quantum state that is a block-encoding of Π_{ν_+}, and then perform a unitary operator that is a block-encoding of $|\nu_-|^{\alpha/2}$ on it.[5] Finally, the (unnormalized) powered quantum ℓ_α distance, $\Lambda_\alpha(\rho_0, \rho_1)$, can be obtained by estimating the trace of $|\nu_-|^{\alpha/2}\Pi_{\nu_+}|\nu_-|^{\alpha/2}$ using quantum amplitude estimation [16]. After the error analysis, their approach was shown to have query complexity $\widetilde{O}(r^{3+1/\{\alpha/2\}}/\epsilon^{4+1/\{\alpha/2\}}) = \mathrm{poly}(r, 1/\epsilon)$.[6] The dependence on the rank is *inherent* in the approach of [71], as they have to prepare a rank-dependent quantum state that is a block-encoding of Π_{ν_+}, making the rank parameters unavoidable in the error analysis.

To overcome this technical issue, we utilize an identity different from theirs:

$$2\Lambda_\alpha(\rho_0, \rho_1) = \|\rho_0 - \rho_1\|_\alpha^\alpha = \mathrm{tr}\Big(\rho_0 \cdot \mathrm{sgn}(\nu_-) \cdot |\nu_-|^{\alpha-1}\Big) - \mathrm{tr}\Big(\rho_1 \cdot \mathrm{sgn}(\nu_-) \cdot |\nu_-|^{\alpha-1}\Big).$$

The idea is to estimate the terms $\mathrm{tr}(\rho_j \cdot \mathrm{sgn}(\nu_-) \cdot |\nu_-|^{\alpha-1})$ for $j \in \{0, 1\}$ individually, and then combine them to obtain an estimate of $\Lambda_\alpha(\rho_0, \rho_1)$. Our algorithm is sketched as follows:

1. Find a good approximation polynomial for $\mathrm{sgn}(x) \cdot |x|^{\alpha-1}$.
2. Implement a unitary block-encoding U of $\mathrm{sgn}(\nu_-) \cdot |\nu_-|^{\alpha-1}$ using Quantum Singular Value Transformation (QSVT) [34] and Linear Combinations of Unitaries (LCU) [23, 15], given the state-preparation circuits of ρ_0 and ρ_1.
3. Perform the Hadamard test [3] on U and ρ_j with outcome $b_j \in \{0,1\}$ for each $j \in \{0,1\}$.
4. Estimate $\Lambda_\alpha(\rho_0, \rho_1)$ by computing the expected value of $b_0 - b_1$.

Our algorithm is actually inspired by the trace distance estimation in [72], which corresponds to the case of $\alpha = 1$. However, the approach in [72] still has a rank-dependent query complexity of $\widetilde{O}(r/\epsilon^2)$, compared to the $\widetilde{O}(r^5/\epsilon^6)$ in [71].[7] Nevertheless, we discover an approach for estimating the quantum ℓ_α distance with a *rank-independent* complexity as long as α is constantly greater than 1. Specifically, we use the best uniform approximation polynomial $P_d(x)$ (of degree d) for the function $\mathrm{sgn}(x) \cdot |x|^q$, as given in [31, Theorem 8.1.1]:

$$\max_{x\in[-1,1]} \big|P_d(x) - \mathrm{sgn}(x) \cdot |x|^q\big| \to \frac{1}{d^q}, \text{ as } d \to \infty.$$

Our use of the best uniform approximation by polynomials is inspired by the recent work [53] on estimating the q-Tsallis entropy of quantum states for non-integer q, where they used the best uniform approximation polynomial for x^q in the non-negative range $[0, 1]$ (given in [67]). The difference is that in our case, we have to further consider the sign of x, thereby requiring the polynomial approximation to behave well in the negative part. It turns out that the polynomial approximation given in [31] is suitable for our purpose. Having noticed this, we then use the now standard techniques (used in [48, 53]) such as Chebyshev truncations and the de La Vallée Poussin partial sum (cf. [65]) to construct efficiently computable asymptotically best approximation polynomials such that

$$\max_{x\in[-1,1]} \left|P(x) - \frac{1}{2}\,\mathrm{sgn}(x) \cdot |x|^q\right| \le \epsilon, \quad \max_{x\in[-1,1]} |P(x)| \le 1, \text{ and } \deg(P) = O\left(\frac{1}{\epsilon^{1/q}}\right).$$

[4] See Definition 10 for the formal definition of block-encoding.
[5] This is because of the evolution of subnormalized density operators [71, Lemma II.2].
[6] Here, $\{x\} \coloneqq x - \lfloor x \rfloor$ denotes the fractional part of x.
[7] Some readers may wonder if our approach applies to trace distance estimation ($\alpha = 1$) to remove the rank dependence in quantum query and sample complexities in [72]. However, the answer is generally no, as the rank dependence is intrinsic to trace distance estimation due to the polynomial dependence of the rank in the quantum query and sample complexities lower bounds (see [52, Section 2.2.2] in the full version).

Using this efficiently computable polynomial (with $q = \alpha - 1$) and with further analysis, we can then estimate the quantum ℓ_α distance to within additive error ϵ with the desired query upper bound in Theorem 2. Moreover, using the (multi-)samplizer [74, 75], a quantum query-to-sample simulation, we can also achieve the desired sample upper bound.

1.3 Proof techniques: QSZK completeness for $\alpha > 1$ near 1

To establish the BQP- and QSZK-hardness results in Theorem 3, we reduce the promise problems QSD and PureQSD ($\alpha = 1$) to the corresponding promise problems QSD_α and $\mathrm{PureQSD}_\alpha$ for appropriate ranges of α. The key technique underlying these reductions is the following rank-dependent inequalities that generalize the case of $\alpha = 2$ from [24, 25]:

▶ **Theorem 4** (T_α vs. T, informal). *For any states ρ_0 and ρ_1 and $\alpha \in [1, \infty]$, it holds that:*

$$2^{1-\frac{1}{\alpha}} \cdot \mathrm{T}_\alpha(\rho_0, \rho_1) \leq \mathrm{T}(\rho_0, \rho_1) \leq 2\big(\mathrm{rank}(\rho_0)^{1-\alpha} + \mathrm{rank}(\rho_1)^{1-\alpha}\big)^{-\frac{1}{\alpha}} \cdot \mathrm{T}_\alpha(\rho_0, \rho_1). \tag{1}$$

For $\alpha = \infty$, the inequalities hold in the limit as $\alpha \to \infty$.

The proof of Theorem 4 follows from considering orthogonal positive semi-definite matrices ς_0 and ς_1 satisfying $\rho_0 - \rho_1 = \varsigma_0 - \varsigma_1$, and analyzing their properties carefully.

We then illuminate the hardness results in Theorem 3:

- For the easy regime, Equation (1) becomes an equality when both ρ_0 and ρ_1 are pure states. This equality implies the BQP-hardness of $\mathrm{PureQSD}_\alpha$, along with the query and sample complexity lower bounds for all $1 \leq \alpha \leq \infty$, establishing Theorem 3(1).
- For the hard regime, Equation (1) is *sensitive* to α. Particularly, for $\alpha = 1 + \frac{1}{n}$, if quantum states ρ_0 and ρ_1 are τ-far, meaning $\mathrm{T}(\rho_0, \rho_1) \geq \tau$, it follows only that $\mathrm{T}_{\alpha=1+\frac{1}{n}}(\rho_0, \rho_1) \geq \tau/2$. However, when $\alpha \leq 1 + \frac{1}{n^{1+\delta}}$ for any arbitrarily small constant δ, the same trace distance condition ensures that $\mathrm{T}_\alpha(\rho_0, \rho_1) \geq \tau$ as $n \to \infty$, leading to the QSZK hardness result in Theorem 3(2) and distinct query complexity lower bounds in Table 1.

Lastly, we explain the QSZK containment in the hard regime. Simply combining Theorem 4 and the QSZK containment of QSD from [77, 78] does not work, as the resulting QSZK containment of $\mathrm{QSD}_\alpha[a, b]$ holds only for $a(n)^2/2 - b(n) \geq 1/O(\log n)$, which is even weaker than the polarizing regime defined in Footnote 1. To address this, we establish a *partial* polarization lemma for T_α (Lemma 25), which ensures that for quantum states ρ_0 and ρ_1 where $\mathrm{T}(\rho_0, \rho_1)$ is either at least a or at most b, we can construct new quantum states $\widetilde{\rho}_0$ and $\widetilde{\rho}_1$ such that $\mathrm{T}_\alpha(\widetilde{\rho}_0, \widetilde{\rho}_1)$ is either at least $\frac{1}{2} - \frac{1}{2}e^{-k}$ or at most $1/16$, as long as the parameters a and b are in the polarizing regime. Theorem 3(2) follows by combining this partial polarization lemma for T_α with the polarization lemma for T in [77].

1.4 Discussion and open problems

While the quantum ℓ_α distance $\mathrm{T}_\alpha(\cdot, \cdot)$ and its powered version $\Lambda_\alpha(\cdot, \cdot)$ are almost computationally interchangeable for $1 \leq \alpha \leq O(1)$, their behavior differs greatly when $\alpha = \infty$:

- The quantity $\mathrm{T}_\infty(\rho_0, \rho_1)$ corresponds to the largest eigenvalue $\lambda_{\max}$ of $(\rho_0 - \rho_1)/2$. The associated promise problem QSD_∞ is BQP-hard and contains in QMA.[8] However, establishing a BQP containment appears challenging, as $(\rho_0 - \rho_1)/2$ does not directly admit an efficiently computable basis – unlike its classical counterpart in [69], which does.

[8] The verification circuit in the QMA containment simply follows from phase estimation [45], where a (normalized) eigenvector corresponding to $\lambda_{\max}$ serves as a witness state.

- The quantity $\Lambda_\infty(\rho_0, \rho_1)$ takes values in $\{0, 1/2, 1\}$ for any states ρ_0 and ρ_1, and is nonzero if and only if the states are orthogonal, with at least one being pure. Thus, even the pure-state-restricted variant of the associated promise problem, PurePoweredQSD$_\infty[1, 0]$, is $\mathsf{C_{=}P}$-hard (see [52, Appendix A] in the full version). Here, $\mathsf{C_{=}P} = \mathsf{coNQP}$ [2, 80], a subclass of PP that provides a precise variant of BQP, ensuring acceptance for all *yes* instances.

This fundamental difference between these quantities raises an intriguing question on QSD$_\infty$:

(a) What is the computational complexity of the promise problem QSD$_\infty$, defined by $\mathrm{T}_\infty(\cdot, \cdot)$? Can we show that QSD$_\infty$ is also in BQP, or is it inherently more difficult?

Another open problem concerns quantitative bounds for QSD$_\alpha$:

(b) Can the query and sample bounds in Table 1 be improved, particularly for the regime $1 + \Omega(1) \leq \alpha \leq O(1)$? Moreover, can tight bounds be established when the states have *small support*, analogous to the classical case in [69, Table 1]?

1.5 Related works

Schatten p-norm estimation $\mathrm{tr}(|A|^p)$ of $O(\log n)$-local Hermitian A on n qubits to within additive error $2^{n-p}\epsilon\|A\|^p$ for $\epsilon(n) \leq 1/\operatorname{poly}(n)$ and real $p(n) \leq \operatorname{poly}(n)$ was shown to be $\mathsf{DQC1}$-complete in [19]. Given a unitary block-encoding of a matrix A, in [56], they presented a quantum algorithm that estimates the Schatten p-norm $(\mathrm{tr}(|A|^p))^{1/p}$ to relative error ϵ for integer p, where a condition number κ satisfying $A \geq I/\kappa$ is required for the case of odd p.

The query complexity of N-dimensional quantum state certification (i.e., determine whether two quantum states are identical or ϵ-far) with respect to trace distance was shown to be $O(N/\epsilon)$ in [32]. The query complexity of trace distance estimation was shown to be $\widetilde{O}(r^5/\epsilon^6)$ in [71] and later improved to $\widetilde{O}(r/\epsilon^2)$ in [72], where r is the rank of the quantum states, confirming a conjecture in [25] that low-rank trace distance estimation is in BQP. Both Low-rank trace distance and fidelity estimations are known to be BQP-complete [1, 72]. Based on the approach of [72], space-bounded quantum state discrimination with respect to trace distance was shown to be BQL-complete in [48]. In addition to trace distance, fidelity is another important measure of the closeness between quantum states. The query complexity of fidelity estimation was shown to be $\widetilde{O}(r^{12.5}/\epsilon^{13.5})$ in [76] and later improved to $\widetilde{O}(r^{5.5}/\epsilon^{6.5})$ in [71] and to $\widetilde{O}(r^{2.5}/\epsilon^5)$ in [33]. Recently, the query complexity of pure-state trace distance and fidelity estimations was shown to be $\Theta(1/\epsilon)$ in [70] and was recently extended in [29] to estimating fidelity of a mixed state to a pure state.

In addition to the query complexity, the sample complexity has also been studied in the literature. In [7], the sample complexity of N-dimensional quantum state certification was shown to be $\Theta(N/\epsilon^2)$ with respect to trace distance and $\Theta(N/\epsilon)$ with respect to fidelity. The sample complexity of trace distance estimation is known to be $\widetilde{O}(r^2/\epsilon^5)$ in [72] and that of fidelity estimation is known to be $\widetilde{O}(r^{5.5}/\epsilon^{12})$, where r is the rank of quantum states. The sample complexity of pure-state squared fidelity estimation is known to be $\Theta(1/\epsilon^2)$ via the SWAP test [17], where the matching lower bound was given in [4]. Recently, the sample complexity of pure-state trace distance and fidelity estimations was shown to be $\Theta(1/\epsilon^2)$ in [73], which was achieved by using the samplizer in [75].

2 Preliminaries

We assume a fundamental knowledge of quantum computation and quantum information theory. For an introduction, we refer the reader to the textbook [59].

We adopt the following notations throughout the paper: (1) $[n] \coloneqq \{1, 2, \ldots, n\}$; (2) $\widetilde{O}(f)$ denotes $O(f \operatorname{polylog}(f))$, while $\widetilde{\Omega}(f)$ denotes $\Omega(f/\operatorname{polylog}(f))$; and (3) $|\bar{0}\rangle$ represents $|0\rangle^{\otimes a}$ for integer $a > 1$. In addition, the Schatten α-norm of a matrix A is defined as $\|A\|_\alpha \coloneqq \left(\operatorname{tr}(|A|^\alpha)\right)^{1/\alpha}$, where $|A| \coloneqq \left(A^\dagger A\right)^{1/2}$. For simplicity, we use the notation $\|A\|$ to denote the operator norm (equivalently, the Schatten ∞-norm) of a matrix A.

2.1 Closeness measures for quantum states

We start by defining the trace distance and providing useful properties of this distance:

▶ **Definition 5** (Trace distance). *Let ρ_0 and ρ_1 be two quantum states that are mixed in general. The trace distance between ρ_0 and ρ_1 is defined by*

$$\mathrm{T}(\rho_0, \rho_1) \coloneqq \frac{1}{2} \operatorname{tr}(|\rho_0 - \rho_1|) = \frac{1}{2} \operatorname{tr}\Big(\left((\rho_0 - \rho_1)^\dagger (\rho_0 - \rho_1)\right)^{1/2}\Big).$$

Notably, the trace distance is a distance metric (e.g., [79, Lemma 9.1.8]).

Next, we define the quantum ℓ_α distance and its *powered* version, which generalize the trace distance ($\alpha = 1$) using the Schatten norm. Notably, the quantum ℓ_α distance coincides with the Hilbert-Schmidt distance when $\alpha = 2$:

▶ **Definition 6** (Quantum ℓ_α distance and its powered version). *Let ρ_0 and ρ_1 be two quantum states that are mixed in general. The quantum ℓ_α distance $\mathrm{T}_\alpha(\cdot, \cdot)$ and its powered version $\Lambda_\alpha(\cdot, \cdot)$ between ρ_0 and ρ_1 are defined as follows*:

$$\mathrm{T}_\alpha(\rho_0, \rho_1) \coloneqq \frac{1}{2} \|\rho_0 - \rho_1\|_\alpha \quad \text{and} \quad \Lambda_\alpha(\rho_0, \rho_1) \coloneqq \frac{1}{2} \|\rho_0 - \rho_1\|_\alpha^\alpha.$$

Here, the Schatten α-norm of $\rho_0 - \rho_1$ is given by $\|\rho_0 - \rho_1\|_\alpha \coloneqq \left(|\rho_0 - \rho_1|^\alpha\right)^{1/\alpha}$.

By the monotonicity of the Schatten norm, e.g., [6, Equation (1.31)], it holds that:

$$\forall \alpha \geq 1, \quad 0 \leq \Lambda_\alpha(\rho_0, \rho_1) \leq \mathrm{T}_\alpha(\rho_0, \rho_1) \leq \mathrm{T}(\rho_0, \rho_1) \leq 1.$$

As a corollary of the Davis convexity theorem [26], the quantum ℓ_α distance $\mathrm{T}_\alpha(\cdot, \cdot)$ also serves as a distance metric, whereas its powered version $\Lambda_\alpha(\cdot, \cdot)$ does not. Further inequalities for the trace distance and the quantum ℓ_α distance are given in the full version, particularly in [52, Section 2.1].

Lastly, we require the following relationship for additive error estimation between the quantum ℓ_α distance and its powered version. The proof is provided in the full version, specifically in [52, Proposition 2.5].

▶ **Proposition 7** (T_α vs. powered T_α). *The quantum ℓ_α distance $\mathrm{T}_\alpha(\cdot, \cdot)$ and its powered version $\Lambda_\alpha(\cdot, \cdot)$ are related through the equality $\mathrm{T}_\alpha(\rho_0, \rho_1) = 2^{\frac{1}{\alpha}-1} \cdot \Lambda_\alpha(\rho_0, \rho_1)^{\frac{1}{\alpha}}$. Accordingly, if x is an estimate of $\Lambda_\alpha(\rho_0, \rho_1)$ to within additive error ϵ, then $2^{\frac{1}{\alpha}-1} \cdot x^{\frac{1}{\alpha}}$ serves as an estimate of $\mathrm{T}_\alpha(\rho_0, \rho_1)$ to within additive error $2^{\frac{1}{\alpha}-1} \cdot \epsilon^{\frac{1}{\alpha}}$.*

2.2 Closeness testing of quantum states via state-preparation circuits

We begin by defining the closeness testing of quantum states with respect to the trace distance, denoted as $\mathrm{QSD}[a, b]$,[9] and two variants of this problem:

[9] While Definition 8 aligns with the classical counterpart of QSD defined in [66, Section 2.2], it is slightly less general than the definition in [77, Section 3.3]. Specifically, Definition 8 assumes that the input length m and the output length n are *polynomially equivalent*, whereas [77, Section 3.3] allows for cases where the output length (e.g., a single qubit) is *much smaller* than the input length.

▶ **Definition 8** (Quantum State Distinguishability Problem, QSD, adapted from [77, Section 3.3]). *Let Q_0 and Q_1 be quantum circuits acting on m qubits ("input length") and having n specified output qubits ("output length"), where $m(n)$ is a polynomial function of n. Let ρ_i denote the quantum state obtained by running Q_i on state $|0\rangle^{\otimes m}$ and tracing out the non-output qubits. Let $a(n)$ and $b(n)$ be efficiently computable functions. Decide whether*:

- Yes: *A pair of quantum circuits (Q_0, Q_1) such that $\mathrm{T}(\rho_0, \rho_1) \geq a(n)$;*
- No: *A pair of quantum circuits (Q_0, Q_1) such that $\mathrm{T}(\rho_0, \rho_1) \leq b(n)$.*

Furthermore, we denoted the restricted version, where ρ_0 and ρ_1 are pure states, as PureQSD.

In this work, we consider the *purified quantum access input model*, as defined in [77], in both white-box and black-box scenarios:

- **White-box input model**: The input of the problem QSD consists of descriptions of polynomial-size quantum circuits Q_0 and Q_1. Specifically, for $b \in \{0, 1\}$, the description of Q_b includes a sequence of polynomially many 1- and 2-qubit gates.
- **Black-box input model**: In this model, instead of providing the descriptions of the quantum circuits Q_0 and Q_1, only query access to Q_b is allowed, denoted as O_b for $b \in \{0, 1\}$. For convenience, we also allow query access to $Q_b^\dagger$ and controlled-Q_b, denoted by $O_b^\dagger$ and controlled-O_b, respectively.

In addition to query complexity, defined within the black-box input model, *sample complexity* refers to the number of copies of quantum states ρ_0 and ρ_1 needed to accomplish a specific closeness testing task. Useful lemmas on computational hardness and quantitative lower bounds for query and sample complexities are available in the full version, specifically in [52, Section 2.2].

2.3 Polynomial approximations

We now present useful tools on best uniform polynomial approximations. In addition, definitions and lemmas on Chebyshev expansion and truncations can be found in the full version, particularly in [52, Section 2.3.2].

Let $f(x)$ be a continuous function defined on the interval $[-1, 1]$ that we aim to approximate using a polynomial of degree at most d. We define P_d^* as a *best uniform approximation* on $[-1, 1]$ to f of degree d if, for any degree-d polynomial approximation P_d of f, it holds that $\max_{x\in[-1,1]}|f(x) - P_d^*(x)| \leq \max_{x\in[-1,1]}|f(x) - P_d(x)|$.

The best uniform (polynomial) approximation of positive (constant) powers $|x|^\alpha$ was first established by Serge Bernstein [14, 13]. However, the focus here is on the best uniform approximation of *signed* positive powers $\mathrm{sgn}(x)|x|^\alpha$, as stated in Lemma 9. This result is often attributed to Bernstein's work (see, e.g., [68, Equation (10.2)]), and a proof of a more general version can be found in [31, Theorem 8.1.1].

▶ **Lemma 9** (Best uniform approximation of signed positive powers, adapted from [31, Theorem 8.1.1]). *For any positive real (constantly large) order α, let $P_d^* \in \mathbb{R}[x]$ be the best uniform polynomial approximation for $f(x) = \mathrm{sgn}(x)|x|^\alpha$ of degree $d = \left\lceil (\beta_\alpha/\epsilon)^{1/\alpha} \right\rceil$, where β_α is a constant depending on α. Then, for sufficiently small ϵ, $\max_{x\in[-1,1]} |P_d^*(x) - f(x)| \leq \epsilon$.*

2.4 Quantum algorithmic toolkit

In this subsection, we recap the quantum singular value transformation. Additional quantum algorithmic tools, including useful quantum algorithmic subroutines and the quantum samplizer can be found in the full version, specifically in [52, Sections 2.4.2 and 2.4.3].

2.4.1 Quantum singular value transformation

We start by defining block-encoding:

▶ **Definition 10** (Block-encoding). *A linear operator A on an $(n+a)$-qubit Hilbert space is said to be an (α, a, ϵ)-block-encoding of an n-qubit linear operator B, if $\|\alpha(\langle 0|^{\otimes a} \otimes I_n)A(|0\rangle^{\otimes a} \otimes I_n) - B\| \leq \epsilon$, where I_n is the n-qubit identity operator and $\|\cdot\|$ is the operator norm.*

Then, we state the quantum singular value transformation:

▶ **Lemma 11** (Quantum singular value transformation, [34, Theorem 31]). *Suppose that unitary operator U is a (α, a, ϵ)-block-encoding of Hermitian operator A, and $P \in \mathbb{R}[x]$ is a polynomial of degree d with $|P(x)| \leq \frac{1}{2}$ for $x \in [-1, 1]$. Then, we can implement a quantum circuit $\tilde{U}$ that is a $(1, a+2, 4d\sqrt{\epsilon/\alpha} + \delta)$-block-encoding of $P(A/\alpha)$, by using $O(d)$ queries to U and $O((a+1)d)$ one- and two-qubit quantum gates. Moreover, the classical description of $\tilde{U}$ can be computed in deterministic time* $\mathrm{poly}(d, \log(1/\delta))$.

3 Efficient quantum algorithms for estimating quantum ℓ_α distance

In this section, we present efficient quantum algorithms for estimating the quantum ℓ_α distance $\mathrm{T}_\alpha(\rho_0, \rho_1)$ when $\alpha \geq 1 + \Omega(1)$. These algorithms utilize either queries to state-preparation circuits or samples of the states ρ_0 and ρ_1. The core of our approach is an *efficient* uniform approximation to *signed* positive constant power functions (Lemma 12), which provides a uniform error bound over the entire interval $[-1, 1]$.

This uniform polynomial approximation enables a query-efficient quantum algorithm for estimating $\mathrm{T}_\alpha(\rho_0, \rho_1)$ through its powered version $\Lambda_\alpha(\rho_0, \rho_1)$, as shown in Theorem 14. As a result, we establish a BQP containment of the promise problem QSD_α defined in Section 4. Additionally, by leveraging the multi-samplizer in [73], we devise a sample-efficient quantum algorithm for estimating $\mathrm{T}_\alpha(\rho_0, \rho_1)$, detailed in Theorem 16.

3.1 Efficient uniform approximations of signed positive powers

Using the averaged Chebyshev truncation specified in [52, Section 2.3.2] in the full version, we provide an *efficiently computable* uniform polynomial approximation of *signed* positive constant powers (see also [52, Lemma 3.1]):

▶ **Lemma 12** (Efficient uniform polynomial approximation of signed positive powers). *Let α be a positive real (constantly large) number. For any $\epsilon \in (0, 1/2)$, there is a degree-d polynomial $P_d \in \mathbb{R}[x]$, where $d = \left\lceil (\beta'_\alpha/\epsilon)^{1/\alpha} \right\rceil$ and β'_α is a constant depending on α, that can be deterministically computed in $\widetilde{O}(d)$ time. For sufficiently small ϵ, it holds that*:

$$\max_{x\in[-1,1]} \left| \frac{1}{2}\mathrm{sgn}(x)|x|^\alpha - P_d(x) \right| \leq \epsilon \quad \textit{and} \quad \max_{x\in[-1,1]} |P_d(x)| \leq 1.$$

More specifically, the degree-d uniform approximation of signed positive powers (see Lemma 9) ensures that the degree-$(2d-1)$ averaged Chebyshev truncation (given in [52, Equation 2.3]) achieves almost the same approximation error bound, with efficiently computable Chebyshev coefficients. The complete proof is available in the full version, specifically in [52, Section 3.1].

3.2 Quantum ℓ_α distance estimation for constantly large $\alpha > 1$

3.2.1 Query-efficient quantum algorithm for estimating powered T_α

We now present efficient quantum query algorithms for estimating $\Lambda_\alpha(\rho_0, \rho_1)$ and $\mathrm{T}_\alpha(\rho_0, \rho_1)$, as presented in [52, Lemma 3.2] in the full version, and the complete proof is provided in [52, Section 3.2.1]:

▶ **Lemma 13** (Powered quantum ℓ_α distance estimation via queries). *Suppose that Q_0 and Q_1 are unitary operators that prepare purifications of mixed quantum states ρ_0 and ρ_1, respectively. For every constantly large $\alpha \geq 1 + \Omega(1)$, there is a quantum query algorithm that estimates $\Lambda_\alpha(\rho_0, \rho_1)$ to within additive error ϵ by using $O(1/\epsilon^{1+\frac{1}{\alpha-1}})$ queries to Q_0 and Q_1.*

By combining Proposition 7 with Lemma 13 for additive error ϵ^α, we obtain a quantum query algorithm for estimating $\mathrm{T}_\alpha(\rho_0, \rho_1)$ when $\alpha \geq 1 + \Omega(1)$ is constantly large, as also stated in [52, Theorem 3.3] in the full version:

▶ **Theorem 14** (Quantum ℓ_α distance estimation via queries). *Suppose that Q_0 and Q_1 are unitary operators that prepare purifications of mixed quantum states ρ_0 and ρ_1, respectively. For every constantly large $\alpha \geq 1 + \Omega(1)$, there is a quantum query algorithm that estimates $\mathrm{T}_\alpha(\rho_0, \rho_1)$ to within additive error ϵ by using $O(1/\epsilon^{\alpha+1+\frac{1}{\alpha-1}})$ queries to Q_0 and Q_1.*

3.2.2 Sample-efficient quantum algorithm for estimating powered T_α

We proceed by describing efficient quantum sample algorithms for $\Lambda_\alpha(\rho_0, \rho_1)$ and $\mathrm{T}_\alpha(\rho_0, \rho_1)$, as stated in Lemma 15 and in [52, Lemma 3.4] in the full version. Our sample algorithms are obtained by combining the quantum query algorithm from Lemma 13 with the samplizer [74, 75]. The corresponding explanatory framework is presented in in [52, Algorithm 1], and the detailed proof is given in [52, Section 3.2.2].

▶ **Lemma 15** (Powered quantum ℓ_α distance estimation via samples). *For every constantly large $\alpha \geq 1 + \Omega(1)$, $\Lambda_\alpha(\rho_0, \rho_1)$ can be estimated to within additive error ϵ on a quantum computer by using $\widetilde{O}(1/\epsilon^{3+\frac{2}{\alpha-1}})$ samples of ρ_0 and ρ_1.*

By combining Proposition 7 with Lemma 15 for additive error ϵ^α, we obtain a quantum sample algorithm for estimating $\mathrm{T}_\alpha(\rho_0, \rho_1)$ when $\alpha \geq 1 + \Omega(1)$ is constantly large, as also stated in [52, Theorem 3.5] in the full version:

▶ **Theorem 16** (Quantum ℓ_α distance estimation via samples). *For every constantly large $\alpha \geq 1 + \Omega(1)$, there is a quantum sample algorithm that estimates the quantum ℓ_α distance $\mathrm{T}_\alpha(\rho_0, \rho_1)$ to within additive error ϵ by using $\widetilde{O}(1/\epsilon^{3\alpha+2+\frac{2}{\alpha-1}})$ samples of ρ_0 and ρ_1.*

4 Hardness and lower bounds for α constantly above 1

We begin by introducing a generalization of QSD from [77], where the trace distance is replaced with the quantum ℓ_α distance as the closeness measure:

▶ **Definition 17** (Quantum State Distinguishability Problem with Schatten α-norm, QSD_α). *Let Q_0 and Q_1 be quantum circuits acting on m qubits ("input length") and having n specified output qubits ("output length"), where $m(n)$ is a polynomial function of n. Let ρ_i denote the quantum state obtained by running Q_i on state $|0\rangle^{\otimes m}$ and tracing out the non-output qubits. Let $a(n)$ and $b(n)$ be efficiently computable functions. Decide whether*:

- Yes: *A pair of quantum circuits (Q_0, Q_1) such that $\mathrm{T}_\alpha(\rho_0, \rho_1) \geq a(n)$;*
- No: *A pair of quantum circuits (Q_0, Q_1) such that $\mathrm{T}_\alpha(\rho_0, \rho_1) \leq b(n)$.*

Moreover, we denoted the restricted version, where ρ_0 and ρ_1 are pure states, as $\mathrm{PureQSD}_\alpha$.

In the remainder of this section, we establish rank-dependent inequalities between the quantum ℓ_α distance and the trace distance (Theorem 18) in Section 4.1. These inequalities facilitate reductions that demonstrate the BQP hardness (Theorem 21) and derive quantitative lower bounds on queries and samples (Theorem 22) for PureQSD$_\alpha$ in Section 4.2.

4.1 Rank-dependent inequalities between T_α and the trace distance

We generalize the rank-dependent inequalities between the (squared) Hilbert-Schmidt distance and the trace distance, as demonstrated in [24, Appendix G] and [25, Theorem 1] for the case of $\alpha = 2$, to all $1 \le \alpha \le \infty$:

▶ **Theorem 18** (T_α vs. T)**.** *Let ρ_0 and ρ_1 be quantum states. The following holds*:
(1) *For any α in the range $1 \le \alpha < \infty$,*

$$2^{1-\frac{1}{\alpha}} \cdot \mathrm{T}_\alpha(\rho_0,\rho_1) \le \mathrm{T}(\rho_0,\rho_1) \le 2\big(\mathrm{rank}(\rho_0)^{1-\alpha} + \mathrm{rank}(\rho_1)^{1-\alpha}\big)^{-\frac{1}{\alpha}} \cdot \mathrm{T}_\alpha(\rho_0,\rho_1).$$

(2) *For $\alpha = \infty$, $2 \cdot \mathrm{T}_\infty(\rho_0,\rho_1) \le \mathrm{T}(\rho_0,\rho_1) \le 2\min\{\mathrm{rank}(\rho_0),\mathrm{rank}(\rho_1)\} \cdot \mathrm{T}_\infty(\rho_0,\rho_1)$.*

It is worth noting that Item 1 and Item 2 in Theorem 18 are consistent, specifically

$$\lim_{\alpha\to\infty} \big(\mathrm{rank}(\rho_0)^{1-\alpha} + \mathrm{rank}(\rho_1)^{1-\alpha}\big)^{-\frac{1}{\alpha}} = \min\{\mathrm{rank}(\rho_0),\mathrm{rank}(\rho_1)\}.$$

Additionally, the inequalities in Theorem 18 sharpen the inequalities between the trace norm and the Schatten norm (see, e.g., [6, Equation (1.31)]):

$$\forall 1 \le p \le \infty, \quad \|A\|_p \le \|A\|_1 \le r_A^{1-1/p} \cdot \|A\|_p. \tag{2}$$

By considering the maximum rank of ρ_0 and ρ_1, we can derive a simplified form of Theorem 18 for convenience:

▶ **Corollary 19** (T_α vs. T, simplified)**.** *For any quantum states ρ_0 and ρ_1, the following holds*:

$$\forall 1 \le \alpha < \infty, \quad 2^{1-\frac{1}{\alpha}} \cdot \mathrm{T}_\alpha(\rho_0,\rho_1) \le \mathrm{T}(\rho_0,\rho_1) \le (2\max\{\mathrm{rank}(\rho_0),\mathrm{rank}(\rho_1)\})^{1-\frac{1}{\alpha}} \cdot \mathrm{T}_\alpha(\rho_0,\rho_1).$$

Moreover, for pure quantum states, Theorem 18 yields the following equality:

▶ **Corollary 20** ($\mathrm{T}_\alpha = \mathrm{T}$ for pure states)**.** *For any pure states $|\psi_0\rangle\langle\psi_0|$ and $|\psi_1\rangle\langle\psi_1|$, we have*:

$$\forall 1 \le \alpha \le \infty, \quad 2^{1-\frac{1}{\alpha}} \cdot \mathrm{T}_\alpha(|\psi_0\rangle\langle\psi_0|,|\psi_1\rangle\langle\psi_1|) = \mathrm{T}(|\psi_0\rangle\langle\psi_0|,|\psi_1\rangle\langle\psi_1|).$$

The proof of Theorem 18 (see also [52, Theorem 4.2] in the full version) relies on the positive semi-definite matrices ς_0 and ς_1, defined for each $b \in \{0,1\}$ as $\varsigma_b := \frac{1}{2}\big((-1)^b(\rho_0-\rho_1) + |\rho_0-\rho_1|\big)$. These matrices are orthogonal and satisfy the relation $\rho_0 - \rho_1 = \varsigma_0 - \varsigma_1$. The complete proof is provided in [52, Section 4.1].

4.2 Computational hardness and lower bounds

We first state the computational hardness result of PureQSD$_\alpha$ with $1 \le \alpha \le \infty$, obtained by a reduction from PureQSD, which is BQP hard [64], as stated in Theorem 21 and in [52, Theorem 4.5]. The proof can be found in the full version, specifically in [52, Section 4.2].

▶ **Theorem 21** (PureQSD$_\alpha$ is BQP-hard)**.** *For any $1 \le \alpha \le \infty$ and $n \ge 2$, it holds that*:

$$\mathrm{PureQSD}_\alpha\Big[2^{\frac{1}{\alpha}-1} \cdot \big(1-2^{-n}\big),\ 2^{\frac{1}{\alpha}-1-n}\Big] \text{ is BQP-hard.}$$

Next, we provide lower bounds on the query and sample complexities for PureQSD_α by reducing to those for PureQSD in [70], as presented in Theorem 22 and in [52, Theorem 4.6]. The proof is provided in the full version, particularly in [52, Section 4.2].

▶ **Theorem 22** (PureQSD_α). *For any $1 \leq \alpha \leq \infty$ and $0 < \epsilon < 2^{\frac{1}{\alpha}-2}$, there exist n-qubit pure states $|\psi_0\rangle$ and $|\psi_1\rangle$ such that deciding whether $\mathrm{T}_\alpha(|\psi_0\rangle\langle\psi_0|, |\psi_1\rangle\langle\psi_1|)$ is at least ϵ or exactly 0 requires*:
(1) Queries: *In the purified quantum access model, the quantum query complexity is $\Omega(1/\epsilon)$.*

(2) Samples: *The quantum sample complexity is $\Omega(1/\epsilon^2)$.*

5 Quantum ℓ_α distance estimation for $\alpha > 1$ near 1

In this section, we establish that QSD_α is QSZK-complete for $1 \leq \alpha \leq 1 + \frac{1}{n}$ (see also [52, Theorem 5.1] in the full version), extending the prior result that QSD ($\alpha = 1$) is QSZK-complete, as shown in [77]:

▶ **Theorem 23** (QSD_α is QSZK-complete for $\alpha > 1$ near 1). *Let $a(n)$ and $b(n)$ be efficiently computable functions such that $0 \leq b < a \leq 1$. Then, for any $1 \leq \alpha \leq 1 + \frac{1}{n}$, it holds that*:

For any $a(n)^2 - b(n) \geq 1/O(\log n)$, $\text{QSD}_\alpha[a, b]$ is in QSZK.

Moreover, $\text{QSD}_\alpha[a, b]$ is QSZK-hard if $a(n) \leq 1/2 - 2^{-n^\tau - 1}$ and $b(n) \geq 2^{-n^\tau - \frac{1}{n+1}}$ for every constant $\tau \in (0, 1/2)$ and sufficiently large integer n.

The main challenge in proving Theorem 23 is to establish a QSZK containment of QSD_α under the polarizing regime $a(n)^2 - b(n) \geq 1/O(\log n)$.[10] A direct approach, combining the inequalities between T and T_α (Corollary 19) with the QSZK containment of QSD from [77, 78], only yields a QSZK containment of $\text{QSD}_\alpha[a, b]$ under a *weaker* regime, $a(n)^2/2 - b(n) \geq 1/O(\log n)$. To circumvent this, we provide a (partial) polarization lemma for T_α (Lemma 25), which enables us to achieve the desired QSZK containment in Theorem 23.

The remainder of this section establishes the QSZK containment of QSD_α in Section 5.1. We then show the QSZK hardness of QSD_α (Theorem 26) and derive quantitative lower bounds on query complexity (Theorem 27) and sample complexity (Theorem 28) in Section 5.2.

5.1 QSZK containment via a partial polarization lemma for T_α

▶ **Theorem 24** (QSD_α is in QSZK). *Let $a(n)$ and $b(n)$ be efficiently computable functions satisfying $0 \leq b < a \leq 1$. Then, the following holds*:

$$\textit{For any } \alpha \in \left[1, 1 + \frac{1}{n}\right] \textit{ and any } a(n)^2 - b(n) \geq \frac{1}{O(\log n)}, \text{QSD}_\alpha[a, b] \textit{ is in QSZK.}$$

The proof of Theorem 24 (see also [52, Theorem 5.2] in the full version) can be found in [52, Section 5.1]. The main technical tool is a *partial* polarization lemma for T_α (see also [52, Lemma 5.3]), which ensures that any pair (a, b) within the polarizing regime can be made *constantly* separated:

[10] Notably, similar to the classical cases [12], by reducing to the Quantum Jensen-Shannon Divergence Problem (QJSP) or the Measured Quantum Triangular Discrimination Problem (measQTDP) introduced in [51], the QSZK containment of QSD_α holds slightly beyond the polarizing regime.

▶ **Lemma 25** (A partial polarization lemma for T_α). *Let Q_0 and Q_1 be quantum circuits that prepare quantum states ρ_0 and ρ_1, respectively. There is a deterministic procedure that, given an input (Q_0, Q_1, a, b, k) where $a(n)^2 - b(n) \geq 1/O(\log n)$, outputs new quantum circuits $\widetilde{Q}_0$ and $\widetilde{Q}_1$ that prepare the states $\widetilde{\rho}_0$ and $\widetilde{\rho}_1$, respectively. The resulting states satisfy that*:

$$\begin{aligned} \text{For any } \alpha \in [1, 1+1/n], \quad \mathrm{T}_\alpha(\rho_0, \rho_1) \geq a \quad &\Longrightarrow \quad \mathrm{T}_\alpha(\widetilde{\rho}_0, \widetilde{\rho}_1) \geq (1 - e^{-k})/2, \\ \mathrm{T}_\alpha(\rho_0, \rho_1) \leq b \quad &\Longrightarrow \quad \mathrm{T}_\alpha(\widetilde{\rho}_0, \widetilde{\rho}_1) \leq 1/16. \end{aligned}$$

Here, the states $\widetilde{\rho}_0$ and $\widetilde{\rho}_1$ are defined over $\widetilde{O}\Big(nk^{O\left(\frac{b\ln(2/a^2)}{a^2-b}\right)}\Big)$ qubits. Moreover, when $k \leq O(1)$ or $a^2 - b \geq \Omega(1)$, the time complexity of the procedure is $\mathrm{poly}\Big(|Q_0|, |Q_1|, k, \exp\big(\frac{b\ln(1/a^2)}{a^2-b}\big)\Big)$.

In analogy with polarization lemmas for various classical [12, 21, 66] and quantum [51, 77] closeness measures, the proof of Lemma 25 proceeds by separately reducing the errors on both sides of the problem QSD_α. This is achieved using the XOR lemma ([52, Lemma 5.4]) and the direct product lemma ([52, Lemma 5.5]) for T_α. The full statements of these two lemmas, along with the full proofs of all three, are given in the full version, particularly in [52, Section 5.1].

5.2 Computational hardness and lower bounds for $\alpha > 1$ near 1

▶ **Theorem 26** (QSD_α is QSZK-hard). *For any positive constant $\delta > 0$ that can be made arbitrarily small, the following holds for sufficiently large n:*

(1) *For any $1 \leq \alpha \leq 1 + \frac{1}{n^{1+\delta}}$, $\forall \tau \in (0, 1/2)$, $\mathrm{QSD}_\alpha\big[1 - \gamma_{\delta,\tau}(n), \gamma'_{\delta,\tau}(n)\big]$ is QSZK-hard, where $\gamma_{\delta,\tau}(n) \coloneqq 1 - 2^{-\frac{n+1}{n^{1+\delta}+1}} + 2^{-n^\tau - \frac{n+1}{n^{1+\delta}+1}}$ and $\gamma'_{\delta,\tau}(n) \coloneqq 2^{-n^\tau - \frac{1}{n^{1+\delta}+1}}$.*

(2) *For any $1 + \frac{1}{n^{1+\delta}} < \alpha \leq 1 + \frac{1}{n}$, $\forall \tau \in (0, 1/2)$, $\mathrm{QSD}_\alpha\big[\frac{1}{2} - 2^{-n^\tau - 1}, 2^{-n^\tau - \frac{1}{n+1}}\big]$ is QSZK-hard.*

The proof of Theorem 26 (see also [52, Theorem 5.6]), based on reductions to QSD that is QSZK-hard [77, 78], is provided in the full version, particularly in [52, Section 5.2]. For any n-qubit state ρ of rank r, let ρ_{U} be the corresponding n-qubit state whose eigenvalues are uniformly distributed over the support of ρ. Next, we can establish the following lower bounds (see also [52, Theorem 5.7]), and the proof also can be found in [52, Section 5.2]:

▶ **Theorem 27** (Query complexity lower bounds for QSD_α). *The following query complexity lower bounds hold in the purified quantum query access model, depending on the range of α, where $\delta > 0$ is a constant that can be made arbitrarily small*:

(1) *For any $1 + \frac{1}{n^{1+\delta}} < \alpha \leq 1 + \frac{1}{n}$ and $0 < \epsilon \leq 2^{\frac{1}{\alpha} - 2}$, there exist an n-qubit state ρ of rank r and the corresponding state ρ_{U} such that deciding whether $\mathrm{T}_\alpha(\rho, \rho_{\mathsf{U}})$ is at least ϵ or exactly 0 requires $\Omega(r^{1/3})$ queries.*

(2) *For any $1 \leq \alpha \leq 1 + \frac{1}{n^{1+\delta}}$, there exist a constant $\epsilon > 0$ such that, for some n-qubit state ρ of rank r and the corresponding state ρ_{U}, estimating $\mathrm{T}_\alpha(\rho, \rho_{\mathsf{U}})$ to within additive error ϵ requires $\widetilde{\Omega}(r^{1/2})$ queries.*

By leveraging the same reduction used to prove Theorem 27(1), the rank-dependent sample complexity lower bound (given in [52, Lemma 2.10(2)] in the full version) for estimating the trace distance $\mathrm{T}(\cdot, \cdot)$ can be extended to the quantum ℓ_α distance $\mathrm{T}_\alpha(\cdot, \cdot)$ with $1 \leq \alpha \leq 1 + \frac{1}{n}$, as also stated in [52, Theorem 5.8]:

▶ **Theorem 28** (Sample complexity lower bound for QSD_α). *For any $1 \leq \alpha \leq 1 + \frac{1}{n}$ and $0 \leq \epsilon \leq 2^{\frac{1}{\alpha} - 2}$, there exists an n-qubit state ρ of rank r and the corresponding state ρ_{U} such that deciding whether $\mathrm{T}_\alpha(\rho, \rho_{\mathsf{U}})$ is at least ϵ or exactly 0 requires $\Omega\big(r/\epsilon^2\big)$ samples of ρ.*

References

1 Jayadev Acharya, Ibrahim Issa, Nirmal V. Shende, and Aaron B. Wagner. Estimating quantum entropy. *IEEE Journal on Selected Areas in Information Theory*, 1(2):454–468, 2020. Preliminary version in *ISIT 2019*. doi:10.1109/JSAIT.2020.3015235.

2 Leonard M. Adleman, Jonathan Demarrais, and Ming-Deh A. Huang. Quantum computability. *SIAM Journal on Computing*, 26(5):1524–1540, 1997. doi:10.1137/S0097539795293639.

3 Dorit Aharonov, Vaughan Jones, and Zeph Landau. A polynomial quantum algorithm for approximating the Jones polynomial. *Algorithmica*, 55(3):395–421, 2009. Preliminary version in *STOC 2006*. doi:10.1007/s00453-008-9168-0.

4 Anurag Anshu, Zeph Landau, and Yunchao Liu. Distributed quantum inner product estimation. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 44–51, 2022. doi:10.1145/3519935.3519974.

5 Andrew Arrasmith, Lukasz Cincio, Andrew T. Sornborger, Wojciech H. Zurek, and Patrick J. Coles. Variational consistent histories as a hybrid algorithm for quantum foundations. *Nature communications*, 10(1):3438, 2019. doi:10.1038/s41467-019-11417-0.

6 Guillaume Aubrun and Stanisław J. Szarek. *Alice and Bob Meet Banach: The Interface of Asymptotic Geometric Analysis and Quantum Information Theory*, volume 223 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2017. doi:10.1090/surv/223.

7 Costin Bădescu, Ryan O'Donnell, and John Wright. Quantum state certification. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 503–514, 2019. doi:10.1145/3313276.3316344.

8 Bernhard Baumgartner. An inequality for the trace of matrix products, using absolute values. ArXiv e-prints, 2011. arXiv:1106.6189.

9 Aleksandrs Belovs. Quantum algorithms for classical probability distributions. In *Proceedings of the 27th Annual European Symposium on Algorithms, ESA 2019*, volume 144 of *LIPIcs*, pages 16:1–16:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ESA.2019.16.

10 Avraham Ben-Aroya, Oded Schwartz, and Amnon Ta-Shma. Quantum expanders: motivation and construction. *Theory of Computing*, 6(3):47–79, 2010. Preliminary version in *CCC 2008*. doi:10.4086/toc.2010.v006a003.

11 Michael Ben-Or, Michał Horodecki, Debbie W. Leung, Dominic Mayers, and Jonathan Oppenheim. The universal composable security of quantum key distribution. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 386–406. Springer, 2005. doi:10.1007/978-3-540-30576-7_21.

12 Itay Berman, Akshay Degwekar, Ron D. Rothblum, and Prashant Nalini Vasudevan. Statistical difference beyond the polarizing regime. In *Proceedings of the 17th International Conference on Theory of Cryptography Conference*, pages 311–332. Springer, 2019. ECCC:TR19-038. doi:10.1007/978-3-030-36033-7_12.

13 Serge Bernstein. Sur la meilleure approximation de $|x-c|^p$. *Doklady Akademii Nauk SSSR*, 18:379–384, 1938.

14 Serge Bernstein. Sur la meilleure approximation de $|x|^p$ par des polynômes de degrés très élevés. *Izvestiya Akademii Nauk SSSR. Seriya Matematicheskaya*, 2(2):169–190, 1938. URL: https://www.mathnet.ru/eng/im3513.

15 Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. Simulating Hamiltonian dynamics with a truncated Taylor series. *Physical Review Letters*, 114(9):090502, 2015. doi:10.1103/PhysRevLett.114.090502.

16 Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In Samuel J. Lomonaco, Jr. and Howard E. Brandt, editors, *Quantum Computation and Information*, volume 305 of *Contemporary Mathematics*, pages 53–74. AMS, 2002. doi:10.1090/conm/305/05215.

17 Harry Buhrman, Richard Cleve, John Watrous, and Ronald de Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001. doi:10.1103/PhysRevLett.87.167902.

18 Mark Bun, Robin Kothari, and Justin Thaler. The polynomial method strikes back: tight quantum query bounds via dual polynomials. *Theory of Computing*, 16(10):1–71, 2020. Preliminary version in *STOC 2018*. doi:10.4086/toc.2020.v016a010.

19 Chris Cade and Ashley Montanaro. The quantum complexity of computing Schatten p-norms. In *Proceedings of the 13th Conference on the Theory of Quantum Computation, Communication and Cryptography*, volume 111 of *LIPIcs*, pages 4:1–4:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.TQC.2018.4.

20 Clément L. Canonne. A survey on distribution testing: your data is big. but is it blue? In *Theory of Computing Library*, number 9 in Graduate Surveys, pages 1–100. University of Chicago, 2020. ECCC:TR15-063. doi:10.4086/toc.gs.2020.009.

21 André Chailloux, Dragos Florin Ciocan, Iordanis Kerenidis, and Salil Vadhan. Interactive and noninteractive zero knowledge are equivalent in the help model. In *Proceedings of the Fifth Theory of Cryptography Conference*, pages 501–534. Springer, 2008. IACR ePrint:2007/467. doi:10.1007/978-3-540-78524-8_28.

22 Sourav Chakraborty, Eldar Fischer, Arie Matsliah, and Ronald de Wolf. New results on quantum property testing. In *30th Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010*, volume 8 of *LIPIcs*, pages 145–156. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPICS.FSTTCS.2010.145.

23 Andrew M. Childs and Nathan Wiebe. Hamiltonian simulation using linear combinations of unitary operations. *Quantum Information and Computation*, 12(11–12):901–924, 2012. doi:10.26421/QIC12.11-12-1.

24 Patrick J. Coles. Unification of different views of decoherence and discord. *Physical Review A*, 85(4), 2012. doi:10.1103/physreva.85.042103.

25 Patrick J. Coles, M. Cerezo, and Lukasz Cincio. Strong bound between trace distance and Hilbert-Schmidt distance for low-rank states. *Physical Review A*, 100(2):022103, 2019. doi:10.1103/physreva.100.022103.

26 Chandler Davis. All convex invariant functions of Hermitian matrices. *Archiv der Mathematik*, 8(4):276–278, 1957. doi:10.1007/bf01898787.

27 Artur K. Ekert, Carolina Moura Alves, Daniel K. L. Oi, Michał Horodecki, Paweł Horodecki, and Leong Chuan Kwek. Direct estimations of linear and nonlinear functionals of a quantum state. *Physical Review Letters*, 88(21):217901, 2002. doi:10.1103/physrevlett.88.217901.

28 Nic Ezzell, Elliott M. Ball, Aliza U. Siddiqui, Mark M. Wilde, Andrew T. Sornborger, Patrick J. Coles, and Zoë Holmes. Quantum mixed state compiling. *Quantum Science and Technology*, 8(3):035001, 2023. doi:10.1088/2058-9565/acc4e3.

29 Wang Fang and Qisheng Wang. Optimal quantum algorithm for estimating fidelity to a pure state. In *Proceedings of the 33rd Annual European Symposium on Algorithms (ESA)*, 2025. arXiv:2506.23650.

30 Christopher A. Fuchs and Jeroen van de Graaf. Cryptographic distinguishability measures for quantum-mechanical states. *IEEE Transactions on Information Theory*, 45(4):1216–1227, 1999. doi:10.1109/18.761271.

31 Michael I. Ganzburg. *Limit theorems of polynomial approximation with exponential weights*, volume 192 of *Memoirs of the American Mathematical Society*. American Mathematical Society, 2008. doi:10.1090/memo/0897.

32 András Gilyén and Tongyang Li. Distributional property testing in a quantum world. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference*, pages 25:1–25:19, 2020. doi:10.4230/LIPIcs.ITCS.2020.25.

33 András Gilyén and Alexander Poremba. Improved quantum algorithms for fidelity estimation. ArXiv e-prints, 2022. arXiv:2203.15993.

34 András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 193–204, 2019. doi:10.1145/3313276.3316366.

35 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.

36 Oded Goldreich, Amit Sahai, and Salil Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 399–408, 1998. doi:10.1145/276698.276852.

37 Oded Goldreich and Salil Vadhan. Comparing entropies in statistical zero knowledge with applications to the structure of SZK. In *Proceedings of the Fourteenth Annual IEEE Conference on Computational Complexity*, pages 54–73. IEEE, 1999. ECCC:TR98-063. doi:10.1109/CCC.1999.766262.

38 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996. doi:10.1145/237814.237866.

39 Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):150502, 2009. doi:10.1103/PhysRevLett.103.150502.

40 Carl W. Helstrom. Detection theory and quantum mechanics. *Information and Control*, 10(3):254–291, 1967. doi:10.1016/S0019-9958(67)90302-6.

41 Alexander S. Holevo. Statistical decision theory for quantum systems. *Journal of Multivariate Analysis*, 3(4):337–394, 1973. doi:10.1016/0047-259X(73)90028-6.

42 Zdeněk Hradil, Jaroslav Řeháček, Jaromír Fiurášek, and Miroslav Ježek. Maximum-likelihood methods in quantum mechanics. *Quantum State Estimation*, pages 59–112, 2004. doi:10.1007/978-3-540-44481-7_3.

43 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006. Preliminary version in *FOCS 2000*. doi:10.1145/1147954.1147955.

44 Shelby Kimmel, Cedric Yen-Yu Lin, Guang Hao Low, Maris Ozols, and Theodore J. Yoder. Hamiltonian simulation with optimal sample complexity. *npj Quantum Information*, 3(1):1–7, 2017. doi:10.1038/s41534-017-0013-7.

45 A. Yu. Kitaev. Quantum measurements and the Abelian stabilizer problem. ArXiv e-prints, 1995. arXiv:quant-ph/9511026.

46 Marius Kloft, Ulf Brefeld, Sören Sonnenburg, and Alexander Zien. ℓ_p-norm multiple kernel learning. *The Journal of Machine Learning Research*, 12:953–997, 2011. doi:10.5555/1953048.2021033.

47 Hirotada Kobayashi. Non-interactive quantum perfect and statistical zero-knowledge. In *Proceedings of the 14th International Symposium on Algorithms and Computation*, pages 178–188. Springer, 2003. doi:10.1007/978-3-540-24587-2_20.

48 François Le Gall, Yupan Liu, and Qisheng Wang. Space-bounded quantum state testing via space-efficient quantum singular value transformation. ArXiv e-prints, 2023. doi:10.48550/arXiv.2308.05079.

49 James R Lee and Assaf Naor. Embedding the diamond graph in ℓ_p and dimension reduction in ℓ_1. *Geometric & Functional Analysis GAFA*, 14(4):745–747, 2004. doi:10.1007/s00039-004-0473-8.

50 Nana Liu, Qisheng Wang, Mark M. Wilde, and Zhicheng Zhang. Quantum algorithms for matrix geometric means. *npj Quantum Information*, 11:101, 2025. doi:10.1038/s41534-025-00973-7.

51 Yupan Liu. Quantum state testing beyond the polarizing regime and quantum triangular discrimination, 2025. *To appear in* computational complexity. arXiv:2303.01952.

52 Yupan Liu and Qisheng Wang. On estimating the quantum ℓ_α distance. ArXiv e-prints, 2025. The full version of this paper also includes references [8, 9, 21, 22, 30, 44, 47, 50, 54, 55]. arXiv:2505.00457v2.

53 Yupan Liu and Qisheng Wang. On estimating the trace of quantum state powers. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 947–993. SIAM, 2025. doi:10.1137/1.9781611978322.28.

54 Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014. doi:10.1038/nphys3029.

55 Guang Hao Low and Isaac L. Chuang. Hamiltonian simulation by qubitization. *Quantum*, 3:163, 2019. doi:10.22331/q-2019-07-12-163.

56 Alessandro Luongo and Changpeng Shao. Quantum algorithms for spectral sums. ArXiv e-prints, 2020. arXiv:2011.06475.

57 John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. Grand unification of quantum algorithms. *PRX quantum*, 2(4):040203, 2021. doi:10.1103/PRXQuantum.2.040203.

58 Ashley Montanaro and Ronald de Wolf. A survey of quantum property testing. In *Theory of Computing Library*, number 7 in Graduate Surveys, pages 1–81. University of Chicago, 2016. doi:10.4086/toc.gs.2016.007.

59 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010. doi:10.1017/CBO9780511976667.

60 Ryan O'Donnell and John Wright. Quantum spectrum testing. *Communications in Mathematical Physics*, 387(1):1–75, 2021. Preliminary version in *STOC 2015*. doi:10.1007/s00220-021-04180-1.

61 Palash Pandya, Omer Sakarya, and Marcin Wieśniak. Hilbert-Schmidt distance and entanglement witnessing. *Physical Review A*, 102(1):012409, 2020. doi:10.1103/PhysRevA.102.012409.

62 Yihui Quek, Eneet Kaur, and Mark M. Wilde. Multivariate trace estimation in constant quantum depth. *Quantum*, 8:1220, 2024. doi:10.22331/Q-2024-01-10-1220.

63 Renato Renner and Robert König. Universally composable privacy amplification against quantum adversaries. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 407–425. Springer, 2005. doi:10.1007/978-3-540-30576-7_22.

64 Soorya Rethinasamy, Rochisha Agarwal, Kunal Sharma, and Mark M. Wilde. Estimating distinguishability measures on quantum computers. *Physical Review A*, 108(1):012409, 2023. doi:10.1103/PhysRevA.108.012409.

65 Theodore J. Rivlin. *Chebyshev Polynomials: From Approximation Theory to Algebra and Number Theory*. Courier Dover Publications, 1990.

66 Amit Sahai and Salil Vadhan. A complete problem for statistical zero knowledge. *Journal of the ACM*, 50(2):196–249, 2003. Preliminary version in *FOCS 1997*. ECCC:TR00-084. doi:10.1145/636865.636868.

67 Aleksandr F. Timan. *Theory of Approximation of Functions of a Real Variable*, volume 34 of *International Series of Monographs on Pure and Applied Mathematics*. Pergamon Press, 1963. doi:10.1016/c2013-0-05307-8.

68 Vilmos Totik. *Metric properties of harmonic measures*, volume 184 of *Memoirs of the American Mathematical Society*. American Mathematical Society, 2006. doi:10.1090/memo/0867.

69 Bo Waggoner. ℓ_p testing and learning of discrete distributions. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 347–356, 2015. doi:10.1145/2688073.2688095.

70 Qisheng Wang. Optimal trace distance and fidelity estimations for pure quantum states. *IEEE Transactions on Information Theory*, 70(12):8791–8805, 2024. doi:10.1109/TIT.2024.3447915.

71 Qisheng Wang, Ji Guan, Junyi Liu, Zhicheng Zhang, and Mingsheng Ying. New quantum algorithms for computing quantum entropies and distances. *IEEE Transactions on Information Theory*, 70(8):5653–5680, 2024. doi:10.1109/TIT.2024.3399014.

72 Qisheng Wang and Zhicheng Zhang. Fast quantum algorithms for trace distance estimation. *IEEE Transactions on Information Theory*, 70(4):2720–2733, 2024. doi:10.1109/TIT.2023.3321121.

73 Qisheng Wang and Zhicheng Zhang. Sample-optimal quantum estimators for pure-state trace distance and fidelity via samplizer. ArXiv e-prints, 2024. doi:10.48550/arXiv.2410.21201.

74 Qisheng Wang and Zhicheng Zhang. Quantum lower bounds by sample-to-query lifting, 2025. *To appear in* SIAM Journal on Computing. arXiv:2308.01794.

75 Qisheng Wang and Zhicheng Zhang. Time-efficient quantum entropy estimator via samplizer. *IEEE Transactions on Information Theory*, 2025. Preliminary version in *ESA 2024*. doi:10.1109/TIT.2025.3576137.

76 Qisheng Wang, Zhicheng Zhang, Kean Chen, Ji Guan, Wang Fang, Junyi Liu, and Mingsheng Ying. Quantum algorithm for fidelity estimation. *IEEE Transactions on Information Theory*, 69(1):273–282, 2023. doi:10.1109/TIT.2022.3203985.

77 John Watrous. Limits on the power of quantum statistical zero-knowledge. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 459–468. IEEE, 2002. doi:10.1109/SFCS.2002.1181970.

78 John Watrous. Zero-knowledge against quantum attacks. *SIAM Journal on Computing*, 39(1):25–58, 2009. Preliminary version in *STOC 2006*. doi:10.1137/060670997.

79 Mark M. Wilde. *Quantum Information Theory*. Cambridge University Press, 2013. doi:10.1017/9781316809976.

80 Tomoyuki Yamakami and Andrew C. Yao. $\mathsf{NQP}_{\mathbb{C}} = \mathsf{coC}_{=}\mathsf{P}$. *Information Processing Letters*, 71(2):63–69, 1999. doi:10.1016/s0020-0190(99)00084-8.

Length-Constrained Directed Expander Decomposition and Length-Constrained Vertex-Capacitated Flow Shortcuts

Bernhard Haeupler
INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria
ETH Zürich, Switzerland

Yaowei Long
University of Michigan, Ann Arbor, MI, USA

Thatchaphol Saranurak
University of Michigan, Ann Arbor, MI, USA

Shengzhe Wang
ETH Zürich, Switzerland

Abstract

We show the existence of *length-constrained expander decomposition* in directed graphs and undirected vertex-capacitated graphs. Previously, its existence was shown only in undirected edge-capacitated graphs [24, 21]. Along the way, we prove the multi-commodity maxflow-mincut theorems for length-constrained expansion in both directed and undirected vertex-capacitated graphs. Based on our decomposition, we build a *length-constrained flow shortcut* for undirected vertex-capacitated graphs, which roughly speaking is a set of edges and vertices added to the graph so that every multi-commodity flow demand can be routed with approximately the same vertex-congestion *and* length, but all flow paths only contain few edges. This generalizes the shortcut for undirected edge-capacitated graphs from [20]. Length-constrained expander decomposition and flow shortcuts have been crucial in the recent algorithms in undirected edge-capacitated graphs [20, 23]. Our work thus serves as a foundation to generalize these concepts to directed and vertex-capacitated graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Length-Constrained Expander, Expander Decomposition, Shortcut

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.107

Related Version *Full Version*: https://arxiv.org/abs/2503.23217

Funding Part of this work was done while at INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria. This work was partially funded from the Ministry of Education and Science of Bulgaria (support for INSAIT, part of the Bulgarian National Roadmap for Research Infrastructure).
Bernhard Haeupler: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ERC grant agreement 949272).
Thatchaphol Saranurak: Supported by NSF Grant CCF-2238138.
Shengzhe Wang: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC grant agreement No 949272 and No 101171685).

1 Introduction

Expander decomposition found its early applications in property testing [17], clustering [32], and approximation algorithms [10] and, for the last two decades, has been the crucial ingredient in important developments of fast graph algorithms. This includes the first almost-linear time algorithms for spectral sparsifiers and Laplacian solvers [46], approximate


© Bernhard Haeupler, Yaowei Long, Thatchaphol Saranurak, and Shengzhe Wang;
licensed under Creative Commons License CC-BY 4.0



max flow [33, 45, 42], deterministic global min cut [35], exact max flow [12], as well as many almost-optimal dynamic algorithms for minimum spanning trees [41], shortest paths [14, 4], sparsifiers [5], k-edge-connectivity [30], minimum cuts [31, 16], and more [18]. Significant effort [43, 8, 13, 7, 37, 36, 27, 1, 19, 11] has focused on constructing expander decomposition itself. Below, we discuss two successful orthogonal generalizations of expander decomposition.

Vertex and Directed Expander Decomposition. In 2005, Chekuri, Khanna, and Shepherd [10] showed that the construction of expander decomposition in undirected edge-capacitated graphs naturally extends to work in undirected vertex-capacitated graphs and applies them for approximating all-or-nothing vertex-capacitated flow problems. Later, this was extended to directed graphs, an even more general setting [9].[1]

Since 2020, almost-linear time expander decomposition algorithms in these generalized settings have been developed [3, 39, 27, 47] and found impressive applications. For the vertex-capacitated ones, they were crucial for the fastest deterministic vertex connectivity algorithms [44, 40] and data structures for connectivity queries under vertex failures [39, 38, 29]. For the directed ones, they were used for dynamic algorithms in directed graphs [3] and the new combinatorial approaches for exact max flow [15, 2].

Length-Constrained Expander Decomposition and Flow Shortcuts. More recently, Haeupler, Räcke, and Ghaffari [24] introduced *length-constrained expanders* (LC-expanders). At a very high level, these are graphs such that any "reasonable" demand can be routed with low congestion *and* length. In contrast, normal expanders only guarantee low congestion. [24] constructed LC-expander decomposition and applied it to show universally optimal distributed algorithms. In general, LC-expander decomposition is much more effective for problems that simultaneously concern length and congestion.

Based on the new decomposition, [20] introduced the notion of *LC-flow shortcut*[2], a new kind of graph augmentation. Roughly speaking, an LC-flow shortcut is a set of edges and vertices added to the graph so that every multi-commodity flow demand can be routed with approximately the same congestion *and* length, but all flow paths only have a few edges. We use steps, which count the number of edges, to distinguish between edge lengths. This is formalized as follows (see Section 2 for background).

▶ **Definition 1** (Length-Constrained Flow Shortcut). *Given a graph $G = (V, E)$, we say an edge set E' (possibly with endpoints outside V) is a t-step flow shortcut of G with length slack λ and congestion slack κ if*

- *(Forward Mapping) for every demand D routable in G with congestion 1 and length h, D is routable in $G \cup E'$ with congestion 1, length λh, and maximum step t, and*
- *(Backward Mapping) for every demand D on $V(G)$ routable in $G \cup E'$ with congestion 1 and length h, D is routable in G with congestion κ and length h.*

In any undirected edge-capacitated graph, [20] showed, for any $\epsilon > 0$, the existence of a $O(1/\epsilon^2)$-step LC-flow shortcut E' of size $|E'| \leq O(n^{1+O(\epsilon)}\text{polylog}(n))$ with length slack $O(1/\epsilon^3)$ and congestion slack $n^{O(\epsilon)}$.[3] Combined with newly developed close-to-linear time LC-expander decomposition [21, 22], they also obtained a close-to-linear time construction for LC-flow shortcuts albeit with worse quality.

[1] In fact, expander decomposition was only implicit in [10, 9] as their definitions were specific to their applications. The purely graph-theoretic definition was later formalized in [3].

[2] It was called a *low-step flow emulator* in [20].

[3] The shortcut E' in [20] actually has $O(1/\varepsilon^4)$ length slack and $O(1/\varepsilon^2)$ maximum step, but this is only because they tried to ensure that all endpoints of E' are in V. Allowing endpoints outside V, one can replace their *router* with a star and improve the quality to be as we stated.

LC-flow shortcuts have led to significant further progress. This includes the first close-to-linear time constant-approximation algorithm for minimum cost multi-commodity flow [20]. The dynamic but weaker version of flow shortcuts was also the key object in the first deterministic dynamic constant-approximate distance oracle with $O(n^\epsilon)$ update time [23].

However, all applications of LC-expander decomposition until now are limited to undirected edge-capacitated graphs.

1.1 Our Results

To extend the reach of the expander decomposition paradigm further, the history above suggests the following research question:

Can we construct length-constrained expander decomposition and flow shortcuts beyond undirected edge-capacitated graphs?

Indeed, we answer this question affirmatively. In this paper, we focus on the existential results, but the arguments naturally give polynomial-time algorithms. For future work, we are working towards almost-linear-time constructions, which would lead to further applications for minimum cost (multi-commodity) flow in vertex-capacitated and directed graphs. Below, we discuss our contribution in more detail.

Length-Constrained Directed and Vertex Expander Decompositions. We formalize the notions of length-constrained expanders in directed graphs and in undirected vertex-capacitated graphs. Then, we show the existence of length-constrained expander decomposition in directed graphs (Theorem 10) and in undirected vertex-capacitated graphs (Theorem 11). Along the way, we also show that the definition of length-constrained expanders based on cuts is almost equivalent to the characterization based on multi-commodity flow (Theorems 9 and 12). This can be viewed as a version of the approximate multicommodity maxflow mincut theorem [34] but for length-constrained expansion in directed and vertex-capacitated graphs. While this part does not require technical novelty, it is an important foundation for our paper and, we believe, for future work using this concept.

Length-Constrained Vertex-Capacitated Flow Shortcuts. Our main technical contribution (Theorem 14) is to show that, for any undirected *vertex-capacitated* graph and any $\epsilon > 0$, there exists a $2^{O(\frac{1}{\epsilon})}$-step flow shortcut E' of size $|E'| = O(n^{1+O(\epsilon)}\text{polylog}(n))$ with length slack $O(1/\epsilon^3)$ and congestion slack $n^{O(\epsilon)}$. This generalizes the flow shortcut of [20] in undirected edge-capacitated graphs.

Our trade-off between size, length slack, and congestion slack matches the one of [20]. However, our step-bound is $2^{O(\frac{1}{\epsilon})}$ instead of $O(1/\epsilon^2)$. This is due to technical barriers unique to vertex-capacitated graphs, which also requires us to use very different analysis. We leave as a very interesting open problem if it is possible to obtain $\text{poly}(1/\epsilon)$ steps.

We note that obtaining similar LC-flow shortcuts on directed graphs is currently out of reach because it would give the breakthrough on *reachability shortcuts*. Given a graph $G = (V, E)$, an edge set E' is a t-step reachability shortcut of G if, for every pair of vertices $u, v \in V$, u can reach v in G if and only if u can reach v in $G \cup E'$ using at most t steps. Observe that an LC-flow shortcut in a directed graph is strictly stronger than a reachability shortcut. It is a major open problem whether there exists a $n^{o(1)}$-step reachability shortcut of size $n^{1+o(1)}$.[4]

[4] When endpoints of E' must be in V, [26, 28, 6] already showed that there is no $\Omega(n^{1/4})$-step reachability shortcut of size $O(n)$. The lower bounds extend to the shortcut of size $n^{1+\epsilon}$ with a worse step bound.

2 Preliminaries

We give some brief background on length-constrained expansion. A *demand* $D : V \times V \to \mathbb{R}_{\geq 0}$ assigns value to pair of vertices (u, v) and D is *h-length* is it assigns non-zero values only to vertex pairs of distance $\text{dist}_G(u, v) \leq h$ in G. A demand D is *routable* with congestion κ and length λ if there exists a multi-commodity flow routing D with congestion κ and length λ. D *respects* a *node-weighting* $A : V \to \mathbb{R}_{\geq 0}$ if for each vertex u, $\sum_v D(u, v) \leq A(u)$. Let $|A| = \sum_{u \in V} A(u)$. For any $s \geq 1$, A is *(h, s)-length ϕ-expanding in* G if every h-length A-respecting demand is routable with length hs and congestion $O(\frac{\log n}{\phi})$.[5] A length-constrained cut C assigns to each edge an integral length increase, and $G - C$ is the graph G applied with the length increase from cut C. We informally say that G is a *length-constrained expander (LC-expander)* if a node-weighting A whose support is the whole vertex set V is expanding in G. An (h, s)-length ϕ-expander decomposition for A is a length-constrained cut C such that A is (h, s)-length ϕ-expanding in $G - C$.

We use standard graph terminology, deferring details to the full version for further (standard) preliminaries for directed graphs and vertex-capacitated graphs.

2.1 Our Techniques

Next, we give a technical overview of our LC-flow shortcut on vertex-capacitated graphs. We will explain how the strategy used in [20] fails in our setting and how we overcome the obstacle. For simplicity, here we only consider graphs with *unit capacity*. Also, we only construct a slightly weaker notion of LC-flow shortcut in the sense that, it receives an additional length parameter h and the forward mapping only guarantees that every demand routable in G with length $h' \leq h$ and congestion 1 is routable in $G \cup E'$ with length λh, congestion 1 and step t.

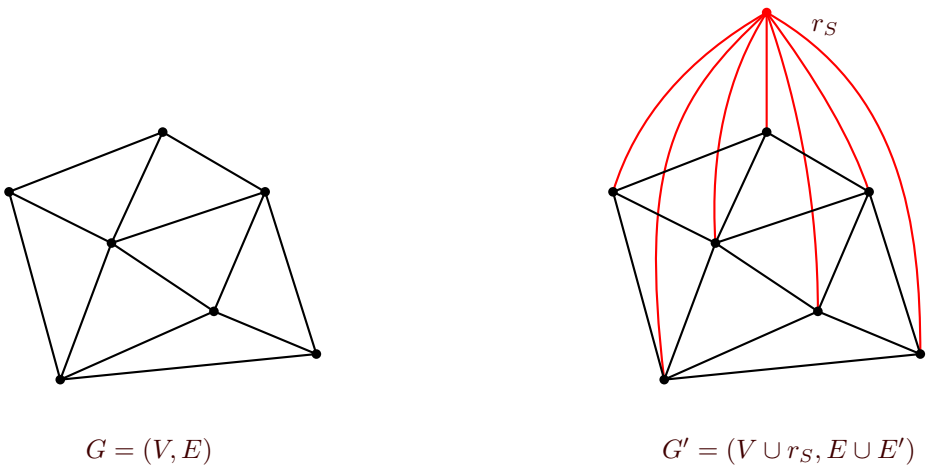


Figure 1 An LC-flow shortcut of a low-diameter LC-expander.

Warm-up: Shortcutting LC-expanders. Before explaining the obstacle, we first show how to shortcut an LC-vertex expander as a warm-up. Suppose that a node-weighting A is (h, s)-length ϕ-vertex expanding in G. Say, $A := \mathbb{1}_V$ (i.e., $A(v) = 1$ for all $v \in V$).

[5] Our definition in the paper (Definition 7) is actually cut-based. This almost-equivalent flow-based definition follows from Theorem 12 and is more convenient in this overview.

Suppose further that G has diameter at most h. In this case, our shortcut is simply a star S connecting each original vertex v to a Steiner vertex r_S with an (hs)-length $A(v)$-capacity edge. We can shortcut any feasible flow in G with 2 steps. An illustrative example is shown in Figure 1. The length slack is $O(s)$ since an h-length original feasible flow is mapped forward to a $(2hs)$-length feasible flow in the star. The congestion slack is $O(\log n/\phi)$ because any feasible flow in the star induces an h-length A-respecting demand. Since A is (h,s)-length ϕ-expanding in G, we route such demand in G with congestion $O(\log n/\phi)$ and without length increasing.

In general, the diameter can be large. Thus, we can construct a *sparse neighborhood cover* to decompose the graph into clusters with diameter h, such that (1) for each vertex v, there is a cluster containing all vertices within distance h/s from v, and (2) each vertex is inside $n^{O(1/s)}$ clusters. Then, we can construct a shortcut by adding an (hs)-edge-length star on each cluster. By a similar argument, we obtain a flow shortcut graph for (h/s)-length original flows with length slack $O(s^2)$, congestion slack $O(n^{O(1/s)}\log n/\phi)$ and step 2.

So far, when we build an LC-flow shortcut for an LC-expander, the vertex-capacitated setting presents no difficulties compared to the edge-capacitated setting, because the above simple approach works in both settings. However, the differences between the two settings arise when generalizing this approach to general graphs via expander hierarchies.

Previous Approach: Shortcutting General Graphs via Boundary-Linkedness. The key idea of [20] is to exploit a hierarchy of *boundary-linked* LC-expander decomposition, defined as follows. Let G be an edge-unit-capacity graph. Initialize the node-weight $A_0 = \deg_G$. For each level $0 \le i \le d$, compute a cut $C_{i+1} \subseteq E$ of size $|C_{i+1}| \approx \phi|A_i|$ such that $A_i + \deg_{C_{i+1}}$ is (h,s)-length ϕ-expanding in $G - C_{i+1}$ where $\deg_{C_{i+1}}(v)$ counts the number of C_{i+1}-edges incident to v.[6] The cut C_{i+1} is called the *boundary-linked* LC-expander decomposition for A_i because it gives a stronger expansion guarantee of $A_i + \deg_{C_{i+1}}$ instead of just A_i. Then, we set $A_{i+1} := \deg_{C_{i+1}}$ and continue to the next level $i+1$. By setting $\phi = 1/(n^{O(1/s)}n^{\varepsilon})$, we have that $d = O(1/\varepsilon)$ and $A_{d+1} = 0$.

From the above construction, we conclude that, for each i, $A_i + A_{i+1}$ is (h,s)-length ϕ-expanding in $G - C_{i+1}$. Therefore, as we have seen in the warm-up, we can add stars on the support of $A_i + A_{i+1}$ so that any flows routing h-length $(A_i + A_{i+1})$-respecting demands in $G - C_{i+1}$ can be shortcut.

Now consider a feasible h-length flow in G. The boundary-linkedness suggests a natural *bottom-up* shortcut scheme. For each flow path P, we can think of routing P's *head packet* and *tail packet* (initially at P's left and right endpoints, denoted by u_0 and v_0) to the same place via shortcuts. Take the head packet as an example. Start with $u_0 \in \mathrm{supp}(A_0) = V$. At each level $0 \le i \le d-1$, let u_{i+1} be the left endpoint of the first $P \cap C_{i+1}$-edge behind u_i. By definition, $u_{i+1} \in \mathrm{supp}(A_{i+1})$ and P's subpath between u_i and u_{i+1} is disjoint from C_{i+1}, so we can use star graphs at level i to route the head packet from u_i to u_{i+1} within 2 steps. In sum, each of the head and tail packets is routed from the bottom up until they reach $u_d, v_d \in \mathrm{supp}(A_d)$, and the top-level star graphs can route them together. The total number of steps is $O(d) = O(1/\epsilon)$[7].

[6] In the actual construction, C_{i+1} assigns fractional values to edges and is called a *moving cut*, defined in Section 3. Here, we assume C_{i+1} is a classic edge cut for simplicity.

[7] We note that the step bound in [20] is $O(1/\epsilon^2)$ because they used powers of expander graphs instead of star graphs to avoid creating vertices outside G, which brought another $O(1/\epsilon)$ factor.

The Obstacle from Vertex Cuts. The overall strategy of the above approach is to shortcut flow paths from a vertex of A_i to an endpoint of edges in C_{i+1}. This was possible since the boundary-linked expander decomposition guarantees that $A_i + \deg_{C_{i+1}}$ is expanding.

In the vertex-capacitated graph, however, the cut $C_{i+1} \subseteq V$ is now a vertex set. To follow the same strategy, we have two natural options. We shortcut flow from a vertex of A_i to either (1) a vertex in C_{i+1}, or (2) a neighbor of C_{i+1}.

In the first case, the strategy requires that $A_i + C_{i+1}$ is expanding in $G - C_{i+1}$. This is trivially impossible because C_{i+1} is not even in the graph $G - C_{i+1}$. In the second case, let $N(C_{i+1})$ denote the neighbors of C_{i+1} that are not in C_{i+1}. The strategy requires $A_i + N(C_{i+1})$ is expanding in $G - C_{i+1}$. However, possibly $N(C_{i+1})$ is very big and has size $|N(C_{i+1})| = \Omega(n|C_{i+1}|)$. It is unlikely that expander decomposition exists to guarantee the expansion of such a large node-weighting. Even if it exists, we would set $A_{i+1} = N(C_{i+1})$ and, hence, we cannot guarantee $|A_{i+1}| \ll |A_i|$. So the number of levels of the hierarchy is unbounded.

In either option, this overall strategy fails in the vertex-capacitated graphs. At a very high level, this is because edges have two endpoints while vertices may have an unbounded number of neighbors.

Our Approach: Top-Down Analysis without Boundary-linkedness. We construct a similar hierarchy of LC-vertex expander decomposition *without* boundary-linkedness as follows. Let $G = (V, E)$ be a vertex-unit-capacity graph. Initialize node-weighting $A_0 = \mathbb{1}_V$. At each level $0 \le i \le d$, computes a cut $C_{i+1} \subseteq V$ such that A_i is (h, s)-length ϕ-vertex-expanding in $G - C_{i+1}$, and set $A_{i+1} := \mathbb{1}_{C_{i+1}}$. In particular, the top level d has $C_{d+1} = \emptyset$. The LC-vertex-expander decomposition guarantees $|C_{i+1}| \approx \phi|A_i|$, so the number d of levels is $O(1/\epsilon)$ by choosing proper ϕ.

Next, we construct the shortcut as follows. For each i, by the expansion of A_i, we can add stars on the support of A_i into our shortcut so that any flows routing h-length A_i-respecting demands in $G - C_{i+1}$ can be shortcut. To analyze the shortcut quality, we will no longer try to route from A_i to A_{i+1} as in the edge-capacitated setting, because we no longer have boundary-linkedness guarantee.

Our analysis is instead *top-down*. At each level i, we shortcut the current flow path as much as possible, and then the prefix and suffix that have not yet been shortcut will be deferred to lower levels as subproblems. To be more concrete, say our initial goal is to shortcut a flow path P in a feasible h-length original flow. At each level $0 \le i \le d$, assume we will receive a subpath P' of P with length at most h in $G - C_{i+1}$ (note that P is a valid input to the top level d because C_{d+1} is empty). We will shortcut P' using star graphs at levels up to i as follows (see Figure 2 for an illustration when $i = 1$).

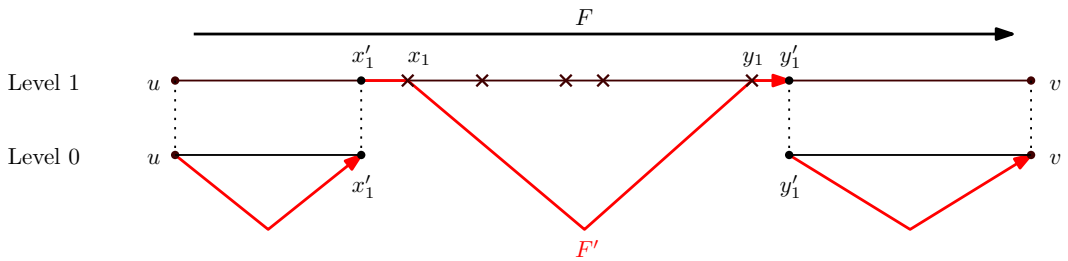


Figure 2 A toy example of forward mapping given we have 2 levels in total. Crossings represent cut vertices in C_1 along the witness path $P_{u,v}$.

Step 1. Let x_i and y_i be the first and last P'-vertices in $\text{supp}(A_i)$ respectively. We can easily shortcut the subpath $P'[x_i, y_i]$ (i.e. the subpath from x_i to y_i) within 2 steps using the star graphs at level i.

Step 2. Let x'_i be the P'-vertex right before x_i and let y'_i be the P'-vertex right after y_i. We regard shortcutting the prefix $P'[u, x'_i]$ and the suffix $P'[y'_i, v]$ as two subproblems at level $i-1$, where u, v are endpoints of P'. Note that both $P'[u, x'_i]$ and $P'[y'_i, v]$ has length at most h in $G - C_i$ because they are disjoint from C_i by definition.

Step 3. After the recursion, we obtain shortcuts for both $P'[u, x'_i]$ and $P'[y'_i, v]$. The shortcut for P' is given by concatenating shortcuts for $P'[u, x'_i], P'[x_i, y_i]$ and $P'[y'_i, v]$ using two original edges (x'_i, x_i) and (y_i, y'_i).

It is not hard to see the final step bound is $2^{O(d)} = 2^{O(1/\epsilon)}$ since the recursion has d levels and each level has two branches. We note that the actual argument is more complicated because the cuts C_i are actually moving cuts which have fractional cut values, and there is no clear partition of P' into 3 parts.

3 Length-Constrained Directed Expansion

In this section, we follow the theory of length-constrained expansion and extend it to the setting of directed graphs. We start with the generalization of notations from length-constrained expanders, which serves as the foundation for subsequent results. Next, we characterize length-constrained expansion in directed graphs with routing, and show the existence of length-constrained directed expander decomposition.

Basic Concepts of Length-Constrained Directed Expansion. The following definition of moving cuts and separation was introduced by Haeupler, Wajc and Zuzic in [25].

▶ **Definition 2** (Length-Constrained Cut). *An h-length moving cut $C : E \mapsto \{0, \frac{1}{h}, \frac{2}{h}, \ldots, 1\}$ assigns to each edge e a fractional cut value between zero and one which is a multiple of $\frac{1}{h}$. The* size *of C is defined as $|C| = \sum_e u(e) \cdot C(e)$. The length increase associated with the h-length moving cut C is denoted with $\ell_{C,h}$ and defined as assigning an edge e the length increase $\ell_{C,h}(e) = h \cdot C(e)$. Any moving cut which only assigns cut values equal to either 0 or 1 is called a pure moving cut. We define the degree of a moving cut over vertex v to be $\deg_C(v) = \sum_{e \ni v} u_G(e) \cdot C(e)$.*

▶ **Definition 3** (h-Length Separated Demand). *For any demand D and any h-length moving cut C, we define the amount of h-length separated demand as the sum of demands between vertices that are h-length separated by C. We denote this quantity with $\text{sep}_h(C, D)$, i.e.,*

$$\text{sep}_h(C, D) = \sum_{u,v:\text{dist}_{G-C}(u,v)>h} D(u, v).$$

▶ **Definition 4** (h-Length Sparsity of a Cut C for Demand D). *For any demand D and any h-length moving cut C with $\text{sep}_h(C, D) > 0$, the h-length sparsity of C with respect to D is the ratio of C's size to how much demand it h-length separates i.e.,*

$$\text{spars}_h(C, D) = \frac{|C|}{\text{sep}_h(C, D)}.$$

Above we generalize the definition of length-constrained moving cut w.r.t arbitrary directed h-length demand. However, for the definition of a directed length-constrained expander, we restrict to symmetric h-length demands.

▶ **Definition 5** ((h,s)-Length Sparsity of a Cut w.r.t. a Node-Weighting). *The (h,s)-length sparsity of any $h \cdot s$-length moving cut C with respect to a node-weighting A is defined as:*

$$\text{spars}_{(h,s)}(C, A) = \min_{A\text{-respecting } h\text{-length symmetric demand } D} \text{spars}_{h\cdot s}(C, D).$$

Intuitively, $(h \cdot s)$-length sparsity of a cut measures how much it $h \cdot s$-length separates h-length demand w.r.t its own size. Furthermore, for a given node-weighting, we associate the sparsest cut w.r.t the node-weighting with its conductance.

▶ **Definition 6** ((h,s)-Length Conductance of a Node-Weighting). *The (h,s)-length conductance of a node-weighting A in a graph G is defined as the (h,s)-length sparsity of the sparsest $h \cdot s$-length moving cut C with respect to A, i.e.,*

$$\text{cond}_{(h,s)}(A) = \min_{h\cdot s\text{-length moving cut } C} \text{spars}_{(h,s)}(C, A).$$

▶ **Definition 7** ((h,s)-Length ϕ-Expanding Node-Weightings). *We say a node-weighting A is (h,s)-length ϕ-expanding if the (h,s)-length conductance of A in G is at least ϕ.*

To see the connection, in the full version, we explain how our notion of length-constrained directed expansion generalizes the non-length-constrained version. Lastly, we give the formal definition of length-constrained directed expander decompositions as follows:

▶ **Definition 8** (Length-Constrained Directed Expander Decomposition). *Given a graph $G = (V, E)$, a directed (h,s)-length ϕ-expander decomposition for a node-weighting A with length slack s and cut slack κ is an $h \cdot s$-length cut C of size at most $\kappa \cdot \phi|A|$ such that A is (h,s)-length ϕ-expanding in $G - C$.*

Routing Characterization of Length-Constrained Directed Expansion. The definition of ϕ-expanding characterizes the sparsity of moving cuts in directed graphs. With the routing characterization, we further show that sparsity is closely related to demand routing.

▶ **Theorem 9** (Routing Characterization of Length-Constrained Directed Expanders). *Given a directed graph G and node-weighting A, for any $h \geq 1$, $\phi < 1$ and $s \geq 1$ we have:*

- *If A is (h,s)-length ϕ-expanding in G, then every A-respecting h-length symmetric demand can be routed in G with congestion at most $O(\frac{\log N}{\phi})$ and length at most $h \cdot s$.*
- *If A is not (h,s)-length ϕ-expanding in G, then some A-respecting h-length symmetric demand cannot be routed with congestion at most $\frac{1}{2\phi}$ and length at most $\frac{h\cdot s}{2}$.*

The proof idea of Theorem 9 is similar to the undirected case as shown in [24], and for completeness, we restate and adapt the proof for the directed setting in the full version.

Existence of Length-Constrained Directed Expander Decompositions. Now, we prove the existence of length-constrained directed expander decompositions. The following theorem formally states the result:

▶ **Theorem 10.** *For any $G = (V, E)$, a node-weighting A, $h > 1$, $\alpha \geq 1$, $\phi < 1$ and a length slack parameter $s = O(\log n)$, there is a directed (h,s)-length ϕ-expander decomposition for A with cut slack $\kappa = O(n^{O(\frac{1}{s})} \log n)$.*

The proof of Theorem 10 closely follows the undirected-case proof in [24], and we append it in the full version for completeness. The basic idea is that, unless the graph is already an (h, s)-length ϕ-expander for node-weighting A, we can repeatedly identify and apply cuts with (h, s)-sparsity less than ϕ. The union of these cuts ultimately renders the graph as an (h, s)-length ϕ-expander. To bound for the total size of all cuts, we sum individual cut sizes. However, since each cut size depends on the sparsity associated with different demands, analysis becomes complex. Thus, we first introduce a special base demand, called exponential demand to relate all other demands in terms of sparsity. Using this, we apply a potential argument to prove the above main theorem.

In the full version, we further discuss *boundary-linked* LC-directed expander decomposition (also called *linked* LC-directed expander decomposition). Expander decompositions with boundary-linkedness have been shown to be very useful in the length-constrained undirected setting and the classic (i.e. non-length-constrained) setting.

4 Length-Constrained Vertex Expansion

In this section we extend the theory of length-constrained expander decomposition to vertex-capacitated graphs. The basic concepts of length-constrained vertex expansion are analogous to those of length-constrained directed expansion in Section 3. The major difference is that now a moving cut C can assign cut values to both vertices and edges. See the full version for preliminaries and formal description of the basic concepts of vertex-capacitated graphs.

The main results of this section is the existence of length-constrained expander decomposition for vertex-capacitated graphs (Theorem 11) and the routing characterization of length-constrained vertex expanders (Theorem 12).

▶ **Theorem 11** (Existential (h, s)-length Expander Decomposition for Vetex-Capacitated Graphs)**.** *For any vertex-capacitated graph $G_{\text{vc}} = (V_{\text{vc}}, E_{\text{vc}})$, node-weighting A_{vc}, $h > 1$, $\phi < 1$ and a length slack parameter $s = O(\log n)$, there is an (h, s)-length ϕ-expander decomposition for A with cut slack $\kappa = O(n^{O(\frac{1}{s})} \log n)$.*

▶ **Theorem 12** (Routing Characterization of Length-Constrained Vertex Expanders)**.** *Given a vertex-capacitated graph G_{vc} and node-weighting A_{vc}, for any $h \geq 1$, $\phi < 1$ and $s \geq 1$ we have:*

- *If A_{vc} is (h, s)-length ϕ-expanding in G_{vc}, then every h-length A_{vc}-respecting demand can be routed in G_{vc} wth congestion at most $O(\frac{\log N}{\phi})$ and dilation at most $h \cdot s$.*
- *If A_{vc} is not (h, s)-length ϕ-expanding in G_{vc}, then some h-length A_{vc}-respecting demand cannot be routed with congestion at most $\frac{1}{6\phi}$ and dilation at most $\frac{h \cdot s}{2}$.*

The proofs of above are in the full version. We introduce a reduction that transforms vertex-capacitated graphs into directed edge-capacitated graphs to show the equivalence, which is crucial for the proofs of Theorem 11 and Theorem 12.

5 Length-Constrained Vertex-Capacitated Flow Shortcuts

In this section, we show the existence of LC-flow shortcuts in vertex-capacitated graphs. The Definition 13 below generalizes Definition 1 of LC-flow shortcuts in the sense that an additional length parameter h is given and the forward mapping only holds for demands routable in G with congestion 1 and length at most h.

Algorithm 1 LC-FlowShortcut(G, ϵ, h).

Initialize $A_0 = u_{V(G)}$, where $u_{V(G)}$ denotes the vertex capacity function of G.
Initialize $s = 1/\epsilon$, $\phi = 1/(n^{\epsilon}\kappa)$ ($\kappa = O(n^{O(1/s)} \log n)$ is the cut slack from Theorem 11).
Initialize $i \leftarrow 0$.
while $|A_i| > 0$ **do**
  for j from 1 to $\lceil \log_2 h \rceil$ **do**
    $h_j \leftarrow 2^j, h_{\mathrm{cov},j} = 4h_j, h_{\mathrm{diam},j} = h_{\mathrm{cov},j} \cdot s$.
    $C_{i+1,j} \leftarrow$ an $(h_{\mathrm{diam},j}, s)$-length ϕ-expander decomposition of A_i in G by Theorem 11.
    $\mathcal{N}_{i,j} \leftarrow$ a neighborhood cover with covering radius $h_{\mathrm{cov},j}$, diameter $h_{\mathrm{diam},j}$ in $G - C_{i+1,j}$ by Theorem 2.1 in the full version.
    $H_{i,j} = \bigcup_{S \in \mathcal{N}_{i,j}} H_S$, where H_S is the $h_j s$-length A_i-capacitated star graph on S.
  end for
  $A_{i+1} = \sum_j \frac{h_{\mathrm{diam},j} \cdot s}{h_j} \cdot \deg_{C_{i+1,j}} = 4s^2 \cdot \sum_j \deg_{C_{i+1,j}}$.
end while
Return $E' = \bigcup_{i,j} E(H_{i,j})$.

▶ **Definition 13** (Length-Constrained Flow Shortcut). *Given a graph $G = (V, E)$, we say an edge set E' (possibly with endpoints outside V) is an t-step h-LC-flow shortcut of G with length slack λ and congestion slack κ if*

- *(Forward Mapping) for every demand D routable in G with congestion 1 and length $h' \le h$, D is routable in $G \cup E'$ with congestion 1, length $\lambda h'$, and maximum step t, and*
- *(Backward Mapping) for every demand D on $V(G)$ routable in $G \cup E'$ with congestion 1 and length h', D is routable in G with congestion κ and length h'.*

We note that in [20] there is an analogous but weaker definition of h-LC-flow shortcut, in which the forward mapping only guarantee that D is routable in $G \cup E'$ with length λh instead of $\lambda h'$ (and the same congestion and step). That is, the length slack in Definition 13 is *competitive* in the sense that it upper bounds the ratio between the lengths of the shortcut flow and the original flow. Hence, by choosing a sufficiently large h, the total length of vertices and edges in G, an h-LC-flow shortcut is automatically an LC-flow shortcut.

▶ **Theorem 14.** *Given a vertex-capacitated graph G with parameters $\epsilon = \Omega(\frac{1}{\log n})$, there exists a t-step LC-flow shortcut E' with length slack $O(1/\epsilon^3)$, congestion slack $O(n^{O(\epsilon)} \log^3 n/\epsilon^2)$, $t = 2^{O(\frac{1}{\epsilon})}$ and size $|E'| \le O(n^{1+O(\epsilon)} \log n/\epsilon^2)$.*

Theorem 14 is the main theorem of this section. In what follows, actually we will focus on constructing h-LC-flow shortcut. Setting $h = (m+n)N$ gives the LC-flow shortcut in Theorem 14.

5.1 The Construction

The construction of the flow shortcut graph G' is given by Algorithm 1. The star graphs in Algorithm 1 are formally defined in Definition 15.

▶ **Definition 15** (Star Graphs). *Given a graph G with a node-weighting A and a length parameter h, the h-length A-capacitated star graph on some $S \subseteq V(G)$, denoted by H_S, has*

$$V(H_S) = (\mathrm{supp}(A) \cap S) \cup \{r_S\} \text{ and } E(H_S) = \{(v, r_S) \mid v \in V(H_S) \setminus \{r_S\}\},$$

where the vertex r_S is a Steiner vertex serving as the center and $V(H_S) \setminus \{r_S\}$ are original vertices. The length and capacity of each original vertex is unchanged, while r_S has length 1 and capacity $\sum_{v \in S} A(v)$. Each edge (v, r_S) has length h and capacity $A(v)$.

In short, Algorithm 1 mainly constructs a length-constrained expander hierarchy

$$\{A_i, C_{i+1,j} \mid 0 \le i \le d, 1 \le j \le \lceil \log_2 h \rceil\}$$

where d is the largest i such that $|A_i| > 0$. We point out that $C_{d+1,j}$ is a zero cut for all j. Then the shortcut graph G' is obtained by adding star graphs on neighborhoods of each LC-expander $G - C_{i+1,j}$.

We remark that we do LC-expander decompositions with different length parameters h_j at one level because we aim at a shortcut graph with length slack significantly smaller than its step bound. Intuitively, if we only use LC-expanders with length parameter around h to shortcut an original h-length flow path P, then inevitably each step will have length around h, which means the length slack cannot go far below the number of steps. Now, providing LC-expanders with different length parameters, when we want to shortcut a subpath of P with length h' far smaller than h, we can choose the appropriate LC-expander to obtain a shortcut with length around h' instead of h. Another benefit is that this automatically gives a competitive length slack (this is why in Definition 13 we define the length slack of h-LC-flow shortcut to be competitive).

We first argue the size bound of E'. Observe that, in Algorithm 1, we have $|C_{i+1,j}| \le \kappa\phi|A_i| \le |A_i|/n^\epsilon$ by Theorem 11. In Algorithm 1, the width of each neighborhood cover $\mathcal{N}_{i,j}$ is $\omega = n^{O(1/s)}s = n^{O(\epsilon)}/\epsilon$ by Theorem 2.1 in the full version. Furthermore, because in Algorithm 1 we have $|A_{i+1}| \le 4s^2 \sum_{j'} |C_{i+1,j'}| \le 4s^2 \log h |A_i|/n^\epsilon = O(\log N/(\epsilon^2 n^\epsilon))|A_i|$, we can upper bound d by $d \le O(\log |A_0|/\log(\epsilon^2 n^\epsilon/\log N)) = O(1/\epsilon)$. Finally, by the algorithm, we have

$$|E'| \le O(d \log h) \cdot \omega \cdot n = O(n^{1+O(\epsilon)} \log n/\epsilon^2).$$

Next we show the quality of the shortcut. Before that, we introduce a helper lemma Lemma 16, which shows the demands that each $H_{i,j}$ can route within small steps. We defer its proof to the full version.

▶ **Lemma 16.** *For each i, j, any demand $\hat{D}$ that is $h_{\mathrm{cov},j}$-length in $G - C_{i+1,j}$, A_i-respecting and $u_{V(G)}$-respecting can be routed in $H_{i,j}$ with length $2h_{\mathrm{diam},j}s + 1$, congestion 1 and step 2.*

Now we show that the shortcut E' constructed by Algorithm 1 has length slack $O(1/\epsilon^3)$, congestion slack $n^{O(\epsilon)}$ and step $2^{O(1/\epsilon)}$. To do this, it suffices to show the quality of the forward mapping and backward mapping, i.e. Lemma 17 and Lemma 18, whose proofs are given in Section 5.2 and the full version. Let $G' = G \cup E'$ be the shortcut graph.

▶ **Lemma 17** (Forward Mapping). *For any feasible h-length flow F in G, there is a feasible flow F' routing $\mathrm{Dem}(F)$ in G' with $\mathrm{leng}(F') \le \mathrm{leng}(F) \cdot O(1/\epsilon^3)$ and $\mathrm{step}(F') \le 2^{O(1/\epsilon)}$.*

▶ **Lemma 18** (Backward Mapping). *For any feasible flow F' in G' such that $V(\mathrm{Dem}(F')) \subseteq V(G)$, there is a flow F routing $\mathrm{Dem}(F')$ in G with $\mathrm{leng}(F) \le \mathrm{leng}(F')$ and $\mathrm{cong}(F) \le n^{O(\epsilon)}$.*

5.2 Forward Mapping: Proof of Lemma 17

We will employ a top-down argument. Start from the top level d. At the beginning of processing a level $i \geq 0$, we are given a feasible flow F_i with the following invariant: each flow path $P \in \text{path}(F_i)$ has $\text{leng}(P, G - C_{i+1,j}) \leq h_{\text{cov},j} = 4h_j$, where j is the minimum index s.t. $h_j \geq \text{leng}(P, G)$ (which means $h_j/2 \leq \text{leng}(P, G) \leq h_j$). Initially at the top level d, we set $F_d = F$. Note that F_d satisfies the invariant above because $C_{d+1,j}$ is a zero cut for any j.

First, at the bottom level $i = 0$, we can easily shortcut every flow path in F_0. For each star graph $H_{0,j}$, we assign it a demand $\hat{D}_{0,j}$ which sums over $\text{Dem}(P)$ for each flow path $P \in \text{path}(F_0)$ such that j is the minimum index with $h_j \geq \text{leng}(P, G)$. Observe that each $\hat{D}_{0,j}$ is an $h_{\text{cov},j}$-length in $G - C_{1,j}$ and A_0-respecting, so it can be routed in $H_{0,j}$ with length $2h_{\text{diam},j}s + 1$, congestion 1 and step 2 by Lemma 16.

From now on we consider levels $i \geq 1$. When processing a level i, for each flow path $P \in \text{path}(F_i)$, we may shortcut some subpaths of P using shortcut edges in H. The subpaths of P that have not been shortcut will be added to F_{i-1}, meaning that they are deferred to lower levels to get shortcut. At the end, the final F_{i-1} should ensure the invariants above, and we proceed to the lower level $i - 1$.

Now we will explain the shortcut at a level $i \geq 1$ in detail. Fix a flow path $P \in \text{path}(F_i)$. Let j be the minimum index such that $h_j \geq \text{leng}(P, G)$. We consider two cases.

Case 1: Defer. When $4s^2 \cdot \sum_{j'} C_{i,j'}(P) \leq 3$, we simply add P to F_{i-1}. We have

$$\text{leng}(P, G - C_{i,j}) = \text{leng}(P, G) + h_{\text{diam},j} \cdot s \cdot C_{i,j}(P) \leq 4h_j \leq h_{\text{cov},j},$$

where the inequality is by $\text{leng}(P, G) \leq h_j$, $h_{\text{diam},j} = 4sh_j$ and $C_{i,j}(P) \leq 3/(4s^2)$. Therefore, in this case, the flow path added to F_{i-1} satisfies the invariant.

Case 2: Shortcut. Now suppose $4s^2 \cdot \sum_{j'} C_{i,j'}(P) > 3$. Let u and v be P's endpoints. We say u is the left side and v the right side. For each $0 \leq k \leq |P|$, we refer to w_k as the P-vertex with k steps away from u. In particular, $w_0 = u$ and $w_{|P|} = v$.

We now define two functions $x : V(P) \to \mathbb{R}$ denoting the *budgets* of P-vertices from the left and the right respectively: for each vertex $w_k \in V(P)$,

$$x_P(w_k) = 4s^2 \sum_{j'} C_{i,j'}(w_{k-1}, w_k) + C_{i,j'}(w_k) + C_{i,j'}(w_k, w_{k+1}).$$

In particular,

$$x_P(w_0) = 4s^2 \sum_{j'} C_{i,j'}(w_0) + C_{i,j'}(w_0, w_1) \text{ and}$$

$$x_P(w_{|P|}) = 4s^2 \sum_{j'} C_{i,j'}(w_{|P|-1}, w_{|P|}) + C_{i,j'}(w_{|P|}).$$

Let k_L be the minimum index such that $\sum_{0 \leq k \leq k_L} x_P(w_k) \geq 1$, and symmetrically let k_R be the maximum index such that $\sum_{k_R \leq k \leq |P|} x_P(w_k) \geq 1$. To avoid clutter, we let $L = \{w_0, ..., w_{k_L}\}$ and $R = \{w_{k_R}, ..., w_{|P|}\}$. The following Claim 19 says that L and R have at most one common vertex.

▷ Claim 19. $k_L \leq k_R$.

Proof. By the definition of k_L and k_R, we have

$$\sum_{0\le k\le k_L-1} x_P(w_k) < 1 \text{ and } \sum_{k_R+1\le k\le |P|} x_P(w_k) < 1.$$

However, $\sum_{0\le k\le |P|} x_P(w_k) \ge 4s^2 \sum_{j'} C_{i,j'}(P) > 3$. This means there exists a vertex w_k with $k_L - 1 < k < k_R + 1$, which implies $k_L \le k_R$ ◁

We assign each vertex $w \in L \cup R$ a *load* $x'_P(w) \le x_P(w)$, satisfying that $\sum_{w\in L} x'_P(w) = 1$ and $\sum_{w\in R} x'_P(w) = 1$. We consider the following three-phase strategy to route $F_i(P)$ flow units from u to v. *Phase 1:* the vertex u sends $F_i(P)$ flow units and each $w \in L$ receives $F_i(P)x'_P(w)$ units. *Phase 2:* Each vertex $w \in L$ sends $F_i(P)x'_P(w)$ units and each vertex $w \in R$ receives $F_i(P)x'_P(w)$ units. *Phase 3:* Each vertex $w \in R$ sends exactly $F_i(P)x'_P(w)$ units, and the vertex v receives $F_i(P)$ units.

The First and Third Phases. Roughly speaking, for the first and third phases, we will add their corresponding original flows into F_{i-1}, meaning that they will be deferred to lower levels to get shortcut.

Regarding the first phase, recall that for each $w_k \in L$, we want to route $F_i(P)x'(w_k)$ units from u to w_k. To do this, we add into F_{i-1} a flow path $P_k = P[w_0, w_{k-1}]$ with value $F_i(P)\cdot x'(w_k)$. That is, we require the lower levels to give a shortcut that routes $F_i(P)\cdot x'(w_k)$ from u to w_{k-1} (the P-vertex one step closer to u than w_k). Then, we route $F_i(P)x'(w_k)$ units from w_{k-1} to w_k using the original edge $(w_{k-1}, w_k) \in E(G)$.

The third phase is handled in a similar way. For each $w_k \in R$, we route $F_i(P)x'(w_k)$ flow units from w_k to w_{k+1} using the original edge (w_{k-1}, w_k), and then we add into F_{i-1} a flow path $P_k = P[w_{k+1}, v]$ with value $F_i(P) \cdot x'(w_k)$.

To proceed to the lower level $i-1$, it remains to show that the flow paths added into F_{i-1} satisfy the invariant. Consider a flow path $P_k = P[u, w_{k-1}]$ added from the first phase (where $w_k \le L$, i.e. $0 \le k \le k_L$). We want to show that $\text{leng}(P_k, G - C_{i,j_k}) \le h_{\text{cov},j_k}$, where j_k is the minimum index such that $h_{j_k} \ge \text{leng}(P_k, G)$. By the definition of the budget function x, we have $\sum_{j'} C_{i,j'}(P_k) \le \frac{1}{4s^2} \sum_{0\le k'\le k-1} x_P(w_{k'}) < 1/(4s^2)$, where the second inequality is by $k \le k_L$ and $\sum_{0\le k'\le k_L-1} x_P(w_{k'}) < 1$ (from the definition of k_L). In particular, $C_{i,j_k}(P_k) \le 1/(4s^2)$. Because C_{i,j_k} is an $(h_{\text{diam},j_k} s)$-length moving cut, we have

$$\text{leng}(P_k, G - C_{i,j_k}) = \text{leng}(P_k, G) + h_{\text{diam},j_k} \cdot s \cdot C_{i,j_k}(P_k) \le 2h_{j_k} \le h_{\text{cov},j_k},$$

as desired, where the first inequality uses $h_{\text{diam},j_k} = 4sh_{j_k}$. By a similar argument, we can show that each flow path added from the third phase also satisfies the invariant, so we will not explain it in detail.

The Second Phase. The second phase is where the shortcut happens. We define an arbitrary (multi-commodity) demand $\hat{D}_{i,P}$ capturing this single-commodity demand. Namely, $\hat{D}_{i,P}$ satisfies that (1) $|\hat{D}_{i,P}| = F_i(P)$; (2) for each $w \in L$, $\hat{D}_{i,P}(w, \cdot) = F_i(P)x'_P(w)$; and (3) for each $w \in R$, $\hat{D}_{i,P}(\cdot, w) = F_i(P)x'_P(w)$. We will *assign* $\hat{D}_{i,P}$ to $H_{i,j}$, meaning that we route $\hat{D}_{i,P}$ using shortcut edges in $H_{i,j}$.

By the above assignment, for each $H_{i,j}$ at level i, its total assigned demand, denoted by $\hat{D}_{i,j}$, sums over $\hat{D}_{i,P}$ of all $P \in \text{path}(F_i)$ s.t. j is the minimum index with $h_j \ge \text{leng}(P, G)$. The following Lemma 20 showing that $\hat{D}_{i,j}$ can be routed with low steps, small length and congestion 1, and we defer the proof to the full version.

▶ **Lemma 20.** *For each $H_{i,j}$ at level i, its total assigned demand $\hat{D}_{i,j}$ can be routed in $H_{i,j}$ with length $2h_{\mathrm{diam},j}s + 1$, congestion 1 and step 2.*

Quality of the Forward Mapping. It remains to show that $\mathrm{Dem}(F)$ can be routed in $G' = G \cup \bigcup_{i,j} H_{i,j}$ with length $\mathrm{leng}(F) \cdot O(1/\epsilon^3)$, congestion 1 and step $2^{O(1/\epsilon)}$. The proof for this can be found in the full version.

References

1 Daniel Agassy, Dani Dorfman, and Haim Kaplan. Expander decomposition with fewer inter-cluster edges using a spectral cut player. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 9:1–9:20, 2023. doi:10.4230/LIPICS.ICALP.2023.9.

2 Aaron Bernstein, Joakim Blikstad, Thatchaphol Saranurak, and Ta-Wei Tu. Maximum flow by augmenting paths in $n^{2+o(1)}$ time, 2024. doi:10.48550/arXiv.2406.03648.

3 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 1123–1134, 2020. doi:10.1109/FOCS46700.2020.00108.

4 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 1000–1008, 2022. doi:10.1109/FOCS52979.2021.00100.

5 Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 20:1–20:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.20.

6 Greg Bodwin and Gary Hoppenworth. Folklore sampling is optimal for exact hopsets: Confirming the $\sqrt{\mathrm{n}}$ barrier. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 701–720. IEEE, 2023. doi:10.1109/FOCS57990.2023.00046.

7 Y. Chang and T. Saranurak. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 377–388, 2020. doi:10.1109/FOCS46700.2020.00043.

8 Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, pages 66–73, 2019. doi:10.1145/3293611.3331618.

9 Chandra Chekuri and Alina Ene. The all-or-nothing flow problem in directed graphs with symmetric demand pairs. *Mathematical Programming*, 154:249–272, 2015. doi:10.1007/S10107-014-0856-Z.

10 Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 183–192, 2005. doi:10.1145/1060590.1060618.

11 Daoyuan Chen, Simon Meierhans, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Parallel and distributed expander decomposition: Simple, fast, and near-optimal. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1705–1719, 2025. doi:10.1137/1.9781611978322.53.

12 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022. doi:10.1109/FOCS54457.2022.00064.

13 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167, 2020. doi:10.1109/FOCS46700.2020.00111.

14 Julia Chuzhoy and Sanjeev Khanna. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 389–400, 2019. doi:10.1145/3313276.3316320.

15 Julia Chuzhoy and Sanjeev Khanna. Maximum bipartite matching in $n^{2+o(1)}$ time via a combinatorial algorithm. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 83–94, 2024. doi:10.1145/3618260.3649725.

16 Antoine El-Hayek, Monika Henzinger, and Jason Li. Fully dynamic approximate minimum cut in subpolynomial time per operation. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 750–784, 2025. doi:10.1137/1.9781611978322.22.

17 Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 289–298, 1998. doi:10.1145/276698.276767.

18 Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2212–2228, 2021. doi:10.1137/1.9781611976465.132.

19 Lars Gottesbüren, Nikos Parotsidis, and Maximilian Probst Gutenberg. Practical expander decomposition. In *Proceedings of the European Symposium on Algorithms (ESA)*, pages 61:1–61:17, 2024. doi:10.4230/LIPICS.ESA.2024.61.

20 Bernhard Haeupler, D. Ellis Hershkowitz, Jason Li, Antti Roeyskoe, and Thatchaphol Saranurak. Low-step multi-commodity flow emulators. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 71–82. ACM, 2024. doi:10.1145/3618260.3649689.

21 Bernhard Haeupler, D Ellis Hershkowitz, and Zihan Tan. New structures and algorithms for length-constrained expander decompositions. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 1634–1645, 2024. doi:10.1109/FOCS61266.2024.00102.

22 Bernhard Haeupler, Jonas Huebotter, and Mohsen Ghaffari. A cut-matching game for constant-hop expanders. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1651–1678, 2025. doi:10.1137/1.9781611978322.51.

23 Bernhard Haeupler, Yaowei Long, and Thatchaphol Saranurak. Dynamic deterministic constant-approximate distance oracles with n^{ϵ} worst-case update time. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 2033–2044, 2024. doi:10.1109/FOCS61266.2024.00121.

24 Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1325–1338. ACM, 2022. doi:10.1145/3519935.3520026.

25 Bernhard Haeupler, David Wajc, and Goran Zuzic. Network coding gaps for completion times of multiple unicasts. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 494–505, 2020. doi:10.1109/FOCS46700.2020.00053.

26 William Hesse. Directed graphs requiring large numbers of shortcuts. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 665–669, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644216.

27 Yiding Hua, Rasmus Kyng, Maximilian Probst Gutenberg, and Zihang Wu. Maintaining expander decompositions via sparse cuts. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 48–69, 2023. doi:10.1137/1.9781611977554.CH2.

28 Shang-En Huang and Seth Pettie. Lower bounds on sparse spanners, emulators, and diameter-reducing shortcuts. *SIAM J. Discret. Math.*, 35(3):2129–2144, 2021. doi:10.1137/19M1306154.

29 Yonggang Jiang, Merav Parter, and Asaf Petruschka. New oracles and labeling schemes for vertex cut queries, 2025. doi:10.48550/arXiv.2501.13596.

30 Wenyu Jin and Xiaorui Sun. Fully dynamic s-t edge connectivity in subpolynomial time (extended abstract). In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 861–872. IEEE, 2021. doi:10.1109/FOCS52979.2021.00088.

31 Wenyu Jin, Xiaorui Sun, and Mikkel Thorup. Fully dynamic min-cut of superconstant size in subpolynomial time. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2999–3026, 2024. doi:10.1137/1.9781611977912.107.

32 Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004. doi:10.1145/990308.990313.

33 Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 217–226, 2014. doi:10.1137/1.9781611973402.16.

34 Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)*, 46(6):787–832, 1999. doi:10.1145/331524.331526.

35 Jason Li. Deterministic mincut in almost-linear time. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 384–395, 2021. doi:10.1145/3406325.3451114.

36 Jason Li, Danupon Nanongkai, Debmalya Panigrahi, and Thatchaphol Saranurak. Near-linear time approximations for cut problems via fair cuts. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 240–275, 2023. doi:10.1137/1.9781611977554.CH10.

37 Jason Li and Thatchaphol Saranurak. Deterministic weighted expander decomposition in almost-linear time, 2021. doi:10.48550/arXiv.2106.01567.

38 Yaowei Long, Seth Pettie, and Thatchaphol Saranurak. Connectivity labeling schemes for edge and vertex faults via expander hierarchies. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–47, 2025. doi:10.1137/1.9781611978322.1.

39 Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 1002–1010, 2022. doi:10.1109/FOCS54457.2022.00098.

40 Chaitanya Nalam, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Deterministic k-vertex connectivity in k^2 max-flows, 2023. doi:10.48550/arXiv.2308.04695.

41 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 950–961, October 2017. doi:10.1109/FOCS.2017.92.

42 Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 227–238, 2014. doi:10.1137/1.9781611973402.17.

43 Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2616–2635, 2019. doi:10.1137/1.9781611975482.162.

44 Thatchaphol Saranurak and Sorrachai Yingchareonthawornchai. Deterministic small vertex connectivity in almost linear time. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 789–800, 2022. doi:10.1109/FOCS54457.2022.00080.

45 Jonah Sherman. Nearly maximum flows in nearly linear time. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 263–269, 2013. doi:10.1109/FOCS.2013.36.

46 Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 81–90, 2004. doi:10.1145/1007352.1007372.

47 Aurelio L. Sulser and Maximilian Probst Gutenberg. A simple and near-optimal algorithm for directed expander decompositions. *CoRR*, abs/2403.04542, 2024. doi:10.48550/arXiv.2403.04542.

Deterministic Approximation Algorithm for Graph Burning

Matej Lieskovský
Faculty of Mathematics and Physics, Computer Science Institute of Charles University, Prague, Czech Republic

Abstract

Graph Burning models a contagion spreading in a network as a process such that in each step one node is infected and also the infection spreads to all neighbors of previously infected nodes. Formally, the burning number $b(G)$ of a given graph $G = (V, E)$, possibly with edge lengths, is the minimum number g such that there exists a sequence of nodes $v_1, \ldots, v_g$ satisfying the property that for each $w \in V$ there exists $i \in \{1, \ldots, g\}$ so that the distance between w and v_i is at most $g - i$.

We present an elegant deterministic 2.314-approximation algorithm for the Graph Burning problem on general graphs with arbitrary edge lengths. This algorithm matches the approximation ratio of the previous randomized 2.314-approximation algorithm and improves on the previous deterministic 3-approximation algorithm.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases Graph Algorithms, Approximation Algorithms, Graph Burning

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.108

Funding *Matej Lieskovský*: Supported by GAUK project 234723, GA ČR project 19-27871X, and the grant SVV–2025–260822.

Acknowledgements We wish to thank our reviewers and colleagues for their insightful feedback.

1 Introduction

Graph Burning was introduced by Bonato et al. [2, 1] as a model of a contagion or information spreading in social networks. It demonstrates how strategically selecting a sequence of nodes to "infect" while letting this infection spread over time may reach the entire network relatively quickly. Formally, we study the following process, where *burned* nodes in a graph represent the infected part of the network.

We are given a graph G and a sequence S of length g where the individual elements $S_1, \ldots, S_g$ of the sequence are either nodes of the graph or "null". At time $t = 0$, no nodes are burned. At time $t = 1$, we burn node S_1. At each time step $t > 1$, all neighbors of already burned nodes also become burned and we also burn node S_t unless S_t is "null". If all nodes are burned at time $t = g$, we call S a *burning sequence* of G of length g. The burning number $b(G)$ of a graph G is the length of its shortest burning sequence.

We can also view a burning sequence as a set of centers where the center (v, r) corresponds to S_{g-r} being the node v, ensuring that any node w with distance $d(w, v) \leq r$ is burned. This generalizes naturally in the presence of arbitrary edge lengths and showcases the relation of Graph Burning to the k-Center problem. We call r the *radius* of center (v, r).

We know that $b(G) \leq |V(G)|$ holds for all graphs. If we prove that some algorithm finds a valid burning sequence of length g in polynomial time whenever $g \geq \lfloor \rho\, b(G) \rfloor$ for some ρ, then we can create a ρ-approximation algorithm for the Graph Burning problem by trying all possible values of g. For the rest of this paper, we talk about these algorithms that take the target value of g as input and take as granted that they imply the corresponding approximation algorithms.


© Matej Lieskovský;
licensed under Creative Commons License CC-BY 4.0



1.1 Our results

We present a simple α-approximation algorithm for the Graph Burning problem on graphs with arbitrary edge lengths where $\alpha = 2e^2/(e^2-1) < 2.314$. We use this value of α throughout the paper.

1.2 Previous results

Bonato and Kamali [3] adapted a well-known greedy 2-approximation algorithm for the k-Center problem into a $(3 - 2/b(G))$-approximation algorithm for Graph Burning. A direct greedy algorithm was given by García-Díaz, Pérez-Sansalvador, and Rodríguez-Hénríquez [5]. We are not aware of any better deterministic algorithms for general graphs.

Lieskovský, Sgall, and Feldmann [11] gave a randomized $(\alpha + \varepsilon)$-approximation algorithm for Graph Burning as well as a pair of lower bounds – 4/3 for graphs with unit-length edges and 2 for graphs with edge lengths. Their algorithm repeatedly selects an arbitrary uncovered node and uses randomization to select a radius, creating a new center. Martingales are used to estimate the number of centers needed and a collision resolution method inspired by linear probing prevents the creation of multiple centers with the same radius.

Approximation algorithms and approximation schemes were developed for various special graph classes by Bonato and Kamali [3], Bonato and Lidbetter [4] and others, ultimately resulting in a PTAS for graphs of small treewidth by Lieskovský and Sgall [10].

2 Definitions and algorithm

In this section we first establish some fundamental concepts and then describe our algorithm. The concepts established in this section will be crucial for the analysis of our algorithm in the following two sections.

2.1 Fundamental concepts

For the analysis of our algorithm, we fix a graph G and some optimal solution OPT, which we will view as set of centers (v, r) with radii from 0 to $b(G) - 1$. In the interest of clarity we will, for the rest of this paper, use "centers" exclusively to refer to centers in this optimal solution while the algorithm will be modifying a "sequence" of nodes of some given length g, until it becomes a valid burning sequence of the graph.

We assign each node $w \in V$ to a unique center $c_{\text{OPT}}(w)$ that is covering it (i.e., $d(w, v) \leq r$), thus creating a partition of the set of nodes of the graph.

For every center (v, r) we denote the set of nodes assigned to it as $V(v, r)$. We say that a center (v, r) is *covered* by a sequence S if all nodes in $V(v, r)$ are burned by the sequence.

A central part of our algorithm is the process for modifying the sequence. We view the sequence S as a queue of length g and use the procedure $\text{push}(v, S)$ to add v to the front of the sequence and remove the last element. Thus if $S = S_1, S_2, S_3, \ldots, S_{g-1}, S_g$, then $\text{push}(v, S) = v, S_1, S_2, S_3, \ldots, S_{g-1}$.

Our algorithm works in *cycles*, selecting one unburned node and pushing it into the sequence in each cycle. Note that cycles are distinct from the time steps in the definition of the Graph Burning problem. We number the cycles starting from 0. We say that a center (v, r) is *serviced* whenever any node in $V(v, r)$ is selected by the algorithm.

Suppose that a $w \in V(v, r)$ is selected in cycle i. At the start of cycle $i + 1$, w is at the start of the burning sequence, ensuring that all nodes within distance $g - 1$ from it are burned by the sequence. With each further cycle, the node is moved to later position in the sequence, reducing the area it ensures the burning of.

▶ **Observation 1.** *If center (v, r) is serviced in cycle i, then it will be covered at the start of cycles $i+1$ to $i+g-2r$.*

2.2 Our algorithm

The algorithm is given a graph G and a guess g for the burning number. It starts with a sequence of length g where all elements are "null", which obviously leaves all nodes unburned. While any such unburned nodes remain, the algorithm selects one of them arbitrarily and pushes it into the sequence. When no unburned nodes remain, the sequence S is a valid burning sequence of G.

Algorithm 1 Deterministic Approximation Algorithm for Graph Burning.

```
1: function BURN(graph G, g ∈ ℕ)
2:     S ← queue containing g "null" elements
3:     while an unburned node exists do
4:         select arbitrary unburned node v
5:         S ← push(v, S)
6:     return S
```

3 Simple results

In this section, we analyse our algorithm at three different levels of detail, resulting in gradually improving approximation ratios. We also show a general observation about the Graph Burning problem in Subsection 3.3. The results from this section will play a minor role in the handling of some special cases by the final analysis of our algorithm in Section 4.

3.1 Proof of 3-approximation

This simple proof matches the previous result of Bonato and Kamali.

▶ **Lemma 2.** *Assuming $g \geq 3b(G)-2$, the algorithm always concludes in at most $b(G)$ cycles.*

Proof of Lemma 2. Given that centers have radius at most $b(G)-1$, any center serviced before cycle $b(G)$ will, by Observation 1, still be covered at the start of cycle $b(G)$. Given that there are at most $b(G)$ centers, the algorithm concludes no later than at the start of cycle $b(G)$ as all centers will be covered. ◀

3.2 Easy improvement

Observe what might happen if we were to run the algorithm with $g = 3b(G)-3$. If the algorithm services the center with radius $b(G)-1$ in cycle 0, then it is possible that this center is no longer covered at the start of cycle $b(G)$, preventing the algorithm from concluding. Fortunately, this can be fixed by running the algorithm for one more cycle as the second largest center, with radius $b(G)-2$, can stop being covered no earlier than at the start of cycle $b(G)+2$

▶ **Lemma 3.** *Assuming $g \geq 3b(G)-3$, the algorithm always concludes in at most $b(G)+1$ cycles.*

Proof of Lemma 3. Let us separate the 1 "large" center with $r = b(G) - 1$ and the $b(G) - 1$ "small" centers with $r \leq b(G) - 2$. Any small serviced center will be covered at the start of cycle $b(G) + 1$.

At the start of cycle $b(G) - 1$, at least $b(G) - 1$ centers have been serviced and are covered, leaving at most one center uncovered. Over the next 2 cycles, only the single large center can become uncovered. Thus the algorithm concludes no later than at the start of cycle $b(G) + 1$ as all centers will be covered. ◀

3.3 General observation

Note that slight improvements such as Lemma 3 are generally easy for Graph Burning, as demonstrated by the following observation:

▶ **Observation 4.** *If, for some $\rho \geq 2$, there exists an algorithm $\mathcal{A}$ that, assuming $g \geq \lfloor \rho\, b(G) \rfloor$, always finds a burning sequence of length g in $O(f(n))$ time, then, for any $k \leq b(G)$, there exists an algorithm $\mathcal{B}$ that always finds a burning sequence of length $g - k$ in $O(n^k f(n))$ time.*

Proof of Observation 4. Algorithm $\mathcal{B}$ is given a graph G and a guessed burning number g. For each of the $O(n^k)$ possible length-k prefixes of a burning sequence of length $g - k$, it uses algorithm $\mathcal{A}$ to look for a burning sequence of length $g - 2k$ for the uncovered part of the graph.

One of those prefixes is $P = v_1, v_2, v_3, \ldots, v_k$ such that all $(v_i, b(G) - i)$ are centers of OPT. Since $g - k \geq b(G)$, these k centers are covered by any burning sequence of length $g - k$ that starts with prefix P. The uncovered part of the graph thus has a burning number at most $b(G) - k$ and algorithm $\mathcal{A}$ will be able to find a burning sequence of length $g - 2k$. ◀

3.4 More significant improvement

In order to motivate further exploration of our algorithm, let us now show that our algorithm is indeed capable of finding burning sequences of length $\rho\, b(G)$ where $\rho < 3$. In particular, we demonstrate that $g \geq \frac{8}{3} b(G)$ is sufficient and the algorithm concludes in $\left\lceil \frac{4}{3} b(G) \right\rceil$ cycles.

We use Observation 1 to show how long each center stays covered after being serviced. At the start of each cycle of our algorithm, each center of OPT is in one of the following three states.

1. Untouched - this center has not been serviced yet and might thus be uncovered.
2. Covered - this center has been serviced in the last $g - 2r$ cycles and thus is covered.
3. Expired - this center has last been serviced more than $g - 2r$ cycles ago and thus might not be covered any more.

For the purposes of our analysis, we will divide the cycles of the algorithm into two phases, with each phase shorter than $b(G)$ cycles. During the first phase, the algorithm is only servicing untouched centers. When sufficiently many cycles have passed for a center to possibly become expired, the algorithm enters the second phase. During the second phase, the algorithm services the remaining untouched centers and any expired centers. The second phase concludes before any center can expire for the second time and we show that all centers must be covered.

We divide the centers into a smaller number of large centers and a larger number of small centers. Small centers are the centers with $r < \frac{2}{3} b(G)$. When serviced, these remain covered for $> \frac{4}{3} b(G)$ cycles, which is at least the number of cycles used by the algorithm. Thus they do not become expired. Large centers are the centers with $r \geq \frac{2}{3} b(G)$. When serviced, these remain covered until the end of the phase, but might expire during the second phase if covered during the first phase.

After carefully setting the length of the phases, we show that sufficiently many small centers must be serviced during the first phase so that any untouched small centers and all large centers can be serviced during the second phase. This ensured that all centers are covered even if all the large centers have expired.

▶ **Lemma 5.** *Assuming $g \geq \frac{8}{3}b(G)$, the algorithm always concludes in at most $\lceil \frac{4}{3}b(G) \rceil$ cycles.*

Proof of Lemma 5. A center (v, r) is called large if $r \geq \frac{2}{3}b(G)$. Note that there are at most $\lfloor \frac{1}{3}b(G) \rfloor$ large centers. The remaining centers are small; they have radius $r < \frac{2}{3}b(G)$ and there are at most $\lceil \frac{2}{3}b(G) \rceil$ small centers. The first phase consists of $\lceil \frac{2}{3}b(G) \rceil$ cycles and the second phase consists of $\lfloor \frac{2}{3}b(G) \rfloor$ cycles.

All centers have $r \leq b(G) - 1$. By Observation 1, any center serviced during the first phase is still covered at the start of cycle $\lceil \frac{2}{3}b(G) \rceil$. Thus, at the end of the first phase, $\lceil \frac{2}{3}b(G) \rceil$ centers have been serviced and are covered, leaving at most $\lfloor \frac{1}{3}b(G) \rfloor$ uncovered centers.

Small centers have $r < \frac{2}{3}b(G)$ and thus any serviced small center will still be covered at the start of cycle $\lceil \frac{4}{3}b(G) \rceil$. Thus, during the second phase, only the large centers can become uncovered and there are at most $\lfloor \frac{1}{3}b(G) \rfloor$ large centers. In total, no more than $\lfloor \frac{2}{3}b(G) \rfloor$ centers need to be serviced during the second phase, and the algorithm concludes no later than at the end of the second phase. ◀

4 2.314-approximation ratio

The entirety of this section is devoted to the proof of the following theorem, which is the main result of this paper.

▶ **Theorem 6.** *Assuming $g \geq \lfloor \alpha b(G) \rfloor$, the algorithm always concludes in at most $O((b(G))^2)$ cycles.*

First, we focus on a single center (v, r). In an approach inspired by problems such as Bamboo Garden [7][9] and Pinwheel Scheduling [8][6], we imagine a timer that starts at 1, is reset to zero whenever the algorithm services the center, and otherwise increases with every cycle so that the timer is ≥ 1 whenever the center might be uncovered.

Second, we look at all the centers of OPT and consider the sum of their timers. This sum starts at $b(G)$ and, if g is large enough, decreases with every cycle. Given that all centers must be covered if the sum is less than 1, we proceed to show that $g \geq \lfloor \alpha b(G) \rfloor$ is indeed large enough, completing the proof.

Proof of Theorem 6. Let us define a potential $\Phi_i(v, r)$ for center (v, r) at the start of the i-th cycle as follows:

$$\Phi_0(v, r) = 1 \tag{1}$$

$$\Phi_{i+1}(v, r) = \begin{cases} 0 & \text{if center } (v, r) \text{ was serviced during the } i\text{-th cycle} \\ \Phi_i(v, r) + \dfrac{1}{g - 2r} & \text{otherwise} \end{cases} \tag{2}$$

Observe that, if $\Phi_i(v, r) < 1$, then center (v, r) is covered at the start of the i-th cycle.

We now define the overall potential at the start of the i-th cycle as the sum of potentials for all centers of OPT and observe that, whenever $\Phi_i < 1$, all centers must be covered.

$$\Phi_i = \sum_{(v,r) \in \text{OPT}} \Phi_i(v, r)$$

We claim that if center (v, r) is serviced in cycle i, then

$$\Phi_{i+1} = \Phi_i + \Delta - \left(1 + \frac{1}{g - 2r}\right)$$

where

$$\Delta = \sum_{r=0}^{b(G)-1} \frac{1}{g - 2r}.$$

The positive term Δ accounts for the increase of the potential of all centers. The negative term $(1 + 1/(g - 2r))$ accounts for the change of the potential of the serviced center (v, r). This is because $\Phi_i(v, r) \geq 1$ must be true for (v, r) to be serviced in cycle i and $\Phi_{i+1}(v, r) = 0$ if (v, r) was serviced in cycle i, which means that the potential is not only decreased by 1, but also does not increase by $1/(g - 2r)$ already included in Δ.

We will show that $\Delta \leq 1$ for $g \geq \lfloor \alpha b(G) \rfloor$. To see that this implies the theorem, first observe that $1 + 1/(g - 2r) \geq 1 + 1/g$ and thus we will get $\Phi_{i+1} \leq \Phi_i - 1/g$. Since $\Phi_0 = b(G)$, algorithm concludes in at most $g \cdot b(G)$ cycles. For $g \leq \frac{8}{3} b(G)$ the theorem follows directly, while for $g > \frac{8}{3} b(G)$ we use Lemma 5 to get the conclusion of the theorem.

It remains to prove that $\Delta \leq 1$. The value of α was set to satisfy the following integral equation:

$$\int_0^{b(G)} \frac{1}{\alpha b(G) - 2x}\, dx = \left[-\frac{\ln(\alpha b(G) - 2x)}{2}\right]_{x=0}^{b(G)} = \frac{\ln(\alpha) - \ln(\alpha - 2)}{2} = 1.$$

We express Δ as a similar integral:

$$\Delta = \sum_{r=0}^{b(G)-1} \frac{1}{g - 2r} = \int_0^{b(G)} \frac{1}{g - 2\lfloor x \rfloor}\, dx.$$

To complete the proof, it is now sufficient to show that the following holds:

$$\int_0^{b(G)} \frac{1}{g - 2\lfloor x \rfloor}\, dx \leq \int_0^{b(G)} \frac{1}{\alpha b(G) - 2x}\, dx.$$

We first observe that this holds whenever $g \geq \alpha b(G)$, using a straightforward substitution for g and a simple upper bound on $\lfloor x \rfloor$. This makes it possible for us to get an α-approximation algorithm by using Observation 4.

Finally, we give a full proof of $\Delta \leq 1$ for all $g \geq \lfloor \alpha b(G) \rfloor$. We separate the integration into intervals where $2\lfloor x \rfloor$ is constant, seeking to prove for all natural $a < b(G)$:

$$\int_a^{a+1} \frac{1}{g - 2\lfloor x \rfloor}\, dx \leq \int_a^{a+1} \frac{1}{\alpha b(G) - 2x}\, dx.$$

The following trivially holds:

$$\int_a^{a+1} \frac{1}{g - 2\lfloor x \rfloor}\, dx = \int_a^{a+1} \frac{1}{g - 2a}\, dx = \frac{1}{g - 2a}.$$

Consider the function $f(z) = 1/(g + 1 - 2z)$. It is convex on $(-\infty, (g + 1)/2)$ and thus for every $z < b(G) \leq g/2$ we have $f(z) \leq \int_{z-1/2}^{z+1/2} f(x)\, dx$. Using this for $z = a + 1/2$ we get the following:

$$\frac{1}{g - 2a} = f(a + 1/2) \leq \int_a^{a+1} f(x)\, dx = \int_a^{a+1} \frac{1}{g + 1 - 2x}\, dx \leq \int_a^{a+1} \frac{1}{\alpha b(G) - 2x}\, dx$$

where the last inequality follows from $g + 1 \geq \lfloor \alpha b(G) \rfloor + 1 \geq \alpha b(G)$. This concludes the proof of $\Delta \leq 1$ and thus the proof of Theorem 6. ◀

5 Conclusion

We have shown a simple deterministic algorithm for Graph Burning that achieves good approximation ratios when thoroughly analysed. Despite this, a significant gap remains between our result and the lower bounds of [11].

References

1 Anthony Bonato, Jeannette Janssen, and Elham Roshanbin and. How to burn a graph. *Internet Mathematics*, 12(1-2):85–100, 2016. doi:10.1080/15427951.2015.1103339.

2 Anthony Bonato, Jeannette Janssen, and Elham Roshanbin. Burning a graph as a model of social contagion. In Anthony Bonato, Fan Chung Graham, and Paweł Prałat, editors, *Algorithms and Models for the Web Graph*, volume 8882, pages 13–22, Cham, December 2014. Springer International Publishing. doi:10.1007/978-3-319-13123-8_2.

3 Anthony Bonato and Shahin Kamali. Approximation algorithms for graph burning. In T.V. Gopal and Junzo Watada, editors, *Theory and Applications of Models of Computation*, pages 74–92, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-14812-6_6.

4 Anthony Bonato and Thomas Lidbetter. Bounds on the burning numbers of spiders and path-forests. *Theoretical Computer Science*, 794:12–19, 2019. Special Issue on Theory and Applications of Graph Searching. doi:10.1016/j.tcs.2018.05.035.

5 Jesús García-Díaz, Julio César Pérez-Sansalvador, Lil María Xibai Rodríguez-Henríquez, and José Alejandro Cornejo-Acosta. Burning graphs through farthest-first traversal. *IEEE Access*, 10:30395–30404, 2022. doi:10.1109/ACCESS.2022.3159695.

6 Leszek Gąsieniec, Benjamin Smith, and Sebastian Wild. *Towards the 5/6-Density Conjecture of Pinwheel Scheduling*, pages 91–103. Society for Industrial and Applied Mathematics, 2022. doi:10.1137/1.9781611977042.8.

7 Felix Höhne and Rob van Stee. A 10/7-Approximation for Discrete Bamboo Garden Trimming and Continuous Trimming on Star Graphs. In Nicole Megow and Adam Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, volume 275 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:19, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.16.

8 Akitoshi Kawamura. Proof of the density threshold conjecture for pinwheel scheduling. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, pages 1816–1819, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3618260.3649757.

9 John Kuszmaul. Bamboo trimming revisited: Simple algorithms can do well too. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '22, pages 411–417, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3490148.3538580.

10 Matej Lieskovský and Jiří Sgall. Graph burning and non-uniform k-centers for small treewidth. In Parinya Chalermsook and Bundit Laekhanukit, editors, *Approximation and Online Algorithms*, pages 20–35, Cham, 2022. Springer International Publishing.

11 Matej Lieskovský, Jiří Sgall, and Andreas Emil Feldmann. Approximation Algorithms and Lower Bounds for Graph Burning. In Nicole Megow and Adam Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, volume 275 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:17, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.9.

Combined Search and Encoding for Seeds, with an Application to Minimal Perfect Hashing

Hans-Peter Lehmann
Karlsruhe Institute of Technology, Germany

Peter Sanders
Karlsruhe Institute of Technology, Germany

Stefan Walzer
Karlsruhe Institute of Technology, Germany

Jonatan Ziegler
Karlsruhe Institute of Technology, Germany

Abstract

Randomised algorithms often employ methods that can fail and that are retried with independent randomness until they succeed. Randomised data structures therefore often store indices of successful attempts, called seeds. If n such seeds are required (e.g., for independent substructures) the standard approach is to compute for each $i \in [n]$ the smallest successful seed S_i and store $\vec{S} = (S_1, \ldots, S_n)$.

The central observation of this paper is that this is not space-optimal. We present a different algorithm that computes a sequence $\vec{S'} = (S'_1, \ldots, S'_n)$ of successful seeds such that the entropy of $\vec{S'}$ undercuts the entropy of $\vec{S}$ by $\Omega(n)$ bits in most cases. To achieve a memory consumption of $\mathrm{OPT} + \varepsilon n$, the expected number of inspected seeds increases by a factor of $\mathcal{O}(1/\varepsilon)$.

We demonstrate the usefulness of our findings with a novel construction for minimal perfect hash functions that, for n keys and any $\varepsilon \in [n^{-3/7}, 1]$, has space requirement $(1+\varepsilon)\mathrm{OPT}$ and construction time $\mathcal{O}(n/\varepsilon)$. All previous approaches only support $\varepsilon = \omega(1/\log n)$ or have construction times that increase exponentially with $1/\varepsilon$. Our implementation beats the construction throughput of the state of the art by more than two orders of magnitude for $\varepsilon \leq 3\%$.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Data compression; Theory of computation → Data structures design and analysis

Keywords and phrases Random Seed, Encoding, Bernoulli Process, Backtracking, Perfect Hashing

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.109

Related Version *Extended Version*: `https://arxiv.org/abs/2502.05613` [24]

Supplementary Material *Software (Source code)*: `https://github.com/ByteHamster/ConsensusRecSplit` [23], archived at `swh:1:dir:eeef738e12bea287eae54a77ce81241587a91767`
Software (Comparison with competitors): `https://github.com/ByteHamster/MPHF-Experiments` [17], archived at `swh:1:dir:7b02eda8ffa5a9d61a086ff2fc34fb0fbdc3eff4`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500). This work was also supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF).

Acknowledgements This paper is based on the bachelor's thesis of Jonatan Ziegler [34].

© Hans-Peter Lehmann, Peter Sanders, Stefan Walzer, and Jonatan Ziegler;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The construction of randomised data structures can sometimes fail, so the data structures need to store indices of successful attempts called *seeds*.[1] In this paper, we present a technique that finds and encodes successful seeds in a space-efficient and time-efficient way.

When searching a single seed, we may imagine an infinite sequence of i.i.d. Bernoulli random variables, called a *Bernoulli process*, that indicate which natural numbers lead to success and that can be inspected one by one. To model the task of finding a *sequence* of seeds we use a *sequence* of Bernoulli processes as follows. In this paper $[n] \coloneqq \{1, \dots, n\}$ and $\mathbb{N}_0 \coloneqq \mathbb{N} \cup \{0\}$.

▶ **Definition 1** (Seed Search and Encoding Problem (SSEP)). *Let $n \in \mathbb{N}$ and $p_1, \dots, p_n \in (0, 1]$. Given for each $i \in [n]$ a Bernoulli process $\vec{B}^{(i)} = (B^{(i)}_j)_{j \in \mathbb{N}_0}$ with parameter p_i, the task is to craft a bitstring M that encodes for each $i \in [n]$ a successful seed $S_i \in \mathbb{N}_0$, i.e. $B^{(i)}_{S_i} = 1$.*

Our primary goals are to minimise the length of M in bits, and to minimise for each $i \in [n]$ the number T_i inspected Bernoulli random variables from $\vec{B}^{(i)}$. Moreover, the time to decode S_i given $p_1, \dots, p_n$, M and $i \in [n]$ should be constant. The optimal values for the expectation of $|M|$ and T_i are the following (for a proof see the full version of this paper [24]).

$$M^{\mathrm{OPT}} = \sum_{i=1}^{n} \log_2(1/p_i) \text{ and } T_i^{\mathrm{OPT}} = 1/p_i \text{ for } i \in [n]. \tag{1}$$

1.1 Ad-Hoc Solutions

We briefly consider two natural strategies for solving the SSEP to build intuition. The obvious strategy (MIN) is to encode the smallest successful seeds, i.e. define

$$S_i \coloneqq \min\{j \in \mathbb{N}_0 \mid B^{(i)}_j = 1\} \text{ and } \vec{S}_{\mathrm{MIN}} \coloneqq (S_1, \dots, S_n) \tag{MIN}$$

and let M be an optimal encoding of $\vec{S}_{\mathrm{MIN}}$. Perhaps surprisingly, this requires $\Omega(n)$ bits more than the optimum if $p_1, \dots, p_n$ are bounded away from 1.

▶ **Lemma 2** (MIN is fast but not space-efficient, proof in full version [24]). *If M encodes $\vec{S}_{\mathrm{MIN}}$ then $\mathbb{E}[T_i] = T_i^{\mathrm{OPT}}$ for all $i \in [n]$ but $\mathbb{E}[|M|] \geq M^{\mathrm{OPT}} + \sum_{i=1}^{n} 1 - p_i$.*

Another simple strategy (UNI) is to encode the smallest seed that is simultaneously successful for all Bernoulli processes, i.e. we define

$$S_* = \min\{j \in \mathbb{N}_0 \mid \forall i \in [n] : B^{(i)}_j = 1\} \text{ and } S_i = S_* \text{ for all } i \in [n]. \tag{UNI}$$

This yields a bitstring M of minimal length but requires total time $\sum_{i=1}^{n} T_i = \exp(\Omega(n))$ if $p_1, \dots, p_n$ are bounded away from 1.

▶ **Lemma 3** (UNI is space-efficient but slow, proof in full version [24]). *If M encodes S_* then $\mathbb{E}[|M|] = M^{\mathrm{OPT}} + \mathcal{O}(1)$ but $\mathbb{E}[\sum_{i=1}^{n} T_i] \geq \prod_{i=1}^{n} 1/p_i \gg \sum_{i=1}^{n} T_i^{\mathrm{OPT}}$.*

[1] Throughout this paper, we use the *Simple Uniform Hashing Assumption*, as do many works before us [5, 29, 28, 22, 21, 14, 7]. This is equivalent to saying that we assume access to an infinite sequence of uniformly random and independent hash functions, each identified by its index (seed). The assumption is an adequate model for practical hash functions and can be justified in theory using the "split-and-share" trick [6].

1.2 The Consensus Algorithm

We present an algorithm for the SSEP designed to simultaneously get close to both optima. We call it *Combined Search and Encoding for Successful Seeds*, or Consensus for short. We summarise the guarantees of the stronger of two variants here.

▶ **Theorem 4** (Main result, informal version of Theorem 7). *For any $\varepsilon > 0$ the full Consensus algorithm with high probability solves the SSEP with $|M| \leq M^{\mathrm{OPT}} + \varepsilon n + \mathcal{O}(\log n)$ and $\mathbb{E}[T_i] = \mathcal{O}(T_i^{\mathrm{OPT}}/\varepsilon)$ for all $i \in [n]$. Each seed S_i is a bitstring of fixed length found at a predetermined offset in M.*

Note that if the geometric mean of $p_1, \ldots, p_n$ is at most $\frac{1}{2}$ then $M^{\mathrm{OPT}} \geq n$ and the leading term εn in our space overhead is at most $\varepsilon M^{\mathrm{OPT}}$.[2]

Intuition. First reconsider the MIN-strategy. It involves the sequence $\vec{S}_{\mathrm{MIN}} = (S_1, \ldots, S_n)$ with some entropy $H(\vec{S}_{\mathrm{MIN}})$. There are two issues with storing $\vec{S}_{\mathrm{MIN}}$. The superficial issue is that each number in the sequence follows a geometric distribution, and it is not obvious how to encode $\vec{S}_{\mathrm{MIN}}$ in space close to $H(\vec{S}_{\mathrm{MIN}})$ while preserving fast random access to each component.[3] The deeper issue is that $\vec{S}_{\mathrm{MIN}}$ is not what a space-optimal approach should encode in the first place. This is demonstrated by the fact that even an optimal encoding of $\vec{S}_{\mathrm{MIN}}$ needs space $H(\vec{S}_{\mathrm{MIN}})$, which UNI undercuts by $\Omega(n)$ bits.

Consensus deals with both issues simultaneously. The idea is to store for each $i \in [n]$ a natural number $1 \leq \sigma_i \leq 2^{\varepsilon}/p_i$. This allows for fixed width encoding of σ_i using $\varepsilon + \log_2 1/p_i$ bits. If we simply used $S_i = \sigma_i$ then the expected number of successful choices for σ_i would be $2^{\varepsilon} > 1$. The problem would be, of course, that the actual number of successful choices might be zero for many $i \in [n]$. So instead we interpret σ_i as a *seed fragment* and define S_i to be a concatenation of seed fragments up to and including σ_i. This way, if we run out of choices for σ_i, we can hope that a different successful choice for earlier seed fragments will allow for a successful choice for σ_i as well. We can construct the sequence $(\sigma_1, \ldots, \sigma_n)$ using backtracking. We explain our algorithm in detail in Section 2. The main technical challenge is to bound the additional cost this brings.

1.3 Motivation: Seed Sequences in Randomised Data Structures

Randomised data structures frequently use seeds to initialise pseudo-random number generators or as additional inputs to hash functions. Since a seed normally counts how often a construction has been restarted, a seed $S \in \mathbb{N}_0$ implies that work $\Omega(S+1)$ has been carried out, meaning any reasonably fast algorithm yields seeds that are only a few bits to a few bytes in size. This does not, however, imply that seeds contribute only negligibly to overall memory consumption. That is because some data structures partition their input into many small parts called *buckets* and store a seed for each of these buckets. The seeds may even make up the majority of the stored data. This is the case for many constructions of perfect hash functions, but also occurs in a construction of compressed static functions [2], which, in turn, enable space-efficient solutions to the approximate membership problem and the relative membership problem. The way we phrased the SSEP implicitly assumes that buckets are handled independently. We discuss a slight generalisation in the full version [24].

[2] It is possible to get $\mathbb{E}[|M|] \leq M^{\mathrm{OPT}} + \varepsilon \min(M^{\mathrm{OPT}}, n) + \mathcal{O}(\log n)$ and $\mathbb{E}[T_i] = \mathcal{O}(T_i^{\mathrm{OPT}}/\varepsilon)$ in general. This requires work if many p_i are close to 1 such that $M^{\mathrm{OPT}} = o(n)$. In that case we can "summarise" subsequences of $p_1, \ldots, p_n$ using essentially the UNI strategy and then apply our main theorem to a shorter sequence $\tilde{p}_1, \ldots, \tilde{p}_{n'}$ with $n' \leq M^{\mathrm{OPT}} + \mathcal{O}(1)$. We decided against integrating this improvement into our arguments as it interferes with clarity.

[3] The standard solution [32] using Golomb-Rice coding [11, 31] comes with $\Omega(n)$ bits of overhead.

Minimal Perfect Hash Functions. As an application of CONSENSUS we discuss the construction of minimal perfect hash functions (MPHFs). An MPHF on a set X of keys maps these keys to a range $[|X|]$ without collisions. The function does not necessarily have to store X explicitly. The space lower bound is $\mathrm{OPT}_{\mathrm{MPHF}} = |X| \log_2(e) - \mathcal{O}(\log |X|)$ bits assuming the universe of possible keys has size $\Omega(|X|^2)$ [1, 26, 25, 9]. While a succinct construction has long been known [12], its space overhead is fixed to $\varepsilon = \Theta(\frac{\log^2 \log n}{\log n})$ and large in practice. Many recent practical approaches for constructing MPHFs involve a brute force search for seed values that map certain subsets of the keys in favourable ways [30, 7, 14, 22, 21]. Those that do, all combine aspects of the MIN-strategy (store smallest working seeds) and the UNI-strategy (let a single seed do lots of work) and achieve a space budget of $(1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{MPHF}}$ only in time $|X| \cdot \exp(\Omega(1/\varepsilon))$ (see Section 5.1). In this paper we break this barrier for the first time. While we give the details in Section 5, the clean version of our result is the following.

▶ **Theorem 5** (Bucketed CONSENSUS-RecSplit). *For any $\varepsilon \in [n^{-3/7}, 1]$ there is an MPHF data structure with space requirement $(1 + \mathcal{O}(\varepsilon))\mathrm{OPT}_{\mathrm{MPHF}}$, expected construction time $\mathcal{O}(n/\varepsilon)$ and expected query time $\mathcal{O}(\log 1/\varepsilon)$.*

1.4 Overview of the Paper

The structure of the paper is as follows. In Section 2, we explain CONSENSUS in detail, as well as state the main theoretic results. In Sections 3 and 4 we then prove the results. We give an application of CONSENSUS to minimal perfect hashing in Section 5 and also briefly evaluate our practical implementation. Finally, we conclude the paper in Section 6.

2 Combined Search and Encoding for Successful Seeds

In this section we present two variants of our CONSENSUS algorithm. The first aims for conceptual clarity but produces seeds of length $\Omega(n)$ bits that cannot be decoded efficiently. The second is more technical but solves this problem.

In the following we always assume that $n \in \mathbb{N}$, $p_1, \ldots, p_n \in (0, 1]$ and $\varepsilon \in (0, 1]$ are given and $\vec{B}^{(i)} = (B_j^{(i)})_{j \in \mathbb{N}_0}$ is a Bernoulli process with parameter p_i for all $i \in [n]$.

A Branching Process. Let $k_1, \ldots, k_n \in \mathbb{N}$ be a sequence of numbers.[4] Consider the infinite tree visualised in Figure 1. It has $n+2$ layers, indexed from 0 to $n+1$. The root in layer 0 has an infinite number of children and nodes in layer $1 \leq i \leq n$ have k_i children each, with 0-based indexing in both cases. Edges between layer i and $i+1$ are associated with the variables $B_0^{(i)}, B_1^{(i)}, B_2^{(i)}, \ldots$ from top to bottom where $B_j^{(i)} = 0$ indicates a *broken* edge depicted as a dotted line. For convenience, we make another definition. For any $0 \leq \ell \leq n$ and any sequence $(\sigma_0, \sigma_1, \ldots, \sigma_\ell) \in \mathbb{N}_0 \times \{0, \ldots, k_1 - 1\} \times \ldots \times \{0, \ldots, k_\ell - 1\}$ consider the node reached from the root by taking in layer $0 \leq i \leq \ell$ the edge to the σ_i-th child. We define $\sigma_0 \circ \ldots \circ \sigma_\ell$ to be the index of the reached node within layer ℓ.

The Simplified CONSENSUS Algorithm. In CONSENSUS, we search for a non-broken path from the root of the tree to a leaf node using a DFS, see Algorithm 1. Note that such a path exists with probability 1 because the root has infinitely many children. The path is given by

[4] It may help to imagine that $k_i \approx 2^\varepsilon / p_i$.

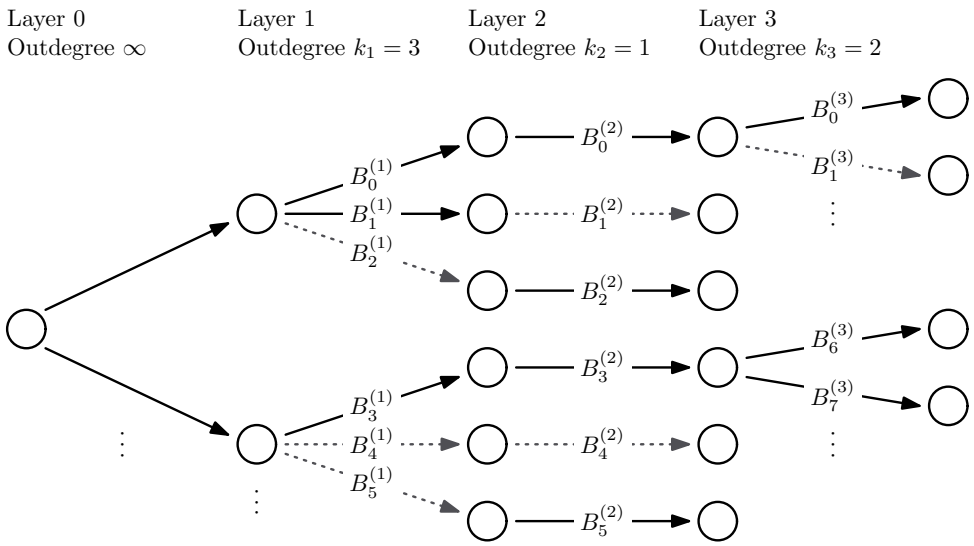


Figure 1 A visualisation of the tree underlying the search of the simplified CONSENSUS algorithm.

the sequence of choices $(\sigma_0, \dots, \sigma_n) \in \mathbb{N}_0 \times \{0, \dots, k_1 - 1\} \times \dots \times \{0, \dots, k_n - 1\}$ made in each layer. The returned sequence can be encoded as a bitstring M. Then $S_i \coloneqq \sigma_0 \circ \dots \circ \sigma_i$ is a successful seed for each $i \in [n]$ by construction. The merits of the approach are summarised in the following theorem.

▶ **Theorem 6** (Performance of Simplified CONSENSUS). *Given $n \in \mathbb{N}$, $p_1, \dots, p_n \in (0, 1]$ and $\varepsilon \in (0, 1]$ let $k_1, \dots, k_n \in \mathbb{N}$ be a sequence of numbers satisfying*[5] $\prod_{j=1}^{i} p_j k_j 2^{-\varepsilon} \in [1, 2]$ *for all $i \in [n]$. Then Algorithm 1 solves the SSEP with $\mathbb{E}[T_i] = \mathcal{O}(T_i^{\mathrm{OPT}}/\varepsilon)$ and $\mathbb{E}[|M|] \leq M^{\mathrm{OPT}} + \varepsilon n + \log_2 1/\varepsilon + \mathcal{O}(1)$.*

Full CONSENSUS Algorithm. The main issues with the simplified CONSENSUS algorithm are that $(\sigma_0, \dots, \sigma_n)$ is awkward to encode in binary and that using $\sigma_0 \circ \dots \circ \sigma_i$ directly as the seed S_i makes for seeds of size $\Omega(n)$ bits, which are inefficient to compute and handle. We solve these issues with three interventions.

1. We introduce a word size $w \in \mathbb{N}$ and encode σ_0 using w bits (hoping that it does not overflow, see below).
2. We demand that $k_1, \dots, k_n \in \mathbb{N}$ are powers of two, i.e. $k_i = 2^{\ell_i}$ for $\ell_i \in \mathbb{N}_0$. With our definition of "$\circ$", the binary representation of the number $\sigma_0 \circ \dots \circ \sigma_i$ now simply arises by concatenating the binary representations of $\sigma_0, \dots, \sigma_i$ (where σ_j is encoded using ℓ_j bits and σ_0 is encoded using w bits).
3. Instead of defining S_i to be $\sigma_0 \circ \dots \circ \sigma_i$, we define S_i to be the w-bit suffix of $\sigma_0 \circ \dots \circ \sigma_i$. We will show that this truncation is unlikely to cause collisions, i.e. any two sequences $(\sigma_0, \dots, \sigma_i)$ and $(\sigma'_0, \dots, \sigma'_i)$ we encounter have distinct suffixes $S_i \neq S'_i$. Intuitively, this guarantees that the search algorithm will always see "fresh" random variables and does not "notice" the change.

The full algorithm is given as Algorithm 2, using notation defined in Figure 2. Our formal result is as follows.

[5] Ideally we would want $k_j \coloneqq \frac{2^{\varepsilon}}{p_j}$ but the integer constraint on the k_j forces us to be more lenient.

Algorithm 1 Simplified CONSENSUS.

Input: $n, (p_1, \dots, p_n), (k_1, \dots, k_n)$ and $B_0^{(i)}, B_1^{(i)}, \dots \sim \mathrm{Ber}(p_i)$ for all $i \in [n]$.
Output: $(\sigma_0, \dots, \sigma_n)$ with $0 \le \sigma_i < k_i$ such that $B^{(i)}_{\sigma_0 \circ \sigma_1 \circ \dots \circ \sigma_i} = 1$ for all $i \in [n]$.
Algorithm Simplified-CONSENSUS:

```
for σ_0 ∈ {0, 1, 2, ...} do
    (σ_1, i) ← (0, 1)
    while i > 0 do
        if B^(i)_{σ_0 ∘ ... ∘ σ_i} = 1 then
            if i = n then
                return M = (σ_0, ..., σ_n)
            i ← i + 1
            σ_i ← 0
        else // backtrack and/or next seed
            while i > 0 and σ_i = k_i − 1 do
                i ← i − 1
            if i > 0 then
                σ_i ← σ_i + 1
```

Algorithm 2 Full CONSENSUS.

Input: $n, w, (p_1, \dots, p_n), (\ell_1, \dots, \ell_n)$ and $B_0^{(i)}, \dots, B_{2^w-1}^{(i)} \sim \mathrm{Ber}(p_i)$ for all $i \in [n]$.
Output: either $\perp$ or M as in Figure 2 such that $B_{S_i}^{(i)} = 1$ for all $i \in [n]$.
Algorithm Full-CONSENSUS:

```
for σ_0 ∈ {0, 1, ..., 2^w − 1} do // bounded
    (σ_1, i) ← (0, 1)
    while i > 0 do
        if B^(i)_{S_i} = 1 then // S_i as in Figure 2
            if i = n then
                return M = (σ_0, ..., σ_n)
            i ← i + 1
            σ_i ← 0
        else // backtrack and/or next seed
            while i > 0 and σ_i = 2^{ℓ_i} − 1 do
                i ← i − 1
            if i > 0 then
                σ_i ← σ_i + 1
return ⊥ // can fail
```

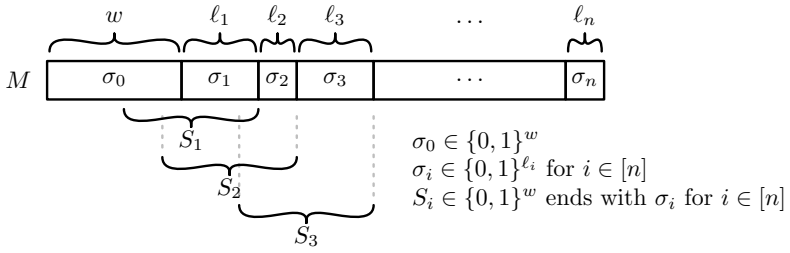


Figure 2 The bitstring M returned by the full CONSENSUS algorithm and the notation used to refer to relevant substrings.

▶ **Theorem 7** (Performance of Full CONSENSUS). *Let $c > 0$ be an arbitrary constant. Given $n \in \mathbb{N}$, $p_1, \dots, p_n \in (n^{-c}, 1]$ and $\varepsilon \in (n^{-c}, 1]$, let $\ell_1, \dots, \ell_n \in \mathbb{N}_0$ be the unique sequence of numbers such that*

$$\sum_{j=1}^{i} \ell_j \coloneqq \left\lceil \varepsilon i + \sum_{j=1}^{i} \log_2(1/p_i) \right\rceil \text{ for } i \in [n].$$

Assume we run Algorithm 2 with these parameters and $w \ge (2 + 7c) \log_2 n$.[6] Then there is an event E with $\Pr[E] = 1 - \mathcal{O}(n^{-c})$ that implies that the run solves the SSEP, producing a bitstring M with $|M| = \lceil M^{\mathrm{OPT}} + \varepsilon n + w \rceil$ containing $S_i \in \{0,1\}^w$ at offset $\sum_{j=1}^{i} \ell_j$ for all $i \in [n]$. Moreover, $\mathbb{E}[T_i \mid E] = \mathcal{O}(T_i^{\mathrm{OPT}}/\varepsilon)$ for all $i \in [n]$.

[6] We did not attempt to improve the constant "$2 + 7c$" and expect that $w = 64$ is enough in practice.

In particular the result guarantees that S_i can be inferred efficiently given M and i provided that $\sum_{j=1}^{i} \log_2(1/p_i)$ can be computed efficiently given i. In practice it is fine to use approximations of p_i.[7] Note that the expectation of T_i is only bounded conditioned on a good event E, which is defined in Section 4. To obtain an unconditionally bounded expectation, Algorithm 2 should be modified to give up early in failure cases.[8]

Special Cases and Variants. An important special case is $p_1 = \ldots = p_n = p$. With $b = \log_2 1/p$ the space lower bound is then simply $M^{\mathrm{OPT}} = bn$ bits. If we apply Theorem 7 with $\varepsilon \in (0,1)$ then the lengths $\ell_1, \ldots, \ell_n$ of the seed fragments $\sigma_1, \ldots, \sigma_n$ are all either $\lceil b+\varepsilon \rceil$ or $\lfloor b+\varepsilon \rfloor$ with an average approaching $b+\varepsilon$. Assuming $b \notin \mathbb{N}$ it may be tempting to use $\varepsilon = \lceil b \rceil - b$ such that all seed fragments have the same length $b+\varepsilon \in \mathbb{N}$. Similarly, we could pick ε such that $b+\varepsilon$ is a "nice" rational number, e.g. $b+\varepsilon = b' + \frac{1}{2}$ for $b' \in \mathbb{N}$ such that $\ell_1, \ldots, \ell_n$ alternate between $b'+1$ and b'.

In some cases it may be reasonable to choose $\varepsilon > 1$. Even though Consensus ceases to be more space efficient than MIN, it still has the advantage that seeds are bitstrings of fixed lengths in predetermined locations. We decided against formally covering this case in Theorem 7, but argue for the following modification in the full version [24] of this paper.

▶ Remark 8. The guarantees of Theorem 7 also hold for $\varepsilon \in [1, \log n]$, except that

$$\mathbb{E}[T_i \mid E] = T_i^{\mathrm{OPT}}(1 + \exp(-\Omega(2^{\varepsilon})) + n^{-c}) \qquad \text{(instead of } \mathbb{E}[T_i \mid E] = \mathcal{O}(T_i^{\mathrm{OPT}}/\varepsilon)\text{)}.$$

It may also be worthwhile to consider variants of Consensus that use different values of the overhead parameter ε in different parts of M to minimise some overall cost. Such an idea makes sense in general if we augment the SSEP to include specific costs c_i associated with testing the Bernoulli random variables belonging to the ith family.

3 Analysis of Simplified Consensus

We begin by reducing the claim of Theorem 6 to a purely mathematical statement given in Lemma 9, which we prove in Section 3.2.

3.1 Reduction of Theorem 6 to Lemma 9

Recall the tree defined in Section 2 and illustrated in Figure 1. For any $i \in [n+1]$ consider a node v in layer i. Let q_i be the probability that there is an unbroken path from v to a node in layer $n+1$. (This probability is indeed the same for all nodes in layer i.) In the last layer, we have $q_{n+1} = 1$ by definition. For all $1 \le i \le n$ we have $q_i = 1 - (1 - p_i q_{i+1})^{k_i}$ since $p_i q_{i+1}$ is the probability that an unbroken path via a specific child of v exists, and since there is an independent chance for each of the k_i children. The technical challenge of this section is to show that the q_i are not too small.

▶ **Lemma 9.** *Under the conditions of Theorem 6 we have $q_i = \Omega(\varepsilon)$ for all $i \in [n]$.*

This lemma implies Theorem 6 as follows.

[7] Our implementation uses fixed point arithmetic (integers counting "microbits") to avoid issues related to floating point arithmetic not being associative and distributive.

[8] To do so we need not decide whether E occurs. We can simply count the number of steps and abort when a limit is reached.

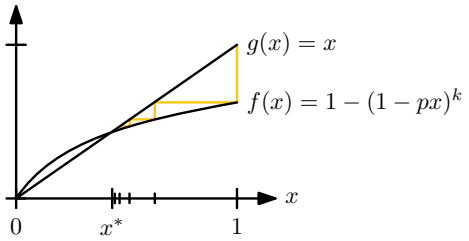


Figure 3 For $p_1 = \ldots = p_n = \frac{1}{4}$ and $k_1 = \ldots = k_n = 5$ (thus $\varepsilon = \log_2(5/4) \approx 0.32$) the values $q_{n+1}, q_n, q_{n-1}, \ldots$ (marked as ticks on the x-axis) arise by iteratively applying $f(x) = 1 - (1 - px)^k$ to the starting value of 1. Each q_i is at least the fixed point $x^* \approx 0.45$ of f.

Proof of Theorem 6. Let T_i for $i \in [n]$ be the number of Bernoulli random variables from the family $(B_j^{(i)})_{j \in \mathbb{N}_0}$ that are inspected. This corresponds to the number of edges between layers i and $i+1$ that are inspected by the DFS. With probability p_i such an edge is unbroken and, if it is unbroken, then with probability q_{i+1} the DFS will never backtrack out of the corresponding subtree and hence never inspect another edge from the same layer. This implies $T_i \sim \text{Geom}(p_i q_{i+1})$ and thus $\mathbb{E}[T_i] = \frac{1}{p_i q_{i+1}} = \mathcal{O}(T_i^{\text{OPT}}/\varepsilon)$ where the last step uses $T_i^{\text{OPT}} = 1/p_i$ and Lemma 9.

Concerning the space to encode $M = (\sigma_0, \ldots, \sigma_n)$, there are two parts to consider. The "fixed width" numbers $(\sigma_1, \ldots, \sigma_n) \in \{0, \ldots, k_1 - 1\} \times \ldots \times \{0, \ldots, k_n - 1\}$ can be encoded using $\lceil \log_2 \prod_{i \in [n]} k_i \rceil$ bits. By the assumption on $(k_i)_{i \in [n]}$ we have $\prod_{i \in [n]} k_i \leq 2 \cdot \prod_{i \in [n]} 2^\varepsilon / p_i$ and can bound the space by

$$1 + \log_2 \prod_{i \in [n]} k_i \leq 1 + \log_2 \left(2 \prod_{i \in [n]} \frac{2^\varepsilon}{p_i} \right) \leq 2 + \varepsilon n + \sum_{i \in [n]} \log_2 \frac{1}{p_i} = M^{\text{OPT}} + \varepsilon n + \mathcal{O}(1).$$

The "variable width" number $1 + \sigma_0 \in \mathbb{N}$ has distribution $1 + \sigma_0 \sim \text{Geom}(q_1)$ because σ_0 is the 0-based index of the first node in layer 1 with an unbroken path to layer $n+1$. Again by Lemma 9 we have $\mathbb{E}[1 + \sigma_0] = \frac{1}{q_1} = \mathcal{O}(1/\varepsilon)$. Using Jensen's inequality [15, 27] and the fact that $\log_2$ is concave, the expected number of bits to encode σ_0 is hence bounded by

$$\mathbb{E}[\lceil \log_2(1 + \sigma_0) \rceil] \leq 1 + \log_2 \mathbb{E}[1 + \sigma_0] = 1 + \log_2(\mathcal{O}(1/\varepsilon)) = \log_2 1/\varepsilon + \mathcal{O}(1).$$

Summing the contributions of both parts yields a bound on $\mathbb{E}[|M|]$ as claimed. ◀

3.2 Proof of Lemma 9

To understand the proof of Lemma 9 it may help to consider a simplified case first. Assume $p_1 = \ldots = p_n = p$ and $k_1 = \ldots = k_n = k$ with $kp = 2^\varepsilon$, i.e. k is slightly larger than $\frac{1}{p}$. We have $q_i = 1 - (1 - pq_{i+1})^k$, which motivates defining $f(x) = 1 - (1 - px)^k$ such that q_i arises by iteratively applying f to an initial value of $q_{n+1} = 1$. From the illustration in Figure 3 we see that q_i can never drop below the largest fixed point x^* of f, and it is not hard to show that $x^* = \Omega(\varepsilon)$.

Our full proof has to deal with a setting that is much less clean. The values p_i and k_i depend on i, yielding a distinct function f_i for each $i \in [n]$ and $p_i k_i \approx 2^\varepsilon$ is only true in the sense of a geometric average over $\Omega(1/\varepsilon)$ consecutive indices. We hence analyse compositions $f_a \circ \ldots \circ f_{b-1}$ for sufficiently large $b - a = \Theta(1/\varepsilon)$. We begin with some definitions.

- Let $f_i(x) = 1 - (1 - p_i x)^{k_i}$. Note that this gives $q_i = (f_i \circ \ldots \circ f_n)(1)$ for $i \in [n+1]$.

- For any set $I = \{a, \dots, b-1\}$ let $P_I = \prod_{i \in I} p_i$, $K_I = \prod_{i \in I} k_i$, and $F_I = f_a \circ \dots \circ f_{b-1}$. Note that the assumption of Theorem 6 gives

$$P_I K_I \cdot 2^{-\varepsilon|I|} = \underbrace{P_{\{1,\dots,b-1\}} K_{\{1,\dots,b-1\}} 2^{-\varepsilon(b-1)}}_{\in[1,2]} \Big(\underbrace{P_{\{1,\dots,a-1\}} K_{\{1,\dots,a-1\}} 2^{-\varepsilon(a-1)}}_{\in[1,2]} \Big)^{-1} \in [\tfrac{1}{2}, 2].$$

- Let $\varepsilon_0 := \lceil 2\varepsilon^{-1} \rceil^{-1}$. Its role is to be a number satisfying $\varepsilon_0 = \Theta(\varepsilon)$, $\varepsilon_0^{-1} \in \mathbb{N}$, and $(2^\varepsilon)^{\varepsilon_0^{-1}} \in [4, 8]$. In particular, if $I = \{a, \dots, b-1\}$ with $b - a = \varepsilon_0^{-1}$ then

$$P_I K_I = \underbrace{P_I K_I 2^{-\varepsilon|I|}}_{\in[\frac{1}{2},2]} \cdot \underbrace{2^{\varepsilon|I|}}_{\in[4,8]} \in [2, 16].$$

- Lastly, let $C = 8$ and $\varepsilon_1 = \varepsilon_0 / C$.

These definitions are designed to make the proof of the following lemma as simple as possible. No attempt was made to optimise constant factors.

▶ **Lemma 10.** *Let $I = \{a, \dots, b-1\} \subseteq [n]$. The following holds.*

(i) *For all $i \in [n]$ and $x \in [0, 1]$: $f_i(x) \geq p_i k_i x - (p_i k_i x)^2/2$.*

(ii) *If $b - a \leq \varepsilon_0^{-1}$ then $F_I(x) \geq P_I K_I x(1 - C|I|x)$ for $x \in [0, \varepsilon_1]$.*

(iii) *If $b - a = \varepsilon_0^{-1}$ then $F_I(\varepsilon_1/2) \geq \varepsilon_1/2$.*

Proof.

(i) e^{-x} satisfies the following bounds: $1 - x \leq e^{-x} \leq 1 - x + x^2/2$ for $x > 0$. We get the left claim by integrating $e^{-x} \leq 1$ for $t \in [0, x]$ to get $1 - e^{-x} \leq x$. Integrating again for $t \in [0, x]$ gives the right claim. Applying to $f_i(x)$ we get

$$f_i(x) = 1 - (1 - p_i x)^{k_i} \geq 1 - e^{-p_i k_i x} \geq 1 - (1 - p_i k_i x + (p_i k_i x)^2/2) = p_i k_i x - (p_i k_i x)^2/2.$$

(ii) We use induction on $b - a$. For $b - a = 0$ we have $I = \emptyset$, $F_I = \mathrm{id}$ as well as $P_I = K_I = 1$ and the claim holds. Let now $I = \{a, \dots, b-1\}$ with $b - a \leq \varepsilon_0^{-1}$and $I' = \{a + 1, \dots, b - 1\}$. We assume the claim holds for I'. Let $x \in [0, \varepsilon_1]$.

$$\begin{aligned}
F_I(x) = (f_a \circ F_{I'})(x) &\overset{\text{monotonic}}{=} f_a(F_{I'}(x)) \overset{\text{induction}}{\geq} f_a(P_{I'} K_{I'} x(1 - C|I'|x)) \\
&\overset{\text{(i)}}{\geq} p_a k_a P_{I'} K_{I'} x(1 - C|I'|x) - (p_a k_a P_{I'} K_{I'} x(1 - C|I'|x))^2/2 \\
&= P_I K_I x(1 - C|I'|x) - (P_I K_I x(1 - \underbrace{C|I'|x}_{\leq C\varepsilon_0^{-1}\varepsilon_1 = 1 < 2}))^2/2 \quad \text{and hence } (1 - C|I'|x)^2 \leq 1 \\
&\geq P_I K_I x(1 - C|I'|x) - (P_I K_I x)^2/2 = P_I K_I x(1 - x(C|I'| + \underbrace{P_I K_I}_{\leq 16 = 2C}/2)) \\
&\geq P_I K_I x(1 - x(C|I'| + C)) = P_I K_I x(1 - C|I|x).
\end{aligned}$$

(iii) By applying (ii) and using $C\varepsilon_0^{-1}\varepsilon_1 = 1$ we get

$$F_I(\varepsilon_1/2) \overset{\text{(ii)}}{\geq} \underbrace{P_I K_I}_{\geq 2} \tfrac{\varepsilon_1}{2} \underbrace{(1 - C\varepsilon_0^{-1} \tfrac{\varepsilon_1}{2})}_{= \frac{1}{2}} \geq \varepsilon_1/2.$$ ◀

The proof of Lemma 9 is now immediate:

Proof of Lemma 9. Let $i \in [n]$. Partition $I = \{i, \dots, n\}$ into contiguous intervals $I_1, \dots, I_k$ such that $F_I = F_{I_1} \circ \dots \circ F_{I_k}$ with $|I_1| \leq \varepsilon_0^{-1}$ and $|I_j| = \varepsilon_0^{-1}$ for $j \in \{2, \dots, k\}$. Then by repeated applications of Lemma 10 (iii), one application of Lemma 10 (ii) and the monotonicity of the relevant functions we get

$$q_i = F_I(1) \geq F_I(\varepsilon_1/2) = F_{I_1}(F_{I_2}(\dots(F_{I_k}(\varepsilon_1/2))\dots)) \overset{\text{(iii)}}{\geq} F_{I_1}(\varepsilon_1/2)$$
$$\overset{\text{(ii)}}{\geq} P_{I_1} K_{I_1} \frac{\varepsilon_1}{2}(1 - C|I_1|\frac{\varepsilon_1}{2}) = \underbrace{P_{I_1} K_{I_1} 2^{-\varepsilon|I_1|}}_{\geq \frac{1}{2}} \underbrace{2^{\varepsilon|I_1|}}_{\geq 1} \frac{\varepsilon_1}{2}(1 - \underbrace{C|I_1|\frac{\varepsilon_1}{2}}_{\leq \frac{1}{2}}) \geq \frac{\varepsilon_1}{8}. \quad \blacktriangleleft$$

4 Analysis of Full Consensus

Full Consensus performs a DFS in a tree as depicted in Figure 1 in much the same way as simplified Consensus, with two differences. First, the root node only has 2^w children (not infinitely many) and the probability that an unbroken path exists is therefore strictly less than 1. Secondly, when examining the outgoing edge of index σ_i of some node in layer i, then its status is no longer linked to B^i_Σ for $\Sigma = \sigma_0 \circ \dots \circ \sigma_i$. Instead Σ is divided as $\Sigma = \text{prefix}(\Sigma) \circ \text{suffix}(\Sigma)$ where $\text{suffix}(\Sigma) \in \{0,1\}^w$ and $B^i_{\text{suffix}(\Sigma)}$ is used. To prepare for the proof of Theorem 7, we present a coupling to, and a claim about simplified Consensus.

A Coupling between Simplified and Full Consensus. Consider the following natural coupling. Let $B^{(i)}_j$ for $i \in [n]$ and $j \in \mathbb{N}_0$ be the random variables underlying simplified Consensus and $\tilde{B}^{(i)}_S$ for $i \in [n]$ and $S \in \{0,1\}^w$ the random variables underlying full Consensus. The coupling should ensure that $B^{(i)}_\Sigma = \tilde{B}^{(i)}_{\text{suffix}(\Sigma)}$ for all pairs (i, Σ) such that simplified Consensus inspects $B^{(i)}_\Sigma$ and does not at an earlier time inspect $B^{(i)}_{\Sigma'}$ with $\text{suffix}(\Sigma') = \text{suffix}(\Sigma)$. In other words, for any pair (i, S), if simplified consensus inspects at least one variable $B^{(i)}_\Sigma$ with $\text{suffix}(\Sigma) = S$ then $\tilde{B}^{(i)}_S$ equals the earliest such variable that is inspected.

This implies that the runs of simplified and full Consensus behave identically as long as, firstly, σ_0 never reaches 2^w and, secondly, simplified Consensus never *repeats a suffix*, by which we mean inspecting $B^{(i)}_\Sigma$ for some pair (i, Σ) such that $B^{(i)}_{\Sigma'}$ with $\text{suffix}(\Sigma) = \text{suffix}(\Sigma')$ was previously inspected. In the following, by a *step* of Consensus we mean one iteration of its main while-loop, which involves inspecting exactly one random variable.

▷ Claim 11. Simplified Consensus repeats a suffix in step t with probability at most $\frac{t}{2^w}$.

Proof. Assume simplified Consensus is about to inspect $B^{(i)}_{\Sigma^*}$. Call $S \in \{0,1\}^w$ a *blocked suffix* if there exists $\Sigma \in \{0,1\}^*$ such that $S = \text{suffix}(\Sigma)$ and $B^{(i)}_\Sigma$ was previously inspected. This necessitates $\text{prefix}(\Sigma) < \text{prefix}(\Sigma^*)$. Clearly, step t repeats a suffix if $\text{suffix}(\Sigma^*)$ is blocked. A crucial observation is that the DFS has exhaustively explored all branches of the tree associated with any prefix less than Σ^* and the order in which this exploration has been carried out is no longer relevant. The symmetry between all nodes of the same layer therefore implies that every suffix is blocked with the same probability. Since at most t suffixes can be blocked at step t, the probability that Σ^* is blocked (given our knowledge of (i, Σ^*)) is at most $\frac{t}{2^w}$. ◁

Proof of Theorem 7. First note that the choice of $(\ell_1, \dots, \ell_n)$ made in Theorem 7 amounts to a valid choice of $(k_1 = 2^{\ell_1}, \dots, k_n = 2^{\ell_n})$ for applying Theorem 6. Indeed, for each $i \in [n]$ there exists some $\text{err}_i \in [1, 2)$ related to rounding up such that

$$\prod_{j=1}^{i} p_j k_j 2^{-\varepsilon} = \Big(\prod_{j=1}^{i} p_j\Big) 2^{\sum_{j=1}^{i} \ell_j} 2^{-\varepsilon i} = \Big(\prod_{j=1}^{i} p_j\Big) 2^{\lceil \varepsilon i + \sum_{j=1}^{i} \log_2(1/p_i)\rceil} 2^{-\varepsilon i}$$
$$= \Big(\prod_{j=1}^{i} p_j\Big) 2^{\varepsilon i + \sum_{j=1}^{i} \log_2(1/p_i)} \cdot \mathrm{err}_i 2^{-\varepsilon i} = \mathrm{err}_i \in [1,2]$$

as required. The number $T = \sum_{i=1}^{n} T_i$ of steps taken by simplified Consensus satisfies $\mathbb{E}[T] = \mathcal{O}(\sum_{i=1}^{n} \frac{1}{p_i \varepsilon}) = \mathcal{O}(n^{1+2c})$. By Markov's inequality T exceeds its expectation by a factor of n^c with probability at most n^{-c} so with probability at least $1 - \mathcal{O}(n^{-c})$ simplified Consensus terminates within $T_{\text{limit}} = n^{1+3c}$ steps.

By Claim 11 and a union bound the probability that simplified Consensus repeats a suffix within its first T_{limit} steps is at most $\sum_{t=1}^{T_{\text{limit}}} t/2^w = \mathcal{O}(T_{\text{limit}}^2/2^w) = \mathcal{O}(n^{2+6c}/2^w)$. By choosing $w \geq (2+7c)\log_2 n$ this probability is $\mathcal{O}(n^{-c})$. This choice also guarantees that σ_0 never exceeds 2^w during the first T_{limit} steps. Let now $E = \{T \leq T_{\text{limit}}\} \cap \{\text{no repeated suffix in the first } T_{\text{limit}} \text{ steps}\}$. By a union bound $\Pr[E] = 1 - \mathcal{O}(n^{-c})$ and by our coupling E implies that simplified and full Consensus explore the same sequence of nodes in their DFS and terminate with the same sequence $\sigma_0, \ldots, \sigma_n$ with $\sigma_0 < 2^w$. In particular full Consensus succeeds in that case as claimed.

The updated bound on $|M|$ simply reflects that σ_0, which in simplified Consensus had expected length $\log_2 1/\varepsilon + \mathcal{O}(1)$ now has fixed length $w = \mathcal{O}(\log n)$. Lastly, if T_i and $\tilde{T}_i$ denote the number of Bernoulli trials from the i-th family inspected by simplified and full Consensus, respectively, then conditioned on E we have $T_i = \tilde{T}_i$ so

$$\mathbb{E}[\tilde{T}_i \mid E] = \mathbb{E}[T_i|E] \leq \mathbb{E}[T_i]/\Pr[E] = \mathcal{O}(\mathbb{E}[T_i]) = \mathcal{O}(T_i^{\text{OPT}}/\varepsilon). \quad \blacktriangleleft$$

5 Application to Minimal Perfect Hashing

In this section, we apply the Consensus idea to the area of minimal perfect hash functions (MPHFs). Recall from Section 1.3 that an MPHF for a given set X of n keys maps each key in X to a unique integer from $[n]$.

5.1 Existing Strategies for Constructing MPHFs

We begin with a brief overview of established approaches with a focus on those we need. Detailed explanations are found in [18, 19].

Partitioning. Most approaches randomly partition the input into small *buckets* using a hash function and then construct an MPHF on each bucket separately. The natural way to obtain an MPHF on the entire input involves storing the bucket-MPHFs and prefix sums for the bucket sizes. Storing the prefix sums can be avoided when the partitioning uses a k-perfect hash function, as is done in ShockHash-Flat [21], and as we do in Section 5.3.

Hagerup and Tholey [12] demonstrate that partitioning into tiny buckets of size $\mathcal{O}(\frac{\log n}{\log\log n})$ when suitably combined with exhaustive tabulation of MPHFs on tiny inputs yields an approach with construction time $\mathcal{O}(n)$, query time $\mathcal{O}(1)$ and space overhead $\varepsilon = \Theta(\frac{\log^2 \log n}{\log n})$. Unfortunately the analysis requires astronomical values of $n \geq \max(\exp(\omega(1/\varepsilon)), 2^{150})$ [4] and is restricted to these rather large ε.

Below we explain *monolithic* variants of MPHFs, i.e. constructions not using partitioning, which are typically combined with partitioning to obtain faster *bucketed* variants.

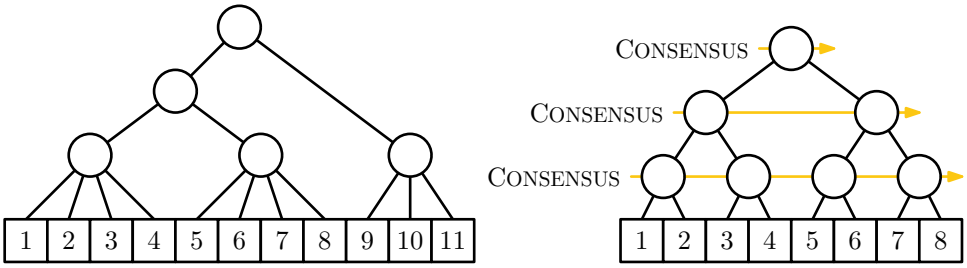


Figure 4 A general splitting tree and a balanced binary splitting tree as used in Section 5.2.

Recursive Splitting. A recursive splitting strategy [7] involves a predefined *splitting tree*, which is a rooted tree with n leaves.[9] See Figure 4 (left) for an example with $n = 11$. An MPHF for an input set of size n is given by a family of functions containing for each internal node v of the tree a *splitting hash function* f_v that maps keys to the children of v. Together, the family $(f_v)_v$ must describe a recursive partitioning of the input set into parts of size 1. The MPHF is queried for a key x by starting at the root. The splitting hash functions are evaluated on x and the resulting path is followed until a leaf is reached, the index of which is returned. Each f_v is identified by a seed that is found using trial and error. In the original paper [7] seeds are stored using Golomb-Rice coding [11, 31]. To keep the space overhead low, the last layer of the splitting tree uses ℓ-way splits for relatively large ℓ (5–16 in practice), which makes splitting hash functions costly to find.

Multiple-Choice Hashing. We randomly assign a set $\{h_1(x), \ldots, h_k(x)\} \subseteq [n(1+\varepsilon)]$ of candidates to each key hoping that a choice $\sigma(x) \in \{1, \ldots, k\}$ can be made for each key such that $x \mapsto h_{\sigma(x)}(x)$ is injective. A perfect hash function is then given by a retrieval data structure that stores σ [4, 10, 20]. A recent variant ShockHash [22, 21] achieves record space efficiency by using the aggressive configuration $\varepsilon = 0$ and $k = 2$ (combined with partitioning and recursive splitting). Many retries are needed until σ exists.

Bucket Placement. A general strategy with many variants assigns a random bucket $f(x) \in \{1, \ldots, b\}$ to each key and considers for each bucket $i \in [b]$ several (hash) functions $g_{i,1}, g_{i,2}, \ldots$ with range $[n]$. A choice $\sigma(i)$ is stored for each bucket such that $x \mapsto g_{f(x),\sigma(f(x))}(x)$ is a bijection. Typically, the values $\sigma(i)$ are chosen greedily in decreasing order of bucket sizes [30, 14, 1], but backtracking has also been considered [33, 8] (in a different sense than Consensus).

5.2 Monolithic Consensus-RecSplit

We now describe a variant of recursive splitting in an idealised setting with $n = 2^d$ that uses Consensus to be particularly space efficient. The idea is simple. The data structure consists of $n - 1$ two-way splitting hash functions, i.e. hash functions with range $\{0, 1\}$, arranged in a balanced binary tree with n leaves, see Figure 4 (right).

This means that during construction for a key set S we have to select the splitting hash functions from top to bottom such that each function splits the part of S arriving at its node perfectly in half. The data structure is entirely given by a sequence of $n - 1$ seeds that identify the splitting hash functions. If we search and encode these seeds using the Consensus algorithm, we obtain the following result.

[9] In the original paper [7], the lowest level is suppressed, with correspondingly changed terminology.

▶ **Theorem 12** (Monolithic CONSENSUS-RecSplit). *Let $n = 2^d$ for $d \in \mathbb{N}$ and $\varepsilon \in [\frac{1}{n}, 1]$. Monolithic-RecSplit is an MPHF data structure with space requirement* $\mathrm{OPT}_{\mathrm{MPHF}} + \mathcal{O}(\varepsilon n + \log^2 n)$*, expected construction time* $\mathcal{O}(\max(n^{3/2}, \frac{n}{\varepsilon}))$ *and query time* $\mathcal{O}(\log n)$*.*

Proof. We use a separate CONSENSUS data structure per level of the splitting tree and construct them from top to bottom. Consider level $\ell \in \{0, \dots, d-1\}$ of the tree, which contains 2^ℓ nodes responsible for $n_\ell = 2^{d-\ell}$ keys each. The probability that a given seed for a splitting hash function amounts to a perfect split for n_ℓ keys is $p(n_\ell) = \binom{n_\ell}{n_\ell/2} \cdot 2^{-n_\ell}$ by a simple counting argument. A standard approximation gives $p(n_\ell) = \Theta(1/\sqrt{n_\ell})$.

By applying Theorem 7 with $p_1 = \ldots = p_{2^\ell} = p(n_\ell)$, $\varepsilon_\ell = \min(1, \varepsilon \cdot n_\ell^{3/4})$ and $w = \mathcal{O}(\log n)$ we obtain a data structure encoding 2^ℓ successful seeds. The expected number of seeds inspected is $\mathcal{O}(\frac{1}{p(n_\ell)\varepsilon_\ell})$ for each node, which corresponds to expected work of $\mathcal{O}(\frac{n_\ell}{p(n_\ell)\varepsilon_\ell}) = \mathcal{O}(n_\ell^{3/2}/\varepsilon_\ell)$ per node when seeds are tested by hashing all n_ℓ relevant keys.[10] For the total expected work w_ℓ in level ℓ there are, up to constant factors, the following two options.

$$\text{if } \varepsilon_\ell = \varepsilon \cdot n_\ell^{3/4} \text{ then } w_\ell = \frac{n_\ell^{3/2}}{\varepsilon n_\ell^{3/4}} \cdot 2^\ell = \frac{n_\ell^{3/4} 2^\ell}{\varepsilon} = \frac{2^{(d-\ell)\cdot 3/4 + \ell}}{\varepsilon} = \frac{n^{3/4} \cdot 2^{\ell/4}}{\varepsilon}.$$

$$\text{if } \varepsilon_\ell = 1 \text{ then } w_\ell = n_\ell^{3/2} \cdot 2^\ell = 2^{(d-\ell)\cdot 3/2 + \ell} = n^{3/2} \cdot 2^{-\ell/2}.$$

The formula for the first case is maximised for $\ell = d-1$ giving $w_{d-1} = \Theta(n/\varepsilon)$. The formula for the second case is maximised for $\ell = 0$ giving $w_0 = n^{3/2}$. Since both describe geometric series we get $\sum_{\ell=0}^{d-1} w_\ell = \mathcal{O}(w_0 + w_{d-1}) = \mathcal{O}(n^{3/2} + n/\varepsilon) = \mathcal{O}(\max(n^{3/2}, n/\varepsilon))$ as desired.

The space requirement for level ℓ is $|M_\ell| = 2^\ell \cdot \log_2(1/p(n_\ell)) + \varepsilon_\ell 2^\ell + \mathcal{O}(\log n)$. To bound the sum $\sum_{\ell=0}^{d-1} |M_\ell|$ we look at the terms in $|M_\ell|$ separately. For the first, we will use that $\prod_{\ell=0}^{d-1} p(2^{d-\ell})^{2^\ell} = \frac{n!}{n^n}$, which can be checked by induction: $d = 0$ is clear. For $d > 0$ we have

$$\prod_{\ell=0}^{d-1} p(2^{d-\ell})^{2^\ell} = p(n) \cdot \prod_{\ell=1}^{d-1} p(2^{d-\ell})^{2^\ell} = p(n) \cdot \Big(\prod_{\ell=1}^{d-1} p(2^{d-\ell})^{2^{\ell-1}} \Big)^2$$

$$= p(n) \cdot \Big(\prod_{\ell=0}^{d-2} p(2^{d-1-\ell})^{2^\ell} \Big)^2 \overset{\text{Ind.}}{=} p(n) \cdot \Big(\frac{(n/2)!}{(n/2)^{n/2}} \Big)^2 = \binom{n}{n/2} 2^{-n} \cdot \Big(\frac{(n/2)!}{(n/2)^{n/2}} \Big)^2 = \frac{n!}{n^n}.$$

The leading terms in $|M_\ell|$ therefore add up to

$$\sum_{\ell=0}^{d-1} 2^\ell \cdot \log_2(1/p(n_\ell)) = \log_2 \Big(1/ \prod_{\ell=0}^{d-1} p(n_\ell)^{2^\ell} \Big) = \log_2 \big(\frac{n^n}{n!} \big) = n \log_2 e - \mathcal{O}(\log n).$$

The overhead terms $\varepsilon_\ell \cdot 2^\ell$ are geometrically increasing in ℓ. This is because

$$\frac{\varepsilon_{\ell+1} \cdot 2^{\ell+1}}{\varepsilon_\ell \cdot 2^\ell} = \frac{\min(1, \varepsilon \cdot n_{\ell+1}^{3/4})}{\min(1, \varepsilon \cdot n_\ell^{3/4})} \cdot 2 = \frac{\min(1, \varepsilon \cdot (2^{d-\ell-1})^{3/4})}{\min(1, \varepsilon \cdot (2^{d-\ell})^{3/4})} \cdot 2 \geq 2^{-3/4} \cdot 2 = 2^{1/4} > 1.$$

The sum of the overhead terms is therefore dominated by the last term $\varepsilon_{d-1} 2^{d-1} \leq \varepsilon n \cdot 2^{-1/4}$ giving $\sum_{\ell=0}^{d-1} \varepsilon_\ell \cdot 2^\ell \leq \varepsilon n / (2^{1/4} - 1) < 6\varepsilon n$. Overall we get as claimed

$$\text{space} = \sum_{\ell=0}^{d-1} |M_\ell| \leq n \log_2 e + 6\varepsilon n + \mathcal{O}(\log^2 n).$$

[10] Dividing the work per node by ε_ℓ gives $-n_\ell^{3/2}/\varepsilon_\ell^2$. Our choice of $\varepsilon_\ell \sim n_\ell^{3/4}$ (as long as $\min(1, \cdot)$ does not kick in) is desired to ensure that using extra bits amounts to roughly the same reduction of work in each level, such that we have a consistent trade-off between time and space.

A query needs to extract one seed from each of the $d = \log_2 n$ CONSENSUS data structures. Since each uses uniform values for $p_1 = \ldots = p_{n_\ell}$, the product $\prod_{j=1}^{i} p_j = p(n_\ell)^i$ is easy to compute in constant time, which is sufficient to find each seed in constant time, so we get a query time of $\mathcal{O}(d)$ in total. ◀

5.3 Bucketed CONSENSUS-RecSplit

We now replace the upper layers of Monolithic CONSENSUS-RecSplit with a minimal k-perfect hash function [13]. This reduces query time as there are fewer layers to traverse and reduces construction time because the splits in the top layers are the most expensive ones to compute. Most importantly, it enables efficiently handling input sizes that are not powers of two.

For any $k \in \mathbb{N}$ a minimal k-perfect hash function for a set S of n keys is a data structure mapping the elements of S to buckets indexed with $[\lceil n/k \rceil]$ such that each bucket $i \in [\lfloor n/k \rfloor]$ receives exactly k keys. If k does not divide n then bucket $\lceil n/k \rceil$ receives between 1 and $k-1$ leftover keys. It has long been known that the space lower bound for such a data structure is $\approx n(\log_2 e - \frac{1}{k}\log_2 \frac{k^k}{k!})$ [1], shown to be $\approx \frac{n}{2k}\log 2\pi k$ in [16]. In the full version of this paper [24] we outline a compact minimal k-perfect hash function with, in expectation, $\mathcal{O}(\frac{n}{k}\log k)$ bits of space, constant query time, and linear construction time.

We call our resulting MPHF Bucketed CONSENSUS-RecSplit. Its performance is as follows (previously stated in Section 1.3).

▶ **Theorem 5** (Bucketed CONSENSUS-RecSplit). *For any $\varepsilon \in [n^{-3/7}, 1]$ there is an MPHF data structure with space requirement $(1 + \mathcal{O}(\varepsilon))\mathrm{OPT}_{\mathrm{MPHF}}$, expected construction time $\mathcal{O}(n/\varepsilon)$ and expected query time $\mathcal{O}(\log 1/\varepsilon)$.*

Proof. The idea is to use our monolithic construction but replace the top-most layers with a k-perfect hash function. More precisely, we choose $k = 2^{\lfloor \frac{4}{3}\log_2 1/\varepsilon \rfloor} = \Theta(\varepsilon^{-4/3})$ and construct a minimal k-perfect hash function F for the input set. As discussed in the full version, the construction time of F is $\mathcal{O}(n)$, the expected query time is $\mathcal{O}(1)$ and the space consumption is $\mathcal{O}(\frac{n}{k}\log k) = \mathcal{O}(\varepsilon n)$ bits, all well within the respective budgets. The resulting $\lfloor n/k \rfloor$ buckets of k keys each (a power of 2) are recursively split like in our monolithic data structure, which takes at most $\mathrm{OPT}_{\mathrm{MPHF}} + \mathcal{O}(\varepsilon n)$ bits of space (the $\mathcal{O}(\log^2 n)$ term is dominated by $\mathcal{O}(\varepsilon n)$). Since we make no assumptions on n there might be up to $k-1$ leftover keys assigned to an extra bucket. For these we construct a separate MPHF F' with range $\{\lfloor n/k \rfloor \cdot k + 1, \ldots, n\}$. Since $k \sim \varepsilon^{-4/3} = \varepsilon \cdot \varepsilon^{-7/3} \leq \varepsilon n$ pretty much any compact (not necessarily succinct) MPHF is good enough here, provided it supports queries in logarithmic time. Queries evaluate F first and, if mapped to a regular bucket, evaluate a sequence of $\log_2 k$ splits, or, if mapped to the extra bucket, evaluate F'. This takes $\mathcal{O}(\log k) = \mathcal{O}(\log 1/\varepsilon)$ time in the worst case.

Concerning construction time, note that the top-most level in use has nodes of size $n_\ell = k \leq \varepsilon^{-4/3}$ and the definition of ε_ℓ we made simplifies to $\varepsilon_\ell = n_\ell^{3/4} \cdot \varepsilon$ without the "$\min(1, \cdot)$". By an argument in Theorem 12, the construction time is then dominated by the bottom layer and bounded by $\mathcal{O}(n/\varepsilon)$ as claimed. ◀

5.4 Experiments

To demonstrate that CONSENSUS-RecSplit is viable in practice, we give an implementation and compare it to other MPHF constructions from the literature.[11] We only include the most space-efficient previous approaches with space consumption below 1.6 bits per key and

[11] We run single-threaded experiments on an Intel i7 11700 processor with a base clock speed of 2.5 GHz. The sizes of the L1 and L2 data caches are 48 KiB and 512 KiB per core, and the L3 cache has a size of

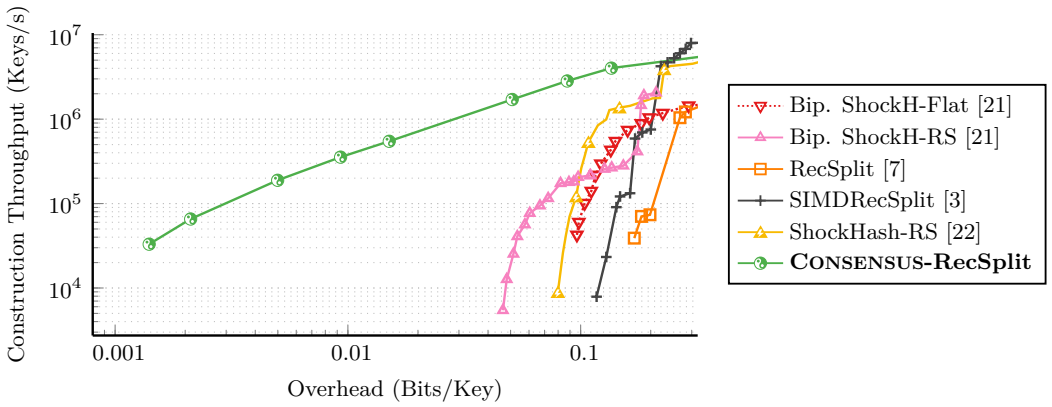


Figure 5 Experimental evaluation of very space-efficient MPHF algorithms, showing the trade-off between space consumption and construction throughput. Because CONSENSUS-RecSplit is focused on space consumption, we only include approaches that achieve below 1.6 bits per key.

refer to [18, 19] for a detailed comparison of state-of-the-art perfect hashing approaches with larger space consumption. Our implementation departs from the algorithmic description in Section 5.3 by using the threshold-based k-perfect hash function [21, 13]. We postpone further tuning and parallelisation efforts to a future paper.

Figure 5 shows the trade-off between space consumption and construction time of CONSENSUS-RecSplit. In the full version [24] we also give a table. The performance of all previous approaches degrades abruptly for smaller overheads because of their exponential dependency on $1/\varepsilon$ while CONSENSUS-RecSplit shows an approximately linear dependence on $1/\varepsilon$ as predicted in Theorem 5. The space lower bound for MPHFs is $\log_2 e \approx 1.443$ bits per key. The previously most space-efficient approach, bipartite ShockHash-RS [21], achieves 1.489 bits per key. In about 15% of the construction time, CONSENSUS-RecSplit achieves a space consumption of just 1.444 bits per key. At a space consumption of 1.489 bits per key, CONSENSUS-RecSplit is more than 350 times faster to construct than the previous state of the art.

Queries take 150–250 ns depending on k and therefore the depth of the splitting tree (see full version), so their performance is not too far from compact configurations of RecSplit (100 ns) and bipartite ShockHash-RS (125 ns). Future work can improve on the query performance by storing the seeds needed for each query closer to each other. We can adapt bucketed CONSENSUS-RecSplit by constructing one CONSENSUS data structure storing seeds bucket by bucket instead of an independent data structure for each layer. Preliminary experiments show up to 30% faster queries with moderately slower construction at the same space consumption. Splittings higher up in the tree not only have a smaller success probability, but are also more costly to test. We are therefore incentivised to select a different space overhead parameter ε for each splitting within the same CONSENSUS data structure. We leave this generalisation of CONSENSUS to non-uniform ε for future work.

16 MiB. The page size is 4 KiB. We use the GNU C++ compiler version 11.2.0 with optimization flags `-O3 -march=native`. Like previous papers [19, 21, 22, 14, 3], we use 100 million random string keys of uniform random length [10..50] as input. The source code is available on GitHub [17, 23].

6 Conclusion

This paper concerns data structures that store sequences of successful seeds, i.e. values known to influence pseudo-random processes in desirable ways. Normally these seeds are first found independently of each other using trial and error and then encoded separately. In this paper we show that, perhaps surprisingly, this wastes $\Omega(1)$ bits per seed compared to a lower bound.

We present the CONSENSUS approach, which **co**mbi**n**es **s**earch and **en**coding of **su**ccessful **s**eeds and reduces the space overhead to $\mathcal{O}(\varepsilon)$ bits at the price of increasing the required work by a factor of $\mathcal{O}(1/\varepsilon)$.

To demonstrate the merits of our ideas in practice, we apply it to the construction of minimal perfect hash functions. CONSENSUS-RecSplit is the first MPHF to achieve a construction time that only depends linearly on the inverse space overhead. Despite using the same recursive splitting idea as [7], the space overhead is, for the same construction throughput, up to two orders of magnitude smaller in practice.

Future Work. Given our promising results, we intend to continue working on CONSENSUS-RecSplit, in particular looking into parallelisation and improving query times. We also believe that CONSENSUS can be applied to other randomised data structures, including but not limited to other perfect or k-perfect hash functions, retrieval data structures and AMQ-filters. We give a first result on k-perfect hash functions using CONSENSUS in [13].

References

1 Djamal Belazzougui, Fabiano C. Botelho, and Martin Dietzfelbinger. Hash, displace, and compress. In *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 682–693. Springer, 2009. doi:10.1007/978-3-642-04128-0_61.

2 Djamal Belazzougui and Rossano Venturini. Compressed static functions with applications. In *SODA*, pages 229–240. SIAM, 2013. doi:10.1137/1.9781611973105.17.

3 Dominik Bez, Florian Kurpicz, Hans-Peter Lehmann, and Peter Sanders. High performance construction of RecSplit based minimal perfect hash functions. In *ESA*, volume 274 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.19.

4 Fabiano C. Botelho, Rasmus Pagh, and Nivio Ziviani. Practical perfect hashing in nearly optimal space. *Inf. Syst.*, 38(1):108–131, 2013. doi:10.1016/J.IS.2012.06.002.

5 Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 6–19. Springer, 1990. doi:10.1007/BFB0032018.

6 Martin Dietzfelbinger and Michael Rink. Applications of a splitting trick. In *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 354–365. Springer, 2009. doi:10.1007/978-3-642-02927-1_30.

7 Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. RecSplit: Minimal perfect hashing via recursive splitting. In *ALENEX*, pages 175–185. SIAM, 2020. doi:10.1137/1.9781611976007.14.

8 Edward A. Fox, Lenwood S. Heath, Qi Fan Chen, and Amjad M. Daoud. Practical minimal perfect hash functions for large databases. *Commun. ACM*, 35(1):105–121, 1992. doi:10.1145/129617.129623.

9 Michael L. Fredman and J'anos Koml'os. On the size of separating systems and families of perfect hash functions. *SIAM J. Algebraic Discrete Methods*, 5(1):61–68, 1984. doi:10.1137/0605009.

10 Marco Genuzio, Giuseppe Ottaviano, and Sebastiano Vigna. Fast scalable construction of (minimal perfect hash) functions. In *SEA*, volume 9685 of *Lecture Notes in Computer Science*, pages 339–352. Springer, 2016. doi:10.1007/978-3-319-38851-9_23.

11 Solomon W. Golomb. Run-length encodings. *IEEE Trans. Inf. Theory*, 12(3):399–401, 1966. doi:10.1109/TIT.1966.1053907.

12 Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In *STACS*, volume 2010 of *Lecture Notes in Computer Science*, pages 317–326. Springer, 2001. doi:10.1007/3-540-44693-1_28.

13 Stefan Hermann, Sebastian Kirmayer, Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. Engineering minimal k-perfect hash functions. In *ESA*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025.

14 Stefan Hermann, Hans-Peter Lehmann, Giulio Ermanno Pibiri, Peter Sanders, and Stefan Walzer. PHOBIC: perfect hashing with optimized bucket sizes and interleaved coding. In *ESA*, volume 308 of *LIPIcs*, pages 69:1–69:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.69.

15 Johan Ludwig William Valdemar Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta mathematica*, 30(1):175–193, 1906. doi:10.1007/BF02418571.

16 Florian Kurpicz, Hans-Peter Lehmann, and Peter Sanders. PaCHash: Packed and compressed hash tables. In *ALENEX*, pages 162–175. SIAM, 2023. doi:10.1137/1.9781611977561.CH14.

17 Hans-Peter Lehmann. ByteHamster/MPHF-Experiments. Software, swhId: swh:1:dir:7b02eda8ffa5a9d61a086ff2fc34fb0fbdc3eff4 (visited on 2025-07-16). URL: https://github.com/ByteHamster/MPHF-Experiments, doi:10.4230/artifacts.23790.

18 Hans-Peter Lehmann. *Fast and Space-Efficient Perfect Hashing*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2024. doi:10.5445/IR/1000176432.

19 Hans-Peter Lehmann, Thomas Mueller, Rasmus Pagh, Giulio Ermanno Pibiri, Peter Sanders, Sebastiano Vigna, and Stefan Walzer. Modern minimal perfect hashing: A survey. *CoRR*, abs/2506.06536, 2025. doi:10.48550/arXiv.2506.06536.

20 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. SicHash - small irregular cuckoo tables for perfect hashing. In *ALENEX*, pages 176–189. SIAM, 2023. doi:10.1137/1.9781611977561.CH15.

21 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. ShockHash: Near optimal-space minimal perfect hashing beyond brute-force. *CoRR*, abs/2310.14959, 2024. doi:10.48550/arXiv.2310.14959.

22 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. ShockHash: Towards optimal-space minimal perfect hashing beyond brute-force. In *ALENEX*, pages 194–206. SIAM, 2024. doi:10.1137/1.9781611977929.15.

23 Hans-Peter Lehmann, Peter Sanders, Stefan Walzer, and Jonatan Ziegler. ByteHamster/ConsensusRecSplit. Software, swhId: swh:1:dir:eeef738e12bea287eae54a77ce81241587a91767 (visited on 2025-07-16). URL: https://github.com/ByteHamster/ConsensusRecSplit, doi:10.4230/artifacts.23789.

24 Hans-Peter Lehmann, Peter Sanders, Stefan Walzer, and Jonatan Ziegler. Combined search and encoding for seeds, with an application to minimal perfect hashing. *CoRR*, abs/2502.05613, 2025. doi:10.48550/arXiv.2502.05613.

25 Harry G. Mairson. The program complexity of searching a table. In *FOCS*, pages 40–47. IEEE, 1983. doi:10.1109/SFCS.1983.76.

26 Kurt Mehlhorn. On the program size of perfect and universal hash functions. In *FOCS*, pages 170–175. IEEE, 1982. doi:10.1109/SFCS.1982.80.

27 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

28 Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008. doi:10.1137/060658400.

29 Anna Pagh, Rasmus Pagh, and Milan Ruzic. Linear probing with constant independence. In *STOC*, pages 318–327. ACM, 2007. doi:10.1145/1250790.1250839.

30 Giulio Ermanno Pibiri and Roberto Trani. PTHash: Revisiting FCH minimal perfect hashing. In *SIGIR*, pages 1339–1348. ACM, 2021. doi:10.1145/3404835.3462849.

31 Robert F. Rice. Some practical universal noiseless coding techniques. *Jet Propulsion Laboratory, JPL Publication*, 1979.

32 Ian H Witten, Alistair Moffat, and Timothy C Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

33 Wei-Pang Yang and M. W. Du. A backtracking method for constructing perfect hash functions from a set of mapping functions. *BIT*, 25(1):148–164, 1985. doi:10.1007/BF01934995.

34 Jonatan Ziegler. Compacting minimal perfect hashing using symbiotic random search. Bachelor's thesis, Karlsruhe Institute for Technology (KIT), 2025. doi:10.5445/IR/1000177814.

Faster Exponential Algorithms for Cut Problems via Geometric Data Structures

László Kozma
Institut für Informatik, Freie Universität Berlin, Germany

Junqi Tan
Institut für Informatik, Freie Universität Berlin, Germany

Abstract

For many hard computational problems, simple algorithms that run in time $2^n \cdot n^{O(1)}$ arise, say, from enumerating all subsets of a size-n set. Finding (exponentially) faster algorithms is a natural goal that has driven much of the field of *exact exponential algorithms* (e.g., see Fomin and Kratsch, 2010). In this paper we obtain algorithms with running time $O(1.9999977^n)$ on input graphs with n vertices, for the following well-studied problems:

- d-Cut: find a proper cut in which no vertex has more than d neighbors on the other side of the cut;
- Internal Partition: find a proper cut in which every vertex has at least as many neighbors on its side of the cut as on the other side; and
- (α, β)-Domination: given *intervals* $\alpha, \beta \subseteq [0, n]$, find a subset S of the vertices, so that for every vertex $v \in S$ the *number* of neighbors of v in S is from α and for every vertex $v \notin S$, the number of neighbors of v in S is from β.

Our algorithms are exceedingly simple, combining the *split and list* technique (Horowitz and Sahni, 1974; Williams, 2005) with a tool from computational geometry: *orthogonal range searching* in the moderate dimensional regime (Chan, 2017). Our technique is applicable to the decision, optimization and counting versions of these problems and easily extends to various generalizations with more fine-grained, *vertex-specific* constraints, as well as to *directed*, *balanced*, and other variants.

Algorithms with running times of the form c^n, for $c < 2$, were known for the first problem only for constant d, and for the third problem for certain special cases of α and β; for the second problem we are not aware of such results.

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

Keywords and phrases graph algorithms, cuts, exponential time, data structures

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.110

1 Introduction

For many hard computational problems, simple algorithms that run in time $2^n \cdot n^{O(1)}$ can be obtained, say, by enumerating all subsets of a size-n set. Obtaining speedups of a factor exponential in n is a natural goal that has driven much of the field of *exact exponential algorithms*, motivating the development of powerful and general algorithmic techniques.

Landmark successes include Independent Set [30], Subset Sum [18], Hamiltonian Cycle [4], Max Cut [33], 3-CNF SAT [27], to name just a few. While all NP-hard, these problems admit algorithms running in time c^n with $c < 2$, significantly faster than simple enumeration or dynamic programming over subsets.[1] We refer to the textbook of Fomin and Kratsch [16] for a broad overview of such results and techniques, as well as to the earlier, influential survey of Woeginger [34].

[1] For the problems we mention, n refers to the most natural problem-specific input parameter, e.g., the number of vertices for graph problems, the number of variables for CNF SAT, or the number of elements for Set Cover or Subset Sum.


© László Kozma and Junqi Tan;
licensed under Creative Commons License CC-BY 4.0



From a practical point of view, replacing a runtime of 2^n by (say) 1.999^n is, of course, rarely consequential, especially if the latter algorithm is more complicated. Yet, such improvements often point at problem-specific algorithmic structure that can be further exploited. Also, the search for algorithms that break such "triviality barriers" has often led to the invention of powerful algorithmic techniques that have found broader applications; examples include *branch and reduce*, *measure and conquer*, or *split and list*.

Yet, the set of techniques known to lead to such speedups is not large and several important problems seem to resist improvements beyond 2^n (Graph Coloring, CNF SAT, Set Cover, TSP are prominent examples). The hypothesis that no substantially faster algorithm exists for CNF SAT (the *SETH Conjecture* [19]) has become a cornerstone of fine-grained complexity theory, and other barriers are similarly conjectured (e.g., the *Set Cover Conjecture* [10, 21]). There are also many lesser known problems "stuck" at 2^n, where the existing techniques do not seem applicable. Problems where such a barrier was recently bypassed include, e.g., various scheduling [11, 25], bin packing [24], clustering [1, 13], multicut [23] and multiway cut [8, 35] problems, to mention just a few.

Restricted cut problems. In this paper, we mainly consider *cut* problems in graphs, with certain natural restrictions on how vertices may interact with the cut. We focus on decision problems (whether a cut of the required form exists); in all cases, an actual cut can be constructed with minimal overhead.

Our first problem is $\boxed{d\text{-Cut}}$: given an undirected graph $G = (V, E)$ and an integer d, find a bipartition $V = V_L \cup V_R$, where V_L, V_R are nonempty disjoint sets, so that each vertex $v \in V$ has at most d neighbors on the other side of the partition. Precisely, denoting by $N(v)$ the (open) set of neighbors of v, we require $|N(v) \cap V_R| \leq d$ for all $v \in V_L$ and $|N(v) \cap V_L| \leq d$ for all $v \in V_R$. The value d is arbitrary, in particular, it may depend on n.

The problem can be seen as a natural *min-max* variant of the famous Min Cut problem. While Min Cut is polynomial time solvable, d-Cut is NP-hard, even for $d = 1$ [9].

The d-Cut problem was studied by Gomes and Sau [12] as a natural generalization of the well-known Matching Cut problem (the case $d = 1$). Graphs admitting a matching cut are known as *decomposable* and have been extensively studied, e.g., see Graham [17], Chvátal [9], and Bonsma [5]. For Matching Cut, Komusiewicz, Kratsch, and Le [20] give an $O(1.3071^n)$ time algorithm, using a reduction to CNF SAT. For the general d-Cut problem, Gomes and Sau obtain a runtime of the form $O(c_d^n)$ where c_d grows at least as $(2^d - 1)^{\frac{1}{d}}$. As such, the base is smaller than 2 for *fixed* d, but cannot, in general, be bounded away from 2; the existence of an algorithm with runtime $O((2 - \varepsilon)^n)$ for $\varepsilon > 0$ was left open by Gomes and Sau. In fact, the SAT-based algorithm of Komusiewicz also applies to d-Cut, resulting in a polynomial number of $(d + 2)$-CNF SAT instances with n variables. Under the SETH conjecture [19], the running time of this approach also cannot be bounded as $O((2 - \varepsilon)^n)$.

We note that the related Max Cut problem can be solved in time $O(1.73^n)$ by reduction to fast matrix multiplication [33], this technique however, seems not to extend to d-Cut.

A second general cut problem we consider is $\boxed{(\alpha, \beta)\text{-Domination}}$. Here, we seek a bipartition $V = V_L \cup V_R$, where $|N(v) \cap V_L| \in \alpha$ for all $v \in V_L$ and $|N(v) \cap V_L| \in \beta$ for all $v \in V_R$. Note that α and β are subsets of the set $\{0, 1, \ldots, n\}$.

This formulation, introduced by Telle [31], generalizes many natural structures and problems in graphs, such as Independent Set, Dominating Set, Induced Matching, etc., by setting α and β to particular values (we refer to [31, 14] for a list of special cases). While the fully general case seems hopeless, it is perhaps most natural to require α and β to

be intervals, i.e., of the form $\alpha = [\alpha', \alpha'']$, $\beta = [\beta', \beta'']$. This already captures most, if not all of the interesting special cases. Notice, however, that the problem does not strictly generalize d-Cut; while setting $\beta = [0, d]$ enforces the degree constraint on V_R, the admissible degrees for vertices in V_L depend on their original degrees in G.

The problem was studied from the point of view of exponential algorithmics by Fomin, Golovach, Kratochvíl, Kratsch, and Liedloff [14]. They obtain algorithms with runtime c^n for $c < 2$ in the special case $|\alpha| = |\beta| = 1$, as well as in some cases where α and β are not necessarily intervals, e.g., when $|\alpha| + |\beta| = 3$ or when α and β are generated by arithmetic sequences; these latter cases fall outside of our definition. The same authors, in a different paper [15] also consider other special cases. These require some combination of α and β being disjoint, finite, or successor-free (i.e., not containing consecutive integers). As such, the conditions are not comparable with our requirement for α and β to be intervals; in our case, α and β may overlap arbitrarily, may be infinite, but cannot have gaps.

The third problem we consider is Internal Partition. Here, we seek a bipartition $V = V_L \cup V_R$, where V_L and V_R are nonempty and every vertex has at least half of its neighbors on its own side of the partition. Formally, for all $v \in V_L$ we have $|N(v) \cap V_L| \geq |N(v) \cap V_R|$, and for all $v \in V_R$ we have $|N(v) \cap V_L| \leq |N(v) \cap V_R|$.

Such a partition, called an *internal partition*, arises in many settings (in the literature the terms *friendly-*, *cohesive-*, or *satisfactory-* partition are also used). Determining whether a graph admits an internal partition is NP-hard [3]; this is in contrast to *external partitions* (where every vertex has at least half of its neighbors on the other side), which always exist. Research has mostly focused on conditions that guarantee the existence of an internal partition, e.g., see Thomassen [32], Stiebitz [29], and the survey of Bazgan, Tuza, and Vanderpooten [3]. A long-standing conjecture is that every sufficiently large r-regular graph has an internal partition; e.g., see Ban and Linial [2] and Linial and Louis [22] for recent partial results. We are not aware of algorithms with runtime of the form $O((2-\varepsilon)^n)$ for deciding the existence of an internal partition.

All three problems can trivially be solved in time $2^n \cdot n^{O(1)}$. Our main result improves this for all three problems.

▶ **Theorem 1.** *The d-Cut, (α, β)-Domination, and Internal Partition problems can be solved in time $O(1.9999977^n)$.*

While we state our results for the decision problems (does a bipartition with the given property exist?), we can also easily extend them to the *optimization* versions of the problems (maximize or minimize the left side $|V_L|$ of the cut while satisfying the constraint), and to the *counting* versions (count the number of cuts with the given property). The optimization view is most natural for (α, β)-Domination, where special cases include the task of finding e.g., a maximum independent set, a minimum dominating set, etc. (of course, for some of the well-known special cases faster algorithms already exist).

Additional constraints on the cuts (e.g., requiring V_L to be of a certain size, say $|V_L| = n/2$), or making the edges directed (and extending the conditions accordingly to in- or out-neighborhoods) can easily be accommodated, as will be clear later.

Our algorithms also extend naturally to more fine-grained, *vertex-specific* conditions. For d-Cut, this could mean setting an upper bound d_v for each vertex v on the number of neighbors across the cut, and possibly, a vertex-specific *lower bound* on the same quantity. For (α, β)-Domination, we could set specific intervals $[\alpha_v, \beta_v]$ for each vertex v; these may even depend on the degree of v in the input, or on other properties of the graph.

Overview of our approach. In spite of the unwieldy bound on the running time, our approach is conceptually exceedingly simple, and all three problems are tackled in essentially the same way.

Our starting point is the *split and list* technique originally used by Horowitz and Sahni [18] for Subset Sum: Given a set $X = \{x_1, \ldots, x_n\}$, find a subset of X whose elements add up to a given value t.

The technique works by splitting X into $X_A = \{x_1, \ldots, x_{\lfloor n/2 \rfloor}\}$ and $X_B = X \setminus X_A$. Then, the sets $\mathcal{S}_A$, resp., $\mathcal{S}_B$ of the possible sums given by subsets of X_A, resp., X_B are computed. The problem now amounts to searching, for all $x \in \mathcal{S}_A$, for a matching pair $t - x \in \mathcal{S}_B$. This can be solved in time $O(n)$ by binary search if $\mathcal{S}_B$ is sorted, which can be achieved by an $O(n2^{n/2})$ time preprocessing. An overall runtime of $O(n2^{n/2})$ results.

We can view the sorting/searching part of the algorithm as storing $\mathcal{S}_B$ in an efficient comparison-based dictionary (say, a balanced binary search tree). Our extension of the technique amounts to replacing this dictionary with a more sophisticated multidimensional search structure.

In particular, we use the *orthogonal range search* data structure of Chan [7]. Such a data structure stores a set of N vectors from $\mathbb{R}^d$. A *dominance range query* $q \in \mathbb{R}^d$ returns the number (or the list) of vectors stored that are upper bounded by q on every coordinate. Precisely, a stored vector $y = (y_1, \ldots, y_d)$ is reported for query vector $q = (q_1, \ldots, q_d)$ if $y_i \leq q_i$ for all $i \in \{1, \ldots, d\}$.

In most applications of range searching, the dimension d is assumed constant, however, Chan's data structure allows for d that is polylogarithmic in N. As we only need $d \in O(\log N)$, we state the result in this form. Note that dominance range queries can be reduced to *orthogonal range queries*, where each query is specified by an axis-parallel hypercube (box), and vice versa (by doubling the number of dimensions). In our application, we always formulate our queries as dominance range queries.

A small remark on the computational model is in order. While the data structure is defined for vectors in $\mathbb{R}^d$, assuming a real RAM with constant time operations, in our application all vector coordinates are integers in $\{-2n, \ldots, 2n\}$, our results thus hold in more realistic models such as the word RAM, or even if we account for the number of bit-operations.

The result we rely on takes the following form.

▶ **Lemma 2** (Chan [7]). *We can preprocess N vectors from $\mathbb{R}^d$ for $d \leq c \log_2 N$, and store them in a data structure that serves a sequence of N dominance range queries in a total time (including the preprocessing) $N^{2-\varepsilon_c}$, where $\varepsilon_c > 0$ depends only on c.*

We use this geometric data structure to solve the mentioned graph cut problems similarly to the split and list technique sketched above. We split the set of vertices in two equal parts, and form all possible bipartitions of both parts. Combining two such bipartitions into one requires checking whether they are "compatible", i.e., whether their union satisfies the degree constraints. This can, in all three cases, be verified with dominance range queries on appropriately preprocessed degree vectors. We precisely state and prove our results in Section 2.

In Section 3 we derive an exact constant ε_c for Lemma 2 in our setting; this is not easy to read off directly from previous work, as results are stated in asymptotic notation, e.g., $\varepsilon_c \in 1/O(\log c)$ where c is allowed to be (slightly) superconstant. In our application, as we will show, $c = 16$ and $\varepsilon_c \geq 3 \cdot 10^{-6}$ are feasible, which for $N = 2^{n/2}$ yield the stated runtime.

Chan [7] gives several different implementations that result in a bound of the form given in Lemma 2. We work out the constant in the exponent for the variant that appears the simplest [7, § 2.1]. It is a randomized, "purely combinatorial" implementation, i.e., not using

fast matrix multiplication or other algebraic tools. The data structure allows for an *online* sequence of queries. As in our setting the queries are *offline* (i.e., known upfront), we could use one of the more efficient offline, as well as deterministic versions from [7], to obtain a similar, possibly better constant; we refrain from this to keep the presentation simple.

While the use of a geometric data structure may seem surprising in a graph theoretic context (its implementation involves an intricate lopsided recursion reminiscent of k-d-trees), Chan's technique did have previous graph-applications, although mostly in the polynomial-time regime, e.g., for the all pairs shortest paths problem [7]. For a different flavor of applying orthogonal range searching (in constant dimensions) to graph algorithms, see [6].

After presenting our results (Section 2) and deriving a precise constant bound for the base of the running time (Section 3), in Section 4 we give some final thoughts.

2 Results

We use standard graph notation. A bipartition (cut) of an undirected graph $G = (V, E)$ is a partitioning (V_L, V_R) of the set of vertices, i.e., $V = V_L \cup V_R$ and $V_L \cap V_R = \emptyset$. A *proper* bipartition is one where both V_L and V_R are nonempty. We denote the open neighborhood of a vertex v by $N(v)$, i.e., $N(v) = \{u \in V : uv \in E\}$. For convenience, for a set $A \subseteq V$ we denote $N_A(v) = N(v) \cap A$.

2.1 Internal partition

We start with the Internal Partition problem whose solution is the simplest. We first formally define the problem.

Internal partition
Input: An undirected graph $G = (V, E)$.
Output: Is there a proper bipartition (V_L, V_R) of V such that
- $|N_{V_L}(v)| \geq |N_{V_R}(v)|$ for all $v \in V_L$,
- $|N_{V_R}(v)| \geq |N_{V_L}(v)|$ for all $v \in V_R$.

In words, every vertex has at least half of its neighbors in its own part.

Algorithm. We now describe our algorithm for Internal Partition. Start by choosing an arbitrary subset V_A of V of size $\lfloor n/2 \rfloor$ and let $V_B = V \setminus V_A$. For ease of notation, assume $V_A = \{v_1, \ldots, v_{\lfloor n/2 \rfloor}\}$, $V_B = \{v_{\lfloor n/2 \rfloor + 1}, \ldots, v_n\}$.

We consider every possible bipartition of V_A and every possible bipartition of V_B. Our goal is to identify a "matching pair", i.e., a bipartition (S, R) of V_A and a bipartition (S', R') of V_B so that $(S \cup S', R \cup R')$ is a valid bipartition of V, satisfying the given degree constraints.

To achieve this, we encode every proper bipartition (S, R) of V_A as a vector $q = q_S$ and every proper bipartition (S', R') as a vector $p = p_{S'}$, both of dimension $2n$. Our encoding is meant to ensure that $(S \cup S', R \cup R')$ is a feasible solution for Internal Partition if and only if $q \geq p$, i.e., vector q dominates vector p. Intuitively, p is a *data vector* stored in the data structure, and q is a *query vector*.

We define $q_S = (q_1^1, \ldots, q_n^1, q_1^2, \ldots, q_n^2)$ as follows. For all $i = 1, \ldots, n$:
- if $v_i \in S$, then $q_i^1 = |N_S(v_i)| - |N_R(v_i)|$, and $q_i^2 = n$,
- if $v_i \in R$, then $q_i^1 = n$, and $q_i^2 = |N_R(v_i)| - |N_S(v_i)|$,
- if $v_i \in V_B$, then $q_i^1 = |N_S(v_i)| - |N_R(v_i)|$, and $q_i^2 = |N_R(v_i)| - |N_S(v_i)|$.

Intuitively, the entries q_i^1 and q_i^2 store the "excess degree" of v_i to its own part. The two groups of entries correspond to the two inequalities in the definition of the problem. If $v_i \in V_A$, then its assignment to either set $S \subseteq V_L$ or $R \subseteq V_R$ is already determined, so only one of the two inequalities is relevant to v_i. Accordingly, we set one of q_i^1 and q_i^2 to the correct excess degree, while we set the other to a large value (n) that ensures dominance. If $v_i \in V_B$, then its assignment to V_L or V_R is not yet determined, and thus, we compute the excess degrees to both parts.

Symmetrically, for every proper bipartition (S', R') of V_B, we create a vector $p_{S'} = (p_1^1, \ldots, p_n^1, p_1^2, \ldots, p_n^2)$ as follows. For all $i = 1, \ldots, n$:

- if $v_i \in S'$, then $p_i^1 = |N_{R'}(v_i)| - |N_{S'}(v_i)|$, and $p_i^2 = -n$,
- if $v_i \in R'$, then $p_i^1 = -n$, and $p_i^2 = |N_{S'}(v_i)| - |N_{R'}(v_i)|$,
- if $v_i \in V_A$, then $p_i^1 = |N_{R'}(v_i)| - |N_{S'}(v_i)|$, and $p_i^2 = |N_{S'}(v_i)| - |N_{R'}(v_i)|$.

The algorithm now stores all vectors $p_{S'}$ in a data structure and queries, for each vector q_S, whether it dominates any of the stored vectors. If a dominating pair is found, then the output is yes, otherwise no.

Both the number of stored vectors and the number of queries is at most $N = 2^{\lceil \frac{n}{2} \rceil}$. The construction of the vectors clearly takes time polynomial in n per vector.

Of course, comparing each pair directly would yield no runtime improvement. However, by applying the data structure of Lemma 2, we can solve the problem in time $O((2^{1-\varepsilon_c/2})^n)$ time, where $c = 4$, since the dimension is $2n = 4 \log_2 N$.

We required (S, R) and (S', R') to be *proper* bipartitions of V_A and V_B, to avoid a possible trivial solution where one part is empty. This means that we need to check the cases $S = \emptyset$ or $R = \emptyset$ against all proper bipartitions of V_B and the cases $S' = \emptyset$ or $R' = \emptyset$ against all proper bipartitions of V_A. These special cases require time $2^{n/2} \cdot n^{O(1)}$, which is absorbed in the overall runtime.

Correctness. We must show that V admits a feasible bipartition if and only if there is some q_S that dominates some $p_{S'}$.

Suppose that (V_L, V_R) is a feasible bipartition, and let V_A, V_B be the algorithm's choice, so $S = V_A \cap V_L$ and $S' = V_B \cap V_L$, and $R = V_A \cap V_R$ and $R' = V_B \cap V_R$ are among the sets considered. The case when one of S, S', R, R' is empty is handled separately, and a solution can clearly be found. Otherwise, let q_i^k, p_i^k, for $i \in \{1, \ldots, n\}$ and $k \in \{1, 2\}$ denote the entries of q_S and $p_{S'}$, as defined above.

If $v_i \in S \subseteq V_L$, or if $v_i \in S' \subseteq V_L$, then $q_i^1 - p_i^1 = |N_S(v_i)| - |N_R(v_i)| + |N_{S'}(v_i)| - |N_{R'}(v_i)| = |N_{V_L}(v_i)| - |N_{V_R}(v_i)|$. This must be nonnegative from the feasibility condition. For the second part, if $v_i \in S \subseteq V_L$, then $q_i^2 = n$, and $p_i^2 = |N_{S'}(v_i)| - |N_{R'}(v_i)|$, which clearly yields $q_i^2 \geq p_i^2$. If $v_i \in S' \subseteq V_L$, then $q_i^2 - p_i^2 = |N_R(v_i)| - |N_S(v_i)| + n \geq 0$.

If $v_i \in R \subseteq V_R$, or if $v_i \in R' \subseteq V_R$, then $q_i^2 - p_i^2 = |N_R(v_i)| - |N_S(v_i)| - |N_{S'}(v_i)| + |N_{R'}(v_i)| = |N_{V_R}(v_i)| - |N_{V_L}(v_i)|$, again, nonnegative from the feasibility condition. For the first part, if $v_i \in R \subseteq V_R$, then $q_i^1 = n$, and $p_i^1 = |N_{R'}(v_i)| - |N_{S'}(v_i)|$, which yields $q_i^1 \geq p_i^1$. If $v_i \in R' \subseteq V_R$, then $q_i^1 - p_i^1 = |N_S(v_i)| - |N_R(v_i)| + n \geq 0$.

Conversely, if $q_S \geq p_{S'}$, then from the corresponding $q_i^1 \geq p_i^1$ we obtain $N_{S \cup S'}(v_i) \geq N_{V \setminus (S \cup S')}(v_i)$ for all $v_i \in S \cup S'$. From $q_i^2 \geq p_i^2$ we obtain $N_{V \setminus (S \cup S')}(v_i) \geq N_{S \cup S'}(v_i)$ for all $v_i \in V \setminus (S \cup S')$. This means that $(V_L, V_R) = (S \cup S', V \setminus (S \cup S'))$ is a feasible solution.

2.2 d-Cut and (α, β)-Domination

Let us formally state our other two problems.

d-Cut
Input: An undirected graph $G = (V, E)$ and a nonnegative integer d.
Output: Is there a proper bipartition (V_L, V_R) of V such that
- $|N_{V_L}(v)| \leq d$ for all $v \in V_R$,
- $|N_{V_R}(v)| \leq d$ for all $v \in V_L$.

In words, every vertex is incident to at most d edges that cross (V_L, V_R).

(α, β)-Domination
Input: An undirected graph $G = (V, E)$ and intervals $\alpha, \beta \subseteq [0, n]$.
Output: Is there a proper bipartition (V_L, V_R) of V such that
- $|N_{V_L}(v)| \in \alpha$ for all $v \in V_L$,
- $|N_{V_L}(v)| \in \beta$ for all $v \in V_R$.

We next introduce a general problem formulation that allows for vertex-specific constraints and that extends both problems.

Interval-Constrained Cut
Input: An undirected graph $G = (V, E)$ and nonnegative integers $a'_v, a''_v, b'_v, b''_v, c'_v, c''_v, d'_v, d''_v$ for all v.
Output: Is there a proper bipartition (V_L, V_R) of V such that
- $|N_{V_L}(v)| \in [a'_v, a''_v]$ for all $v \in V_L$,
- $|N_{V_R}(v)| \in [b'_v, b''_v]$ for all $v \in V_L$,
- $|N_{V_R}(v)| \in [c'_v, c''_v]$ for all $v \in V_R$,
- $|N_{V_L}(v)| \in [d'_v, d''_v]$ for all $v \in V_R$.

The problem generalizes d-Cut, by setting $[a'_v, a''_v] = [c'_v, c''_v] = [0, n]$, i.e., "don't care", and setting $[b'_v, b''_v] = [d'_v, d''_v] = [0, d]$, i.e., the required degree constraint.

The problem also generalizes (α, β)-Domination, for intervals $\alpha = [\alpha', \alpha'']$ and $\beta = [\beta', \beta'']$, by setting $[b'_v, b''_v] = [c'_v, c''_v] = [0, n]$, i.e., "don't care", and setting $[a'_v, a''_v] = [\alpha', \alpha'']$ and $[d'_v, d''_v] = [\beta', \beta'']$, i.e., the required degree constraint.

Algorithm. We now describe our solution for Interval-Constrained Cut, which is very similar to the earlier one (only slightly more complicated, since the problem definition involves 8 inequalities instead of 2).

Again, we choose an arbitrary subset V_A of V of size $\lfloor n/2 \rfloor$ and let $V_B = V \setminus V_A$, denoting $V_A = \{v_1, \ldots, v_{\lfloor n/2 \rfloor}\}$, $V_B = \{v_{\lfloor n/2 \rfloor + 1}, \ldots, v_n\}$.

We encode every proper bipartition (S, R) of V_A as a vector q_S and every proper bipartition (S', R') as a vector $p_{S'}$, where both vectors have dimension $8n$. (Bipartitions where one part is empty are handled separately, in the same way as in Section 2.1.) Additionally, we encode the problem constraints as a vector r, also of dimension $8n$. The encoding ensures that q_S dominates $p_{S'} + r$ if and only if $(S \cup S', V \setminus (S \cup S'))$ is a feasible solution.

The entries of vectors $q = q_S$, $p = p_{S'}$, r are indexed as q_i^k, p_i^k, r_i^k for $i \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, 8\}$, ordered lexicographically according to the index (k, i).

Intuitively, the entries form 8 groups, corresponding to the 8 inequalities of the interval constraints in the problem statement, and $q_i^k \geq p_i^k + r_i^k$ verifies whether the k-th inequality holds for vertex v_i.

The encoding is as follows. We start with r, which is the simplest to state. The entries r_i^k are, for all $i = 1, \ldots, n$:

r_i^1	r_i^2	r_i^3	r_i^4	r_i^5	r_i^6	r_i^7	r_i^8
a'_i	$-a''_i$	b'_i	$-b''_i$	c'_i	$-c''_i$	d'_i	$-d''_i$

We next describe the entries q_i^k of q_S and the entries p_i^k of $p_{S'}$. Recall that (S, R) is a bipartition of V_A and (S', R') is a bipartition of V_B. For all $i = 1, \ldots, n$:

$v_i \in$	q_i^1	q_i^2	q_i^3	q_i^4	q_i^5	q_i^6	q_i^7	q_i^8
S	$\lvert N_S(v_i)\rvert$	$-\lvert N_S(v_i)\rvert$	$\lvert N_R(v_i)\rvert$	$-\lvert N_R(v_i)\rvert$	$2n$	$2n$	$2n$	$2n$
R	$2n$	$2n$	$2n$	$2n$	$\lvert N_R(v_i)\rvert$	$-\lvert N_R(v_i)\rvert$	$\lvert N_S(v_i)\rvert$	$-\lvert N_S(v_i)\rvert$
V_B	$\lvert N_S(v_i)\rvert$	$-\lvert N_S(v_i)\rvert$	$\lvert N_R(v_i)\rvert$	$-\lvert N_R(v_i)\rvert$	$\lvert N_R(v_i)\rvert$	$-\lvert N_R(v_i)\rvert$	$\lvert N_S(v_i)\rvert$	$-\lvert N_S(v_i)\rvert$

The three rows of the table correspond to the cases where v_i is in S, R, or V_B, showing the value to which q_i^k is set. The 8 columns distinguish the entries needed for verifying the 8 inequalities (i.e., the index k). Notice that if $v_i \in S$ or $v_i \in R$, then the placement of v_i in V_L or V_R is determined and only 4 of the 8 inequalities in the definition are applicable to v_i. For these cases we set q_i^k to the relevant degree of v_i (with plus sign where we have a lower bound, and minus sign where we have an upper bound). For the inequalities that do not apply to v_i, we set q_i^k to a sufficiently large value of $2n$ which ensures that on the given (k, i) coordinate q dominates $p + r$, as vertex v_i trivially satisfies the condition k.

On the other hand, if $v_i \in V_B$, then the placement of v_i within the cut (V_L, V_R) is not yet determined, in this case we account for all degrees, preparing for both cases.

The encoding of $p = p_{S'}$ is symmetric. For all $i = 1, \ldots, n$:

$v_i \in$	p_i^1	p_i^2	p_i^3	p_i^4	p_i^5	p_i^6	p_i^7	p_i^8
S'	$-\lvert N_{S'}(v_i)\rvert$	$\lvert N_{S'}(v_i)\rvert$	$-\lvert N_{R'}(v_i)\rvert$	$\lvert N_{R'}(v_i)\rvert$	$-2n$	$-2n$	$-2n$	$-2n$
R'	$-2n$	$-2n$	$-2n$	$-2n$	$-\lvert N_{R'}(v_i)\rvert$	$\lvert N_{R'}(v_i)\rvert$	$-\lvert N_{S'}(v_i)\rvert$	$\lvert N_{S'}(v_i)\rvert$
V_A	$-\lvert N_{S'}(v_i)\rvert$	$\lvert N_{S'}(v_i)\rvert$	$-\lvert N_{R'}(v_i)\rvert$	$\lvert N_{R'}(v_i)\rvert$	$-\lvert N_{R'}(v_i)\rvert$	$\lvert N_{R'}(v_i)\rvert$	$-\lvert N_{S'}(v_i)\rvert$	$\lvert N_{S'}(v_i)\rvert$

We then store all vectors $p_{S'} + r$ in the data structure and execute orthogonal dominance queries for all q_S vectors. A running time of $O((2^{1-\varepsilon_c/2})^n)$ follows similarly to the previous section, with $c = 16$, since the dimension is $8n = 16 \log_2 N$.

Correctness. We need to verify that there is a feasible solution to the Interval-Constrained Cut problem if and only if there is a q_S vector that dominates some $p_{S'} + r$ vector.

Suppose first that a feasible solution (V_L, V_R) exists and let $S = V_A \cap V_L$ and $S' = V_B \cap V_L$. We show that $q_S \geq p_{S'} + r$, so the algorithm correctly finds the solution.

We verify this dominance relation on the entries q_i^1, p_i^1, r_i^1; the calculations for the cases $k = 2, \ldots, 8$ are entirely analogous and thus omitted.

If $v_i \in R$ or $v_i \in R'$ then either $q_i^1 = 2n$ or $p_i^1 = -2n$, and in both cases $q_i^1 \geq p_i^1 + r_i^1$, since $r_i^1 = a'_i \leq n$. Thus, the inequality holds.

If $v_i \in S$ or if $v_i \in S'$, then $q_i^1 = |N_S(v_i)|$ and $p_i^1 = -|N_{S'}(v_i)|$. Thus $q_i^1 \geq p_i^1 + r_i^1$ is equivalent to $|N_S(v_i)| + |N_{S'}(v_i)| \geq a'_i$. Since the left side equals $|N_{V_L}(v_i)|$, the inequality follows from the feasibility of (V_L, V_R).

Conversely, suppose that the algorithm identifies vectors q_S and $p_{S'}$ for which $q_S \geq p_{S'} + r$ holds. In particular, if $v_i \in S \cup S'$, then $q_i^1 \geq p_i^1 + r_i^1$, which amounts to $|N_S(v_i)| + |N_{S'}(v_i)| \geq a'_i$, yielding $|N_{S \cup S'}(v_i)| \geq a'_i$. This means that a partition (V_L, V_R) with $V_L = S \cup S'$ satisfies

the first inequality of the problem statement. If $v_i \notin S \cup S'$, then the first inequality does not apply to v_i, and thus trivially holds. The remaining inequalities follow analogously from the cases $k = 2, \ldots, 8$. Thus, a feasible solution $(S \cup S', V \setminus (S \cup S'))$ exists.

Remarks. If only the d-Cut problem is being solved, then one can encode the partitions with only 2 entries per vertex instead of 8, similarly to Internal Partition, yielding a (small) saving in runtime. An additional optimization that we omitted is to notice when some partial cuts (S, R) and (S', R') already violate the degree upper bounds; in such cases the corresponding vectors need not be stored, resp., queried.

2.3 Extensions

So far we have described how to solve the decision problem, i.e., deciding whether a cut with the given constraints exists. Constructing an actual feasible cut is easy, as the data structure can return for a query vector q_S a single data vector $p_{S'}$ that has led to the positive answer without any overhead. With minor bookkeeping, then S and S' are identified, and the bipartition $(S \cup S', V \setminus (S \cup S'))$ is constructed.

Chan's data structure returns the *number* of data vectors dominated by the query vector. Adding up all these counts yields the total number of solutions without overcounting (notice that every solution is uniquely identified by its intersection with the partition (V_A, V_B) chosen by the algorithm). Note that we must also add the solutions found in the special cases when one of S, S', R, R' is empty.

Finally, if we only look for solutions of a given size, say $|V_L| = t$, then we can encode the size of partial cuts ($|S|$ and $|S'|$) and use two additional dimensions to enforce the inequalities $|S| + |S'| \leq t$ and $|S| + |S'| \geq t$. This does not affect the stated runtimes. If we wish to minimize or maximize the size $|V_L| = t$ for which a solution exists, then we can repeat the procedure for $t = 1, \ldots, n-1$ and choose the optimum, with a factor n overhead, which is negligible in our regime.

3 Obtaining a base constant

In this section, for concreteness, we derive a constant value for the base of the running time, without attempting to significantly optimize it.

This results from the analysis of the dominance range searching data structure stated in Lemma 2. We refer to the *online* implementation by Chan described and analyzed in [7, § 2.1]. While the online aspect of the data structure is not strictly needed in our application, the implementation is relatively simple (using a recursive k-d-tree-like approach) and in particular it does not rely on algebraic techniques. While Chan also describes deterministic, offline designs with stronger asymptotic guarantees, they rely on more complicated techniques, and exact constants are more difficult to derive for them (although their running times are also clearly of the form given in Lemma 2).

Chan upper bounds the cost of a query as $e^{\gamma d} \cdot N^{1-\varepsilon} \cdot (bd \log N)^{O(1)}$. The parameter b is set as $b = (c/\delta) \log (c/\delta)$, where $\delta \in (0, 1)$ is a user-chosen parameter, and log is the base 2 logarithm.

Recall that in our application $N = 2^{n/2}$ and $d = c \log N$, where $c = 16$. Thus, the last factor $(bd \log N)^{O(1)}$ can be upper bounded by a polynomial in n, and thus omitted (polynomial factors are absorbed, if we slightly increase the base of the exponential).

As for the remaining parameters, Chan sets in the analysis $\gamma = 8b\alpha$, where $\alpha = \frac{1}{b^4}$, and finally, $\varepsilon = \frac{1}{6b \log \frac{1}{\alpha}} = \frac{1}{24b \log b}$.

Thus, (after ignoring a polynomial factor, as mentioned), the cost of N queries is at most:

$$N^{2-\varepsilon} \cdot e^{\frac{8}{b^3} c \log N} = N^{2-\frac{1}{24b \log b}} \cdot e^{\frac{4cn}{b^3}}.$$

We now plug in $c = 16$ and $N = 2^{n/2}$ and choose $\delta = 0.1$, from which $1171 < b < 1172$ follows.

For the first factor we get an upper bound of 1.999997583^n and for the second factor an upper bound of 1.00000003986^n, making their product less than 1.9999977^n.

It remains to bound the preprocessing cost, i.e., the time needed to build the data structure that stores N vectors. This is given by Chan [7, § 2.1] as $N^{1+O(\delta)}$ multiplied by a factor which resolves to the form $n^{O(\log n)}$. While this is a *quasi-polynomial* factor, it can still be absorbed by the exponential by a slight rounding up of the exponential base, we thus omit this factor as well.

The crucial part is the constant hidden in the term $O(\delta)$ in the exponent. One can observe in the analysis [7, § 2.1] that the term $N^{1+O(\delta)}$ arises as $N \cdot b^{O(d/b)}$. The $O()$-notation is used to suppress constant factors in two steps of the analysis. We will make these explicit, stating the bound as $N \cdot b^{\alpha_1 \cdot \alpha_2 \cdot (d/b)}$ for constants $\alpha_1, \alpha_2 > 1$ (our notation).

For our earlier choices of N, b, d, the quantity is upper bounded as $2^{n/2} \cdot 1.0495^{n \cdot \alpha_1 \cdot \alpha_2}$. It remains to resolve the constants α_1, α_2.

One of them, α_1, captures a factor of $\sum_i 1/b^i$, absorbed in the $O()$-notation in one step of the analysis. For our choice of b, this geometric sum can be upper bounded as $\alpha_1 \leq \frac{b}{b-1} < 1.00085$. The other constant, α_2, arises from upper bounding $\binom{d}{\lfloor d/b \rfloor}$ as $b^{O(d/b)}$. We replace this estimate by the explicit upper bound $b^{\alpha_2 \cdot (d/b)}$. This follows from the well-known inequality $\binom{n}{k} \leq (\frac{en}{k})^k$, and since $b > e$, the constant $\alpha_2 = 2$ suffices for the inequality to hold.

Putting things together, we obtain a runtime of at most $2^{n/2} \cdot 1.0495^{n \cdot \alpha_1 \cdot \alpha_2} < 1.558^n$, which is within our stated bounds.

4 Conclusion

In this paper we gave an algorithm that solves a family of constrained cut problems, combining the split and list technique with a sophisticated geometric data structure that allows orthogonal range queries on degree vectors. Our method can also solve, with minor adaptation, the following vector-based generalization of Subset Sum in similar $(2-\varepsilon)^n$ running time: given a set of n vectors of dimension linear in n, find a subset of vectors whose sum is within a target box t.

Our overall approach is conceptually very simple. One may complain that this simplicity is illusory, since the complexity is all hidden inside the data structure; to which we would respond: *precisely, that is the point of the method.* In all fairness, one could attempt to "unroll" the data structure implementation to see the algorithm in more elementary steps. We instead propose a different question: can we replace the orthogonal range search structure by different kinds of data structures, to extend *split and list* to further algorithmic problems?

The space usage of Chan's data structure is, with careful optimization, near-linear, i.e., of the form $N^{1+\delta}$ where $N = 2^{n/2}$ and $\delta > 0$. In case of Subset Sum, an original space usage of $2^{n/2}$ has subsequently been reduced [28] to $2^{n/4}$ (with a recent small further improvement [26]). Can similar space improvements, or a finer time-space-tradeoff (again, similarly to Subset Sum) be obtained for the problems considered in this paper?

References

1 Enver Aman, Karthik C. S., and Sharath Punna. On connections between k-coloring and euclidean k-means. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.9.

2 Amir Ban and Nati Linial. Internal partitions of regular graphs. *J. Graph Theory*, 83(1):5–18, 2016. doi:10.1002/JGT.21909.

3 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Satisfactory graph partition, variants, and generalizations. *Eur. J. Oper. Res.*, 206(2):271–280, 2010. doi:10.1016/J.EJOR.2009.10.019.

4 Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. doi:10.1137/110839229.

5 Paul S. Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. *J. Graph Theory*, 62(2):109–126, 2009. doi:10.1002/JGT.20390.

6 Sergio Cabello and Christian Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Comput. Geom.*, 42(9):815–824, 2009. doi:10.1016/J.COMGEO.2009.02.001.

7 Timothy M Chan. Orthogonal range searching in moderate dimensions: kd trees and range trees strike back. *Discrete & Computational Geometry*, 61:899–922, 2019. doi:10.1007/S00454-019-00062-5.

8 Rajesh Chitnis, Fedor V. Fomin, Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, and Saket Saurabh. Faster exact algorithms for some terminal set problems. *J. Comput. Syst. Sci.*, 88:195–207, 2017. doi:10.1016/J.JCSS.2017.04.003.

9 Vasek Chvátal. Recognizing decomposable graphs. *J. Graph Theory*, 8(1):51–53, 1984. doi:10.1002/JGT.3190080106.

10 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016. doi:10.1145/2925416.

11 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Scheduling partially ordered jobs faster than 2^n. *Algorithmica*, 68(3):692–714, 2014. doi:10.1007/S00453-012-9694-7.

12 Guilherme de C. M. Gomes and Ignasi Sau. Finding cuts of bounded degree: Complexity, FPT and exact algorithms, and kernelization. *Algorithmica*, 83(6):1677–1706, 2021. doi:10.1007/S00453-021-00798-8.

13 Fedor V. Fomin, Petr A. Golovach, Tanmay Inamdar, Nidhi Purohit, and Saket Saurabh. Exact exponential algorithms for clustering problems. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.IPEC.2022.13.

14 Fedor V. Fomin, Petr A. Golovach, Jan Kratochvíl, Dieter Kratsch, and Mathieu Liedloff. Sort and search: Exact algorithms for generalized domination. *Inf. Process. Lett.*, 109(14):795–798, 2009. doi:10.1016/J.IPL.2009.03.023.

15 Fedor V. Fomin, Petr A. Golovach, Jan Kratochvíl, Dieter Kratsch, and Mathieu Liedloff. Branch and recharge: Exact algorithms for generalized domination. *Algorithmica*, 61(2):252–273, 2011. doi:10.1007/S00453-010-9418-9.

16 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. doi:10.1007/978-3-642-16533-7.

17 Ron L. Graham. On primitive graphs and optimal vertex assignments. *Annals of the New York academy of sciences*, 175(1):170–186, 1970.

18 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823.

19 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.

20 Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Matching cut: Kernelization, single-exponential time fpt, and exact exponential algorithms. *Discret. Appl. Math.*, 283:44–58, 2020. doi:10.1016/J.DAM.2019.12.010.

21 Robert Krauthgamer and Ohad Trabelsi. The set cover conjecture and subgraph isomorphism with a tree pattern. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 45:1–45:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.STACS.2019.45.

22 Nathan Linial and Sria Louis. Asymptotically almost every 2 r-regular graph has an internal partition. *Graphs and Combinatorics*, 36(1):41–50, 2020. doi:10.1007/S00373-019-02116-0.

23 Daniel Lokshtanov, Saket Saurabh, and Ondrej Suchý. Solving multicut faster than 2 n. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms – ESA 2014 – 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 666–676. Springer, 2014. doi:10.1007/978-3-662-44777-2_55.

24 Jesper Nederlof, Jakub Pawlewicz, Céline M. F. Swennenhuis, and Karol Wegrzycki. A faster exponential time algorithm for bin packing with a constant number of bins via additive combinatorics. *SIAM J. Comput.*, 52(6):1369–1412, 2023. doi:10.1137/22M1478112.

25 Jesper Nederlof, Céline M. F. Swennenhuis, and Karol Wegrzycki. A subexponential time algorithm for makespan scheduling of unit jobs with precedence constraints. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 535–552. SIAM, 2025. doi:10.1137/1.9781611978322.16.

26 Jesper Nederlof and Karol Wegrzycki. Improving schroeppel and shamir's algorithm for subset sum via orthogonal vectors. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1670–1683. ACM, 2021. doi:10.1145/3406325.3451024.

27 Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 410–414. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814612.

28 Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981. doi:10.1137/0210033.

29 Michael Stiebitz. Decomposing graphs under degree constraints. *J. Graph Theory*, 23(3):321–324, 1996. doi:10.1002/(SICI)1097-0118(199611)23:3\%3C321::AID-JGT12\%3E3.0.CO;2-H.

30 Robert Endre Tarjan and Anthony E. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6(3):537–546, 1977. doi:10.1137/0206038.

31 Jan Arne Telle. Complexity of domination-type problems in graphs. *Nord. J. Comput.*, 1(1):157–171, 1994.

32 Carsten Thomassen. Graph decomposition with constraints on the connectivity and minimum degree. *J. Graph Theory*, 7(2):165–167, 1983. doi:10.1002/JGT.3190070204.

33 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/J.TCS.2005.09.023.

34 Gerhard J. Woeginger. Open problems around exact algorithms. *Discret. Appl. Math.*, 156(3):397–405, 2008. doi:10.1016/J.DAM.2007.03.023.

35 Mingyu Xiao. Solving directed multiway cut faster than 2^n. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 104:1–104:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.104.

Hardness of Median and Center in the Ulam Metric

Nick Fischer
INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria

Elazar Goldenberg
The Academic College of Tel Aviv-Yaffo, Israel

Mursalin Habib
Rutgers University, Piscataway, NJ, USA

Karthik C. S.
Rutgers University, Piscataway, NJ, USA

Abstract

The classical *rank aggregation* problem seeks to combine a set X of n permutations into a single representative "consensus" permutation. In this paper, we investigate two fundamental rank aggregation tasks under the well-studied *Ulam metric*: computing a *median* permutation (which minimizes the sum of Ulam distances to X) and computing a *center* permutation (which minimizes the maximum Ulam distance to X) in two settings.

Continuous Setting: In the continuous setting, the median/center is allowed to be any permutation. It is known that computing a center in the Ulam metric is NP-hard and we add to this by showing that computing a median is NP-hard as well via a simple reduction from the Max-Cut problem. While this result may not be unexpected, it had remained elusive until now and confirms a speculation by Chakraborty, Das, and Krauthgamer [SODA '21].

Discrete Setting: In the discrete setting, the median/center must be a permutation from the input set. We fully resolve the fine-grained complexity of the discrete median and discrete center problems under the Ulam metric, proving that the naive $\widetilde{O}(n^2L)$-time algorithm (where L is the length of the permutation) is conditionally optimal. This resolves an open problem raised by Abboud, Bateni, Cohen-Addad, Karthik C. S., and Seddighin [APPROX '23]. Our reductions are inspired by the known fine-grained lower bounds for similarity measures, but we face and overcome several new highly technical challenges.

2012 ACM Subject Classification Mathematics of computing → Permutations and combinations; Mathematics of computing → Combinatorial optimization; Theory of computation → Problems, reductions and completeness

Keywords and phrases Ulam distance, median, center, rank aggregation, fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.111

Related Version *Full Version*: `https://arxiv.org/abs/2504.16437` [25]

Funding Part of this work was done while Mursalin Habib and Karthik C. S. were visiting INSAIT, Sofia University "St. Kliment Ohridski". Nick Fischer, Mursalin Habib, and Karthik C. S. were partially funded by the Ministry of Education and Science of Bulgaria's support for INSAIT as part of the Bulgarian National Roadmap for Research Infrastructure.
Mursalin Habib: Supported by the National Science Foundation under Grants CCF-2313372, CCF-2422558, and CCF-2443697.
Karthik C. S.: Supported by the National Science Foundation under Grants CCF-2313372, CCF-2422558, and CCF-2443697.

Acknowledgements The authors would like to thank the *DIMACS Workshop on Efficient Algorithms for High Dimensional Metrics: New Tools* for a special collaboration opportunity as well as Siam Habib and Antti Roeyskoe for helpful discussions.


© Nick Fischer, Elazar Goldenberg, Mursalin Habib, and Karthik C. S.;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Suppose that n judges each rank the performances of L competitors. Given these rankings, how can the judges agree on a single consensus ranking? This fundamental question lies at the heart of a class of tasks known as *rank aggregation*, which has applications across various fields, including social choice theory [11], bioinformatics [37], information retrieval [29], machine learning [38], and recommendation systems [41], among others. Formally, the judges' rankings can be represented as a set of n permutations $X \subseteq \mathcal{S}_L$. Then, for an appropriate metric $d(\cdot,\cdot)$ on the space of permutations $\mathcal{S}_L$, the two most prominent rank aggregation tasks are to compute a *median* permutation π_M which minimizes $\sum_{\pi\in X} d(\pi_M,\pi)$ [33, 49, 50, 23], or a *center* permutation π_C which minimizes $\max_{\pi\in X} d(\pi_C,\pi)$ [8, 10, 43].

Among the metrics studied in this context, two stand out. The first one is the classic *Kendall's tau distance* which measures the number of disagreeing pairs between two permutations, i.e., the number of pairs (i,j) for which one ranking orders i before j while the other orders j before i. Kendall's tau distance is well-motivated as it satisfies several desirable properties beyond the scope of this paper (e.g., neutrality, consistency, and the extended Condorcet property [33, 47]), and is also well-understood from a computational point-of-view [23, 24]. For example, it is known that computing the median or center of just *four* permutations is already NP-hard [23, 10]. Several approximation algorithms have also been proposed for this metric [5, 46], culminating in a PTAS [34, 42] for approximating the median under Kendall's tau metric.

The other key metric is the *Ulam distance* which measures the minimum number of relocation operations required to turn one permutation π into another permutation π' – i.e., the minimum number of competitors whose ranks have to be adjusted in π so that it agrees with π'. This metric offers a simpler and more practical alternative to Kendall's tau metric for rank aggregation tasks [20, 16, 19, 18]. Perhaps more importantly, the Ulam metric is intimately linked to the more general *edit metric* on arbitrary strings, which enjoys countless applications in computational biology [28, 42], specifically in the context of DNA storage systems [27, 44], and beyond [35, 39]. Despite the significance of Ulam rank aggregation problems and the extensive research dedicated to them [8, 16, 19, 18], some basic questions remain unanswered. This is the starting point of our paper.

1.1 Question 1: Polynomial-Time Algorithms for Ulam Median?

The first basic question is whether polynomial-time algorithms exist for exactly computing the center and median permutations under the Ulam metric. For almost all string metrics, including the aforementioned Kendall's tau metric but also metrics beyond permutations such as the Hamming metric [26, 36, 4] or the edit metric [22, 40], median and center problems are well-known and easily-proven to be NP-hard.[1] Quite surprisingly, while it is known that computing an Ulam center is NP-hard [8], the complexity of computing an Ulam median has remained an open question. This is not due to a lack of interest – despite the absence of an NP-hardness proof, Chakraborty, Das, and Krauthgamer have already initiated the study of approximation algorithms for the Ulam median problem [16, 17], achieving a 1.999-factor approximation in polynomial time. Our first contribution is that we finally provide this missing hardness proof:

▶ **Theorem 1.** *The median problem is NP-hard in the Ulam metric.*

[1] A notable exception is the Hamming median problem that can trivially be solved in polynomial time by a coordinate-wise plurality vote.

1.2 Question 2: Fine-Grained Complexity of Discrete Ulam Center and Median?

How can we circumvent this new lower bound? There are two typical approaches. The first is to resort to approximation algorithms as was done in [16, 17]. But there is a commonly-studied second option for aggregation and clustering type problems: to restrict the solution space to *only the input set of permutations* X and compute the best median or center from it.[2] The best median or center from the input set X is typically referred to as the *discrete median* or the *discrete center*, respectively, of X. (In the same spirit, we will occasionally refer to the unrestricted median and center, discussed in the previous subsection, as the *continuous median* and *continuous center*, respectively.) Besides being a natural polynomial-time rank aggregation task, computing discrete medians or centers has two other motivations. First, it is easy to see that computing a discrete median/center yields a 2-approximation of the (continuous) median/center problems. In fact, a key observation in [16] is that a discrete median often provides a $(2-\varepsilon)$-approximation for the (continuous) median, particularly in practical DNA storage system instances where distances tend to be large. Second, the discrete median and center problems have gained significant attraction for the easier Hamming metric [2, 4] and harder edit metric [2], often leading to matching upper and lower bounds. Studying the Ulam metric therefore serves as an interesting intermediate problem capturing some – but not all – of the hardness of the edit metric.

Driven by these motivations, we study the *fine-grained complexity* of the discrete median and discrete center problems with respect to the Ulam distance. That is, we aim to pinpoint their *precise* polynomial run times.

Discrete Ulam Center

The trivial algorithm for computing a discrete center is to explicitly compute the Ulam distance $d_U(\pi, \pi')$ for all pairs of permutations $\pi, \pi' \in X$. We can then easily select the permutation $\pi_C \in X$ minimizing $\max_{\pi \in X} d_U(\pi_C, \pi)$. As the Ulam distance between two length-L permutations can be computed in near-linear time[3] $\widetilde{O}(L)$ using the well-known patience sorting algorithm [6], the total time is $\widetilde{O}(n^2 L)$.

We prove that this simple algorithm is *optimal*, up to subpolynomial factors and conditioned on a plausible assumption from fine-grained complexity:

▶ **Theorem 2.** *Let $\varepsilon > 0$ and $\alpha > 0$. There is no algorithm running in time $O((n^2 L)^{1-\varepsilon})$ that solves the discrete center problem in the Ulam metric for n permutations of length $L = \Theta(n^\alpha)$, unless the Quantified Strong Exponential Time Hypothesis fails.*

The Quantified Strong Exponential Time Hypothesis (QSETH) is a plausible generalization of the by-now well-established Strong Exponential Time Hypothesis (SETH), postulating that the CNF-SAT problem cannot be solved faster than brute-force search even when only some variables are existentially quantified and others are universally quantified (see Section 3.1 for the formal treatment) [12]. This hypothesis has already proven useful for conditional lower bounds for a wide array of problems [12, 3, 2]. Besides, we remark that it is impossible to obtain SETH-based lower bounds for the discrete center problem unless the *Nondeterministic* Strong Exponential Time Hypothesis [15] is false (see Section 5.1).

[2] Or alternatively, the best median/center among the permutations in another given set $Y \subseteq \mathcal{S}_L$; this is typically referred to as the *bichromatic* discrete center/median problem, or as center/median problem with *facilities* in the theory of clustering. All of our results also apply to these bichromatic variants.

[3] We write $\widetilde{O}(T) = T(\log T)^{O(1)}$ to suppress polylogarithmic factors.

We emphasize that these lower bounds applies to the full range of n versus L, as long as L is at least polynomial in n. In the case when L is very small, $\omega(\log n) < L < n^{o(1)}$, previous work by Abboud, Bateni, Cohen-Addad, Karthik C. S. and Seddighin already established a matching conditional lower bound of $n^{2-o(1)}$ [2].

It is interesting to compare Theorem 2 with the state of the art for discrete center problems in the Hamming metric (say, over a constant-size alphabet) and the edit metric. For concreteness, consider the case $L = \Theta(n)$ (i.e., the input consists of n strings/permutations of length roughly n). Then, on the one hand, the discrete center problem in the Hamming metric can be solved in time $O(n^\omega)$ [2, 4], where $\omega < 2.3714$ is the exponent of matrix multiplication [7]. On the other hand, the discrete center problem in the edit metric cannot be solved in time $n^{4-\Omega(1)}$ unless QSETH fails [2]. Therefore, remarkably, Theorem 2 indeed places the discrete center problem in the Ulam metric as a problem of *intermediate* complexity $n^{3\pm o(1)}$. This answers an explicit open question posed by Abboud, Bateni, Cohen-Addad, Karthik C. S., and Seddighin [2].

Discrete Ulam Median

The trivial algorithm for the discrete median problem is exactly the same as for the center problem: First compute all pairwise distances $d_U(\pi, \pi')$, then select the permutation π_M minimizing $\sum_{\pi \in X} d_U(\pi_M, \pi)$. It also runs in time $\widetilde{O}(n^2 L)$. So perhaps one could also hope that a matching lower bound follows from our Theorem 2. Unfortunately, this turns out to be true only for a restricted subproblem.[4] Nevertheless, with considerable technical overhead we manage to prove essentially the same *matching* lower bound:

▶ **Theorem 3.** *Let $\varepsilon > 0$ and $\alpha \geq 1$. There is no algorithm running in time $O((n^2 L)^{1-\varepsilon})$ that solves the discrete median in the Ulam metric for n permutations of length $L = \Theta(n^\alpha)$, unless the SETH fails.*

In comparison to Theorem 2, Theorem 3 has the advantage that it conditions on the *weaker* assumption SETH (thus constituting a *stronger* lower bound). However, the applicable range of parameters is more restricted ($\alpha \geq 1$ forces the permutations to have length $\Omega(n)$).

2 Proof Overview

In this section, we provide the proof overviews of Theorems 1, 2, and 3. We avoid some technicalities to highlight the core ideas – for instance, some subpolynomial factors have been dropped. Complete proofs can be found in Sections 4, 5, and 6, respectively.

2.1 NP-Hardness for Continuous Median in the Ulam Metric

In this section, we provide a high-level sketch of our NP-hardness proof for the continuous median problem in the Ulam metric. Our starting point is the Max-Cut problem[5]. Given a Max-Cut instance $G = (V, E)$ where $V = [n]$, our goal is to construct a set of permutations of length $O(n)$ such that the median of these permutations encodes the cut in G of maximum size.

[4] Namely, the *bichromatic* discrete median problem mentioned in Footnote 2.

[5] Recall that the Max-Cut problem is, given an undirected graph $G = (V, E)$ to compute a vertex partition $V = A \sqcup B$ maximizing the number of edges from A to B.

To achieve this, we first set up a natural correspondence between cuts in G and permutations of length $O(n)$. However, since there are many more permutations than cuts, not all permutations will represent valid cuts. To ensure that only relevant permutations are considered, we construct two special permutations:

$$\pi^L = 1 \circ 2 \circ \cdots \circ (n-1) \circ n \circ X_1 \circ X_2,$$
$$\pi^R = X_1 \circ n \circ (n-1) \circ \cdots \circ 2 \circ 1 \circ X_2,$$

for some fixed long permutations X_1, X_2. The simple key insight is that any median of π^L and π^R has the following form: it starts with some subset $A \subseteq [n]$ of the symbols in *increasing* order, followed by X_1, followed by the symbols in $[n] \setminus A$ in *decreasing* order, finally followed by X_2. Thus, medians of π^L, π^R will naturally encode cuts of the form $(A, [n] \setminus A)$ in G.

To further enforce that the median represents a *maximum* cut, we include additional permutations in our instance. Specifically, for each edge $e \in E$, we include two permutations, π_e^1 and π_e^2, which reward picking solutions that correspond to partitions that cut e. This ensures that the final median permutation encodes a maximum cut of G. The precise construction and formal analysis of these edge-cutting permutations π_e^1, π_e^2 is detailed in Section 4.

2.2 Fine-Grained Lower Bound for Discrete Center in the Ulam Metric

Our proof of the lower bound for the discrete center problem in the Ulam metric relies on two key components. The first is a reduction from the Orthogonal Vectors (OV) problem to the problem of computing the Ulam distance between two permutations. Specifically, we seek a pair of functions that, given two sets of binary vectors as inputs, independently output two permutations whose Ulam distance is small if and only if there exists an orthogonal pair of vectors in the input sets. This falls into a well-established framework in the fine-grained complexity literature [9, 13, 1]: Given two sets of roughly L binary vectors, one constructs "coordinate gadgets", "vector gadgets", and "OR-gadgets" to produce a pair of length-L strings whose edit distance encodes the existence of an orthogonal pair.

Clearly such a reduction cannot exist for the Ulam distance under SETH. The Ulam distance between two permutations can be computed in near-linear time by a simple reduction to the longest increasing subsequence problem [45], and thus, if there were a way to transform sets of $O(L)$ many vectors into permutations of length L such that the existence of an orthogonal pair in the sets could be determined via an Ulam distance computation of these permutations, then that would imply a near-linear time algorithm for OV, falsifying SETH! In light of this observation, we start with $O(\sqrt{L})$ vectors in the OV instance. The constructions of the coordinate and vector gadgets are similar to the edit distance reduction. However, during the construction of the OR-gadgets, there is a quadratic blowup resulting in length L permutations with the desired properties. A detailed construction of these gadgets can be found in Section 5.

The second component in our proof is a reduction from a problem called $\exists\forall\exists$-Orthogonal Vectors. In this problem, we are given four sets A, B, C, E of binary vectors and we have to decide if there exists $a \in A$, such that for all $b \in B$, there exist $c \in C, e \in E$ such that a, b, c, e are orthogonal[6]. If $|A| = |B| = n$ and $|C| = |E| = \sqrt{L}$, then this problem has a

[6] We say that vectors $a, b, c, e \in \{0,1\}^d$ are *orthogonal* if $\sum_{i \in [d]} a[i]b[i]c[i]e[i] = 0$.

$O((n^2L)^{1-\Omega(1)})$ lower bound under the Quantified Strong Exponential Time Hypothesis. Given such sets A, B, C, E, we proceed as follows. First, for each $a \in A$, we construct the set V_a of $\sqrt{L}$ vectors by taking the pointwise product of a with all $\sqrt{L}$ vectors in C. There will be n such sets V_a, one for each choice of $a \in A$. Similarly, for each $b \in B$, we construct the set W_b of $\sqrt{L}$ vectors by taking the pointwise product of b with all $\sqrt{L}$ vectors in C. We then run the OV to Ulam distance reduction from before on these sets to obtain two sets of n-many permutations of length L. Finally, we show that there exists a permutation in the first set with small Ulam distance to every permutation in the second set if and only if the starting $\exists\forall\exists\exists$-OV instance is a YES-instance.

To go from these two sets to the final discrete center instance, we append additional symbols to each permutation and introduce a new permutation that is far from every permutation in the second set. This ensures that the center indeed comes from the first set completing the reduction. We defer the details to Section 5.

2.3 Fine-Grained Lower Bound for Discrete Median in the Ulam Metric

Our lower bound proof for the discrete median problem follows a similar initial approach as our proof for the center lower bound, with only one difference: instead of starting with an $\exists\forall\exists\exists$-OV instance, we begin with a $\exists\exists\exists\exists$-OV (also known as 4-OV) instance. Given this 4-OV instance, we retrace the same steps to construct two sets, X and Y, each containing n permutations of length L. As in the center proof, we show that there exists a permutation in X whose total Ulam distance to all elements in Y is small if and only if the original 4-OV instance is a YES-instance.

However, going from these two sets to the standard single-set version of the discrete median problem is technically very challenging. In fact, such challenges were addressed in the past in the context of the closest pair problem [21, 32], and more generally identified as the task of *reversing color coding* [14], typically requiring extremal combinatorial objects which are then composed with the input in a black-box manner.

In this paper, we transform the bichromatic instance to a monochromatic one, in multiple steps but in a white-box manner using the structure of the input instance. The first key observation is that all pairwise Ulam distances within X can be computed much faster than the naive $O(n^2L)$ time bound, specifically, in $O(n^2\sqrt{L})$ time. This speedup is possible because the permutations in X are not arbitrary but outputs formed during our OV to Ulam distance reduction. Thus, in $O(n^2\sqrt{L})$ time, we can compute the total Ulam distance of each $x \in X$ to all other elements in X.

Once these n distance sums are computed, we initiate a *balancing* procedure. This procedure iteratively appends additional symbols to each permutation in $X \cup Y$ such that:

- For every permutation in X, the sum of its Ulam distances to all other elements in X becomes equal.
- The relative Ulam distances between permutations across the sets remain unchanged.
- For every permutation in Y, the sum of its Ulam distances to all other elements in Y becomes very large.

We show that this balancing procedure can be performed efficiently without significantly increasing the permutation lengths. Finally, we include all modified permutations into a single set, and output that as our final discrete median instance. The details turn out to be quite involved, and we direct the reader to Section 6 for further details.

3 Preliminaries

Sets, Strings and Permutations

For a positive integer n, let $[n]$ denote the set $\{1, 2, \ldots, n\}$. We denote by $\mathcal{S}_n$ the set of all permutations over $[n]$. Throughout the paper, we treat any permutation $s \in \mathcal{S}_n$ as a string over the alphabet $[n]$ with no repeating symbols, and we write $s := s[1]s[2]\ldots s[n]$. Given k strings $s_1, s_2, \ldots, s_k$ over some alphabet, we denote by $\bigcirc_{i\in[k]} s_i$ the concatenated string $s_1 s_2 \ldots s_k$. We will often require strings that can be obtained by adding some fixed "offset" to each symbol of some other canonical string[7]. For every nonnegative integer k and string of length n, we let $\Delta_k(s) := (s[1]+k)(s[2]+k)\ldots(s[n]+k)$. In other words, $\Delta_k(s)$ is the string obtained by adding k to each symbol of s. We will also often require strings that are obtained by restricting some string to some sub-alphabet. Given any string s over the alphabet Σ and any sub-alphabet $\Sigma' \subseteq \Sigma$, we denote by $s|_{\Sigma'}$ to be string obtained from s by deleting all symbols that are not in Σ'. Given $a, b \in \{0,1\}^d$, we write $\langle a, b\rangle$ for the inner product of a and b, i.e., $\langle a, b\rangle = \sum_{i\in[d]} a[i]b[i]$. We further write $a \odot b$ for the pointwise product a and b, i.e., $a \odot b \in \{0,1\}^d$ is the vector satisfying $(a \odot b)[i] = a[i]b[i]$ for all $i \in [d]$.

Ulam Distances and Common Subsequences

The Hamming distance between two equal-length strings x and y, denoted by $d_H(x, y)$, is the number of locations where x and y have different symbols. Let $\pi := \pi[1]\pi[2]\ldots\pi[n] \in \mathcal{S}_n$ be a permutation and $i, j \in [n]$ be distinct positions. A *symbol relocation* operation from position j to position i in π constitutes of deleting the j^{th} symbol of π and reinserting it at position i. More formally, if $\widetilde{\pi} \in \mathcal{S}_n$ is the permutation obtained after applying a symbol relocation from position j to position i in π, then:

$$\widetilde{\pi} := \begin{cases} \pi[1]\pi[2]\ldots\pi[j-1]\pi[j+1]\cdots\pi[i-1]\pi[j]\pi[i]\pi[i+1]\cdots\pi[n], & \text{if } j < i, \\ \pi[1]\pi[2]\cdots\pi[i-1]\pi[j]\pi[i]\pi[i+1]\cdots\pi[j-1]\pi[j+1]\cdots\pi[n], & \text{if } j > i. \end{cases}$$

Given $\pi, \pi' \in \mathcal{S}_n$, the Ulam distance between π and π', denoted by $d_U(\pi, \pi')$, is the minimum number of symbol relocation operations required to transform π into π'. We will further denote by $\mathsf{LCS}(x, y)$ the length of a longest common subsequence of two strings x and y. We will frequently use the following fact throughout the paper, which relates the Ulam distance between two permutations to the length of their longest common subsequence.

▶ **Fact 4** ([6]). *For every $\pi, \pi' \in \mathcal{S}_n$, we have $d_U(\pi, \pi') = n - \mathsf{LCS}(\pi, \pi')$.*

3.1 Hardness Assumptions

Our results are conditional on several hardness assumptions and hypotheses, which we list in this section. The first one is the Strong Exponential Time Hypothesis, which is a standard assumption in the theory of fine-grained complexity.

▶ **Hypothesis 5** (Strong Exponential Time Hypothesis (SETH)). *For all $\varepsilon > 0$, there exists $q \geq 3$ such that no algorithm running in time $O(2^{(1-\varepsilon)n})$ can solve the q-SAT problem on n variables.*

[7] Here we are treating the symbols of a string as integers themselves.

More specifically, for one of our lower bounds, we will need the following corollary of SETH, which we dub Unbalanced 4-OVH.

▶ **Hypothesis 6** (Unbalanced 4-OVH). *For all $\varepsilon > 0$, no algorithm can, given sets $A, B, C, E \subseteq \{0,1\}^d$ with $|A| = n$, $|B|, |C|, |E| = n^{\Theta(1)}$ and $\omega(\log n) < d < n^{o(1)}$, determine if there exists $a \in A$, $b \in B$, $c \in C$, $e \in E$ such that $\sum_{i \in [d]} a[i]b[i]c[i]e[i] = 0$ in time $O((|A||B||C||E|)^{1-\varepsilon})$.*

It is well-known that SETH in conjunction with the sparsification lemma [30] implies Unbalanced 4-OVH [48]. We will also make use of the following strengthening of SETH.

▶ **Hypothesis 7** (∃∀∃SETH). *For all $\varepsilon > 0$ and $0 < \alpha < \beta < 1$, there exists $q \geq 3$, such that given a q-CNF formula ϕ over the variables $x_1, x_2, \ldots, x_n$, no algorithm running in time $O(2^{(1-\varepsilon)n})$ can determine if the following is true:*

$$\exists x_1, \ldots, x_{\lceil \alpha n \rceil} \forall x_{\lceil \alpha n \rceil + 1}, \ldots, x_{\lceil \beta n \rceil} \exists x_{\lceil \beta n \rceil + 1}, \ldots, x_n \ \phi(x_1, x_2, \ldots, x_n).$$

We note that ∃∀∃SETH is a special case of the Quantified SETH proposed by Bringmann and Chaudhury [12] – a hypothesis about the complexity of deciding quantified q-CNF formulas with a constant number of quantifier blocks where each block contains some constant fraction of the variables. We do not formally define Quantified SETH in all of its generality since we only require three quantfier alternations. In fact, the specific hardness assumption we need is the following which is implied by ∃∀∃SETH.

▶ **Hypothesis 8** (Unbalanced ∃∀∃∃OVH). *For all $\varepsilon > 0$, no algorithm can, given sets $A, B, C, E \subseteq \{0,1\}^d$ with $|A| = n$, $|B|, |C|, |E| = n^{\Theta(1)}$ and $\omega(\log n) < d < n^{o(1)}$, determine if there exists $a \in A$ such that for all $b \in B$, there exist $c \in C, e \in E$ such that $\sum_{i \in [d]} a[i]b[i]c[i]e[i] = 0$ in time $O((|A||B||C||E|)^{1-\varepsilon})$.*

4 NP-Hardness of Continuous Median in the Ulam Metric

In this section, we prove Theorem 1. Before we do so, we first formally define the continuous median problem in the Ulam metric.

Continuous Ulam Median	
Input:	A set $S \subseteq \mathcal{S}_n$ of permutations and an integer d.
Question:	Is there a permutation $\pi^* \in S_n$ such that $\sum_{\pi \in S} d_U(\pi, \pi^*) \leq d$?

The main result of this section is the following. All missing proofs in this section can be found in the full version of this paper [25].

▶ **Theorem 9.** *Continuous Ulam Median is NP-hard.*

Proof. We will reduce from the Max Cut problem, which is NP-hard [31]. Let $G = (V, E)$ be a Max Cut instance with $V = [n]$. From G, we will construct a Continuous Ulam Median instance $S \subseteq \mathcal{S}_N$ consisting of permutations of length $N := 3n + 2$. On a high level, our construction will work as follows. We will first construct many copies of two special permutations that will force every median of S to take on a very specific structure. All permutations of this structure will naturally encode cuts of the vertex set V. Then for each edge e in G, we will construct permutations that reward choosing a median that "cuts" e. Thus, we will end up with a set of permutations whose overall median will encode the maximum cut of G. Details follow.

To describe our construction, it will be convenient to define the following two strings, both of length $(n+1)$:

$$X_1 := (n+1) \circ (n+2) \circ \cdots \circ (2n+1),$$
$$X_2 := (2n+2) \circ (2n+3) \circ \cdots \circ (3n+2).$$

Next, we define the two special permutations $\pi^L, \pi^R \in \mathcal{S}_N$ alluded to earlier:

$$\pi^L := 1 \circ 2 \circ \cdots \circ (n-1) \circ n \circ X_1 \circ X_2,$$
$$\pi^R := X_1 \circ n \circ (n-1) \circ \cdots \circ 2 \circ 1 \circ X_2.$$

We make the observation that every median of the set $\{\pi^L, \pi^R\}$ naturally encodes a cut of the vertex set V.

▶ **Definition 10.** *For a nonnegative integer $r \leq n$, let $A = \{a_1, a_2, \ldots, a_r\}$ and $B = \{b_1, b_2, \ldots, b_{n-r}\}$ be sets such that $A \sqcup B = [n]$, $a_1 < a_2 < \cdots < a_r$ and $b_1 > b_2 > \cdots > b_{n-r}$. Define $\pi^{A,B} \in \mathcal{S}_N$ as:*

$$\pi^{A,B} := a_1 \circ a_2 \circ \cdots \circ a_r \circ X_1 \circ b_1 \circ b_2 \circ \cdots \circ b_{n-r} \circ X_2.$$

Furthermore, define $\mathcal{S}_N^* \subseteq \mathcal{S}_N$ as:

$$\mathcal{S}_N^* := \{\pi \in \mathcal{S}_N : \pi = \pi^{A,B} \text{ for some pair of sets } A, B \text{ with } A \sqcup B = [n]\}.$$

Clearly, permutations in $\mathcal{S}_N^*$ naturally encode cuts of $[n]$. We will first show that every permutation in $\mathcal{S}_N^*$ has the same sum of Ulam distances to π^L and π^R.

▶ **Lemma 11.** *For every $\pi \in \mathcal{S}_N^*$, $d_U(\pi, \pi^L) + d_U(\pi, \pi^R) = n$.*

Next, we show that any permutation that is not in $\mathcal{S}_N^*$ has strictly larger sum of Ulam distances to π^L and π^R.

▶ **Lemma 12.** *Let $\pi \notin \mathcal{S}_N^*$. Then $d_U(\pi, \pi^L) + d_U(\pi, \pi^R) \geq n+1$.*

The final pieces in our construction are the "edge gadgets", which we now define. For each edge $e = \{i, j\} \in E$, where $i < j$, define the following two strings:

$$\pi_e^1 = j \circ i \circ X_1 \circ X_2 \circ 1 \circ 2 \circ \cdots \circ (i-1) \circ (i+1) \circ \cdots \circ (j-1) \circ (j+1) \circ \cdots \circ n,$$
$$\pi_e^2 = X_1 \circ i \circ j \circ X_2 \circ 1 \circ 2 \circ \cdots \circ (i-1) \circ (i+1) \circ \cdots \circ (j-1) \circ (j+1) \circ \cdots \circ n.$$

The role of the edge gadgets associated with an edge e is to reward choosing partitions of the vertices that cut e. This is formalized in the following lemma.

▶ **Lemma 13.** *Let $\pi \in \mathcal{S}_N^*$ such that $\pi = \pi^{A,B}$ with $A \sqcup B = [n]$ and $e \in E$. Then we have the following:*

$$d_U(\pi_e^1, \pi) + d_U(\pi_e^2, \pi) = \begin{cases} 2n-2, & \text{if } e \text{ is cut by the partition } (A,B), \\ 2n-1, & \text{otherwise.} \end{cases}$$

Now let $S_E = \{\pi_e^1 : e \in E\} \cup \{\pi_e^2 : e \in E\}$ and S_{aux} be the set consisting of $t := |E|(2n-1)$ copies of π^L and π^R. Our final Continuous Ulam Median instance will be the multiset

$$S := S_E \cup S_{\text{aux}}.$$

We now show that G has a cut of size at least k if and only if the median of S has cost at most k', where $k' = |E|(2n-1) - k + tn$. For the completeness case, assume there exist sets A, B with $A \sqcup B = n$ such that the partition (A, B) cuts at least k edges in G. Denote by $E(A, B)$ the set of all edges cut by the partition (A, B). Now consider the permutation $\pi^{A,B} \in \mathcal{S}_n$ and note that:

$$\begin{aligned}
\sum_{\pi \in S} d_U(\pi^{A,B}, \pi) &= \sum_{\pi \in S_E} d_U(\pi^{A,B}, \pi) + \sum_{\pi \in S_{\text{aux}}} d_U(\pi^{A,B}, \pi) \\
&= \left(\sum_{e \in E} (d_U(\pi^{A,B}, \pi_e^1) + d_U(\pi^{A,B}, \pi_e^2)) \right) + t(d_U(\pi^{A,B}, \pi^L) + d_U(\pi^{A,B}, \pi^R)) \\
&= \left(\sum_{e \in E(A,B)} (d_U(\pi^{A,B}, \pi_e^1) + d_U(\pi^{A,B}, \pi_e^2)) \right) \\
&\qquad + \left(\sum_{e \notin E(A,B)} (d_U(\pi^{A,B}, \pi_e^1) + d_U(\pi^{A,B}, \pi_e^2)) \right) + tn \\
&= (|E(A,B)|(2n-2) + (|E| - |E(A,B)|)(2n-1)) + tn \\
&= |E|(2n-1) - |E(A,B)| + tn \leq |E|(2n-1) - k + tn.
\end{aligned}$$

For the soundness case, assume that there exists $\pi^* \in \mathcal{S}_N$ whose median cost to S is at most k'. We can further assume that $\pi^* \in \mathcal{S}_N^*$ since otherwise, every $\widetilde{\pi} \in \mathcal{S}_N^*$ will have a median cost that is at most that of π^*. Indeed, assume that $\pi^* \notin \mathcal{S}_N^*$ and fix any $\widetilde{\pi} \in \mathcal{S}_N^*$. We have:

$$\begin{aligned}
\sum_{\pi \in S} d_U(\pi^*, \pi) &= \sum_{\pi \in S_{\text{aux}}} d_U(\pi^*, \pi) + \sum_{\pi \in S_E} d_U(\pi^*, \pi) \\
&\geq t(n+1) + 0 = tn + t \\
&= tn + |E|(2n-1) \geq tn + \sum_{\pi \in S_E} d_U(\widetilde{\pi}, \pi) \\
&= \sum_{\pi \in S_{\text{aux}}} d_U(\widetilde{\pi}, \pi) + \sum_{\pi \in S_E} d_U(\widetilde{\pi}, \pi) = \sum_{\pi \in S} d_U(\widetilde{\pi}, \pi).
\end{aligned}$$

Thus, the assumption that $\pi^* \in \mathcal{S}_N^*$ is without loss of generality. Then, we have $\pi^* = \pi^{A,B}$ for sets A, B with $A \sqcup B = [n]$. We claim that the partition (A, B) cuts at least k edges in G. Indeed, since $\sum_{\pi \in S_{\text{aux}}} d_U(\pi^*, \pi) = tn$, we have $\sum_{\pi \in S_E} d_U(\pi^*, \pi) \leq k' - tn = |E|(2n-1) - k = k(2n-2) + (|E| - k)(2n-1)$. Then, by Lemma 13 the partition (A, B) cuts at least k edges. ◀

▶ Remark 14. Although our NP-hardness reduction produces multisets instead of sets, it is not to hard to turn them into sets by appending a unique permutation to each permutation without affecting the structure of the solution. See [25, Appendix B] for details.

5 Fine-Grained Complexity of Discrete Center in the Ulam Metric

In this section, we prove Theorem 2. We first formally define the discrete center problem in the Ulam metric.

The main result of this section is the following. All missing proofs in this section can be found in the full version of the paper [25].

Discrete Ulam Center	
Input:	A set $S \subseteq \mathcal{S}_L$ of permutations such that $\lvert S\rvert = n$ and an integer τ.
Question:	Is there a permutation $\pi^* \in S$ such that $\max_{\pi \in S} d_U(\pi, \pi^*) \leq \tau$?

▶ **Theorem 15.** *Let $\varepsilon > 0$ and $\alpha > 0$. There is no algorithm running in time $O((n^2L)^{1-\varepsilon})$ that solves the Discrete Ulam Center problem for n permutations of length $L = \Theta(n^\alpha)$, unless $\exists\forall\exists$SETH fails.*

The key step in the proof of Theorem 15 is a reduction from Orthogonal Vectors to Ulam Distance, i.e., to construct a pair of functions that, given a set of n binary vectors of length d each as input, outputs, independently of each other, a pair of permutations whose Ulam distance is small if and only if there exists an orthogonal pair of vectors in the input sets.

▶ **Theorem 16.** *There exists a pair of functions f and g such that for all sets $A, B \subseteq \{0,1\}^d$ with $\lvert A\rvert = \lvert B\rvert = n$, the following holds.*

- *$f(A), g(B) \in \mathcal{S}_{(5d-1)n^2}$, i.e., both f and g output permutations of length $(5d-1)n^2$.*
- *If there exists $(a,b) \in A \times B$ such that $\langle a, b\rangle = 0$, then the Ulam distance between $f(A)$ and $g(B)$ is at most $3n^2d - 1$. Otherwise, the Ulam distance between $f(A)$ and $g(B)$ is exactly $3n^2d$.*
- *Both f and g are computable in time $O(n^2d)$.*

Equipped with Theorem 16, Theorem 15 follows in a straightforward manner. For details, the reader is referred to [25].

5.1 The Need for Quantifiers

Our lower bound for Discrete Ulam Center is based on a plausible generalization of the Strong Exponential Time Hypothesis, namely $\exists\forall\exists$SETH. One could ask if we could get a similar lower bound under the weaker but more standard SETH instead. We remark that this is impossible unless the Nondeterministic Strong Exponential Time Hypothesis (NSETH) [15] is false. This is because Discrete Ulam Center can be solved in (co-)nondeterministic time $\widetilde{O}(nL)$ with the following simple algorithm. We first guess the center π_C and compute $d^* := \max_{\pi \in X} d_U(\pi_C, \pi)$. Then for each permutation π, we guess the furthest permutation $\pi' \in X$ and verify that $d_U(\pi, \pi') \geq d^*$, thereby certifying that our guess π_C is optimal. In light of this algorithm, a $O((n^2L)^{1-\Omega(1)})$ fine-grained lower bound based on SETH would contradict NSETH.

6 Fine-Grained Complexity of Discrete Median in the Ulam Metric

In this section, we prove a tight fine-grained lower bound for the Discrete Ulam Median problem conditioned on SETH. We first give a formal statement of the problem.

Discrete Ulam Median	
Input:	A set $S \subseteq \mathcal{S}_L$ of permutations such that $\lvert S\rvert = n$ and an integer τ.
Question:	Is there a permutation $\pi^* \in S$ such that $\sum_{\pi \in S} d_U(\pi, \pi^*) \leq \tau$?

This section is organized as follows. In Section 6.1, we start with a simple lower bound for the *bichromatic* version of the problem that essentially follows from the proof for the Discrete Ulam Center problem from before. Going from the bichromatic version to the standard monochromatic version defined above is presented in Section 6.2. This step is technically very involved: we show that any set of n permutations can be *balanced* in such a way that the sum of Ulam distances from each permutation to all the others is (approximately) *equal* (see Theorem 19).

6.1 Hardness for Bichromatic Instances

In this subsection, we give a fine-grained lower bound for the Bichromatic Discrete Ulam Median problem. In this problem, we given *two* sets X, Y of permutations and an integer τ, and the goal is to determine if there exists $x \in X$ such that $\sum_{y \in Y} d_U(x, y) \leq \tau$.

▶ **Theorem 17.** *Let $\varepsilon > 0$ and $\alpha > 0$. There is no algorithm running in time $O((n^2 L)^{1-\varepsilon})$ that solves the Bichromatic Discrete Ulam Median problem for n permutations of length $L = \Theta(n^\alpha)$, unless the SETH fails.*

Proof. The proof is almost identical to the first half of the proof of Theorem 15. The only difference is the starting problem, which is 4-OV (i.e., $\exists\exists\exists\exists$OV) instead of $\exists\forall\exists\exists$OV. Given a 4-OV instance $A, B, C, E \subseteq \{0,1\}^d$ where $|A| = |B| = n$, $|C| = |E| = m = n^{\Theta(1)}$ and $\omega(\log n) < d < n^{o(1)}$, we retrace the proof of Theorem 15 and construct the sets X and Y. Finally, we set $\tau := 3m^2nd - 1$. Once again, by Theorem 16, it is not hard to see that there exists $x \in X$ such that $\sum_{y \in Y} d_U(x, y) \leq \tau$ if and only if the starting 4-OV instance is a YES-instance. The conclusion then follows from Unbalanced 4-OVH. ◀

6.2 Hardness for Monochromatic Instances

Finally, in this subsection, we provide a fine-grained lower bound for the Monochromatic Discrete Ulam Median problem. To do so, we need the following technical results (which is one of our main contributions), whose proofs can be found in the full version of the paper [25].

▶ **Lemma 18** (Embedding the Hamming Metric on Small Alphabets). *Let $a_1, \ldots, a_n \in \{0,1,2\}^L$. In time $O(nL)$ we can construct permutations $\pi_1, \ldots, \pi_n \in \mathcal{S}_{3L}$ such that $d_H(a_i, a_j) = d_U(\pi_i, \pi_j)$ for all $i, j \in [n]$.*

▶ **Theorem 19** (Full Balancing). *Let n and L be integers such that n is divisible by 4 and let $k_1, \ldots, k_n \in [O(nL)]$. Given $k_1, \ldots, k_n$, in time $\widetilde{O}(n^2 + nL)$ we can construct permutations $\pi_1, \ldots, \pi_n, \tau \in \mathcal{S}_{\widetilde{O}(n+L)}$ and an integer d with the following two properties:*

- *Writing $d_i = \sum_{j \neq i} d_U(\pi_i, \pi_j)$, it holds that $|(k_i + d_i) - d| \leq 1$ for all $i \in [n]$.*
- *It holds that $d_U(\pi_1, \tau) = \cdots = d_U(\pi_n, \tau)$.*

Using the above theorem, we now prove the following:

▶ **Theorem 20.** *Let $\varepsilon > 0$ and $\alpha \geq 1$. There is no algorithm running in time $O((n^2 L)^{1-\varepsilon})$ that solves the (Monochromatic) Discrete Ulam Median problem for n permutations of length $L = \Theta(n^\alpha)$, unless the SETH fails.*

Proof. We follow the same reduction as in Theorem 17, which, given an initial 4-OV instance produces an instance (X, Y) of Bichromatic Discrete Ulam Median on n permutations of length L. During the reduction, we can set everything up so that $L = \Omega(n)$. Without loss

of generality, we may also assume that n is divisible by 4 by adding some dummy vectors in the initial 4-OV instance if necessary. Let $x_1, \ldots, x_n$ denote the permutations in X and let $y_1, \ldots, y_n$ denote the permutations in Y. As a first preprocessing step, concatenate each permutation x_i (or y_i) two times to itself (using fresh symbols when necessary) so that each permutation becomes of length $3L$ and the median distance $\min_i \sum_j d_U(x_i, y_j)$ becomes a multiple of 3.

We first make the observation that the Ulam distance between any two permutations in X can be computed very quickly – in time that is proportional to the square root of the length of the permutations.

▶ **Observation 21.** *Let $x, x' \in X$. Then $d_U(x, x')$ can be computed in time $O(L^{1/2+o(1)})$.*

Proof. Since $x \in X$, there exist $T = \{t_1, t_2, \ldots, t_m\} \subseteq \{0,1\}^d$ such that $x = f(T) \circ f(T) \circ f(T)$, where f is the function from Theorem 16 and each concatenation is done using a fresh set of symbols. Similarly, there exist $T' = \{t'_1, t'_2, \ldots, t'_m\} \subseteq \{0,1\}^d$ such that $x' = f(T') \circ f(T') \circ f(T')$, where again each concatenation is with a fresh set of symbols.

Therefore, we have $d_U(x, x') = 3m \sum_{i \in [m]} d_H(t_i, t'_i)$. Thus, $d_U(x, x')$ can be found by computing the Hamming distance between two bit strings of length md. Further recall that $L = O(m^2 d)$. Therefore, $md = O(L^{1/2+o(1)})$ and the conclusion follows. ◀

By Observation 21, we can compute in time $O(n^2 L^{1/2+o(1)})$ all pairwise distances $d_U(x_i, x_j)$. Let $k_i := \sum_{j \neq i} d_U(x_i, x_j)$; clearly we have that $k_i \leq 3nL$. Thus, we may apply Theorem 19 to obtain permutations $\pi_1, \ldots, \pi_n, \tau \in \mathcal{S}_{L'}$, where $L' = \widetilde{O}(n + L) = \widetilde{O}(L)$, with

$$|(k_i + \sum_{j \neq i} d_U(\pi_i, \pi_j)) - D| \leq 1$$

for some integer D and for all $i \in [n]$, and with $M := d_U(\pi_1, \tau) = \cdots = d_U(\pi_n, \tau)$.

Additionally, let $K := 10(3L + L')$. Compute some length-$O(K)$ permutations $\mu, \eta_1, \ldots, \eta_n$ such that $d_U(\mu, \eta_i) = K$ and such that $\sum_j d_U(\eta_i, \eta_j) = nK$. For instance, viewing $\mu, \eta_1, \ldots, \eta_n$ as 0-1-strings under the Hamming distance to be embedded by Lemma 18, take μ to be the all-0 string of length $2K$, let half of the strings η_i be the string $0^K 1^K$ and let the other half of the strings η_i be the string $1^K 0^K$.

We are now ready to construct the Monochromatic Discrete Ulam Median instance Z. We include into Z the following $2n$ permutations:[8]

- $x'_i := x_i \, \pi_i \, \mu$ (for $i \in [n]$), and
- $y'_i := y_i \, \tau \, \eta_i$ (for $i \in [n]$).

We claim that this construction is correct in the following sense: (1) All discrete medians of Z are strings x'_i. (2) Whenever x'_i is a discrete median in Z, then x_i is a discrete median in (X, Y). (3) There is some discrete median x_i in (X, Y) such that x'_i is a discrete median in Z. The proofs of all three claims easily follow from the following calculations. On the one hand, the median distance for each x'_i is

[8] As always, we use fresh symbols when necessary to ensure that the resulting strings are permutations.

$$\begin{aligned}\sum_{z\in Z} d_U(x'_i, z) &= \sum_j d_U(x'_i, x'_j) + \sum_j d_U(x'_i, y'_j)\\ &= \sum_j (d_U(x_i, x_j) + d_U(\pi_i, \pi_j) + d_U(\mu, \mu))\\ &\qquad + \sum_j (d_U(x_i, y_j) + d_U(\pi_i, \tau) + d_U(\mu, \eta_i))\\ &= k_i + \sum_j d_U(\pi_i, \pi_j) + \sum_j d_U(x_i, y_j) + nM + nK\\ &= \sum_j d_U(x_i, y_j) + nM + nK + D \pm 1.\end{aligned}$$

On the other hand, the median distance for each y'_i is

$$\sum_{z\in Z} d_U(y'_i, z) = \sum_j d_U(y'_i, x'_j) + \sum_j d_U(y'_i, y'_j) \geq \sum_j d_U(\eta_i, \mu) + \sum_j d_U(\eta_i, \eta_j) = 2nK.$$

Comparing these two terms, and recalling that $\sum_j d_U(x_i, y_j) + nM + D \leq 3nL + nL' + nL' < nK$, it is clear that the median distance of any y'_i is always significantly larger that the median distance of any x'_i proving (1). Recalling further that all median distances $\sum_j d_U(x_i, y_j)$ in the original instance are multiples of 3, the ± 1 term in the first computation becomes irrelevant, completing the proofs of claims (2) and (3).

Finally we comment on the running time. The original reduction, along with the computation of the values k_i takes time $\widetilde{O}(nL + n^2 L^{1/2+o(1)})$. Running Theorem 19 takes time $\widetilde{O}(n^2 + nL)$, and the final instance Z can be implemented in negligible overhead. ◀

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.14.
2. Amir Abboud, MohammadHossein Bateni, Vincent Cohen-Addad, Karthik C. S., and Saeed Seddighin. On complexity of 1-center in various metrics. In Nicole Megow and Adam D. Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPIcs*, pages 1:1–1:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.APPROX/RANDOM.2023.1.
3. Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Scheduling lower bounds via AND subset sum. *J. Comput. Syst. Sci.*, 127:29–40, 2022. doi:10.1016/J.JCSS.2022.01.005.
4. Amir Abboud, Nick Fischer, Elazar Goldenberg, Karthik C. S., and Ron Safier. Can you solve closest string faster than exhaustive search? In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 3:1–3:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.3.
5. Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. doi:10.1145/1411509.1411513.
6. David Aldous and Persi Diaconis. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem. *Bulletin of the American Mathematical Society*, 36(4):413–432, 1999.

7 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.

8 Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner, and Andreas Hofmeier. On the hardness of maximum rank aggregation problems. *J. Discrete Algorithms*, 31:2–13, 2015. doi:10.1016/J.JDA.2014.10.002.

9 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

10 Therese Biedl, Franz-Josef Brandenburg, and Xiaotie Deng. On the complexity of crossings in permutations. *Discret. Math.*, 309(7):1813–1823, 2009. doi:10.1016/J.DISC.2007.12.088.

11 Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors. *Handbook of Computational Social Choice*. Cambridge University Press, 2016. doi:10.1017/CBO9781107446984.

12 Karl Bringmann and Bhaskar Ray Chaudhury. Polyline simplification has cubic complexity. *J. Comput. Geom.*, 11(2):94–130, 2020. doi:10.20382/JOCG.V11I2A5.

13 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.

14 Boris Bukh, Karthik C. S., and Bhargav Narayanan. Applications of random algebraic constructions to hardness of approximation. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 237–244. IEEE, 2021. doi:10.1109/FOCS52979.2021.00032.

15 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270. ACM, 2016. doi:10.1145/2840728.2840746.

16 Diptarka Chakraborty, Debarati Das, and Robert Krauthgamer. Approximating the median under the ulam metric. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 761–775. SIAM, 2021. doi:10.1137/1.9781611976465.48.

17 Diptarka Chakraborty, Debarati Das, and Robert Krauthgamer. Clustering permutations: New techniques with streaming applications. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 31:1–31:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ITCS.2023.31.

18 Diptarka Chakraborty, Syamantak Das, Arindam Khan, and Aditya Subramanian. Fair rank aggregation. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL: http://papers.nips.cc/paper_files/paper/2022/hash/974309ef51ebd89034adc64a57e304f2-Abstract-Conference.html.

19 Diptarka Chakraborty, Kshitij Gajjar, and Agastya Vibhuti Jha. Approximating the center ranking under ulam. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPIcs*, pages 12:1–12:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.FSTTCS.2021.12.

20 Graham Cormode, S. Muthukrishnan, and Süleyman Cenk Sahinalp. Permutation editing and matching via embeddings. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 481–492. Springer, 2001. doi:10.1007/3-540-48224-5_40.

21 Roee David, Karthik C. S., and Bundit Laekhanukit. On the complexity of closest pair via polar-pair of point-sets. *SIAM J. Discret. Math.*, 33(1):509–527, 2019. doi:10.1137/17M1128733.

22 Colin de la Higuera and Francisco Casacuberta. Topology of strings: Median string is np-complete. *Theor. Comput. Sci.*, 230(1-2):39–48, 2000. doi:10.1016/S0304-3975(97)00240-5.

23 Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In Vincent Y. Shen, Nobuo Saito, Michael R. Lyu, and Mary Ellen Zurko, editors, *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 613–622. ACM, 2001. doi:10.1145/371920.372165.

24 Ronald Fagin, Ravi Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 301–312. ACM, 2003. doi:10.1145/872757.872795.

25 Nick Fischer, Elazar Goldenberg, Mursalin Habib, and Karthik C. S. Hardness of median and center in the ulam metric. *CoRR*, abs/2504.16437, 2025. doi:10.48550/arXiv.2504.16437.

26 Moti Frances and Ami Litman. On covering problems of codes. *Theory Comput. Syst.*, 30(2):113–119, 1997. doi:10.1007/S002240000044.

27 Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M. LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nat.*, 494(7435):77–80, 2013. doi:10.1038/NATURE11875.

28 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

29 Donna Harman. Ranking algorithms. In William B. Frakes and Ricardo A. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 363–392. Prentice-Hall, 1992.

30 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

31 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

32 Karthik C. S. and Pasin Manurangsi. On closest pair in euclidean metric: Monochromatic is as hard as bichromatic. *Comb.*, 40(4):539–573, 2020. doi:10.1007/S00493-019-4113-1.

33 John G. Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

34 Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 95–103. ACM, 2007. doi:10.1145/1250790.1250806.

35 Teuvo Kohonen. Median strings. *Pattern Recognit. Lett.*, 3(5):309–313, 1985. doi:10.1016/0167-8655(85)90061-3.

36 J. Kevin Lanctôt, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. Distinguishing string selection problems. *Inf. Comput.*, 185(1):41–55, 2003. doi:10.1016/S0890-5401(03)00057-9.

37 Xue Li, Xinlei Wang, and Guanghua Xiao. A comparative study of rank aggregation methods for partial and top ranked lists in genomic applications. *Briefings in bioinformatics*, 20(1):178–189, 2019. doi:10.1093/BIB/BBX101.

38 Yuting Liu, Tie-Yan Liu, Tao Qin, Zhiming Ma, and Hang Li. Supervised rank aggregation. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 481–490. ACM, 2007. doi:10.1145/1242572.1242638.

39 Carlos D. Martínez-Hinarejos, Alfons Juan, and Francisco Casacuberta. Use of median string for classification. In *15th International Conference on Pattern Recognition, ICPR'00, Barcelona, Spain, September 3-8, 2000*, pages 2903–2906. IEEE Computer Society, 2000. doi:10.1109/ICPR.2000.906220.

40 François Nicolas and Eric Rivals. Complexities of the centre and median string problems. In Ricardo A. Baeza-Yates, Edgar Chávez, and Maxime Crochemore, editors, *Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Morelia, Michocán, Mexico, June 25-27, 2003, Proceedings*, volume 2676 of *Lecture Notes in Computer Science*, pages 315–327. Springer, 2003. doi:10.1007/3-540-44888-8_23.

41 Samuel E. L. Oliveira, Victor Diniz, Anísio Lacerda, Luiz H. C. Merschmann, and Gisele L. Pappa. Is rank aggregation effective in recommender systems? an experimental analysis. *ACM Trans. Intell. Syst. Technol.*, 11(2):16:1–16:26, 2020. doi:10.1145/3365375.

42 Pavel A. Pevzner. *Computational molecular biology - an algorithmic approach*. MIT Press, 2000.

43 V. Y. Popov. Multiple genome rearrangement by swaps and by element duplications. *Theor. Comput. Sci.*, 385(1-3):115–126, 2007. doi:10.1016/J.TCS.2007.05.029.

44 Cyrus Rashtchian, Konstantin Makarychev, Miklós Z. Rácz, Siena Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. Clustering billions of reads for DNA data storage. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3360–3371, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/ab7314887865c4265e896c6e209d1cd6-Abstract.html.

45 Craige Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of mathematics*, 13:179–191, 1961.

46 Anke van Zuylen and David P. Williamson. Deterministic algorithms for rank aggregation and other ranking and clustering problems. In Christos Kaklamanis and Martin Skutella, editors, *Approximation and Online Algorithms, 5th International Workshop, WAOA 2007, Eilat, Israel, October 11-12, 2007. Revised Papers*, volume 4927 of *Lecture Notes in Computer Science*, pages 260–273. Springer, 2007. doi:10.1007/978-3-540-77918-6_21.

47 Tiance Wang, John Sturm, Paul W. Cuff, and Sanjeev R. Kulkarni. Condorcet voting methods avoid the paradoxes of voting theory. In *50th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2012, Allerton Park & Retreat Center, Monticello, IL, USA, October 1-5, 2012*, pages 201–203. IEEE, 2012. doi:10.1109/ALLERTON.2012.6483218.

48 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/J.TCS.2005.09.023.

49 H. Peyton Young. Condorcet's theory of voting. *American Political science review*, 82(4):1231–1244, 1988.

50 H. Peyton Young and Arthur Levenglick. A consistent extension of condorcet's election principle. *SIAM Journal on applied Mathematics*, 35(2):285–300, 1978.

A Faster Parametric Search for the Integral Quickest Transshipment Problem

Mariia Anapolska
Combinatorial Optimization, RWTH Aachen University, Germany

Dario van den Boom[1]
Combinatorial Optimization, RWTH Aachen University, Germany

Christina Büsing
Combinatorial Optimization, RWTH Aachen University, Germany

Timo Gersing
Combinatorial Optimization, RWTH Aachen University, Germany

Abstract

Algorithms for computing fractional solutions to the quickest transshipment problem have been significantly improved since Hoppe and Tardos first solved the problem in strongly polynomial time. For integral solutions, however, no structural improvements on their algorithm itself have yet been proposed. Runtime improvements are limited to general progress on submodular function minimization (SFM), which is an integral part of Hoppe and Tardos' algorithm. In fact, SFM constitutes the main computational load of the algorithm, as the runtime is blown up by using it within Megiddo's parametric search algorithm. We replace this part of Hoppe and Tardos' algorithm with a more efficient routine that solves only a linear number of SFM and, in contrast to previous techniques, exclusively uses minimum cost flow algorithms within Megiddo's parametric search. Our approach improves the state-of-the-art runtime from $\tilde{\mathcal{O}}(m^4k^{15})$ down to $\tilde{\mathcal{O}}(m^2k^5 + m^4k^2)$[2], where k is the number of terminals and m is the number of arcs.

2012 ACM Subject Classification Theory of computation → Network flows

Keywords and phrases Flow over time, dynamic transshipment, quickest transshipment, parametric submodular functions, efficient algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.112

Related Version *Full Version*: `https://arxiv.org/abs/2505.12975` [1]

Funding This work is co-funded by the German research council (DFG) – project number 442047500 – Collaborative Research Center Sparsity and Singular Structures (SFB 1481) and by the German research council (DFG) Research Training Group 2236 UnRAVeL.

Acknowledgements We would like to express our gratitude to the anonymous reviewers for their valuable feedback and constructive comments, which have greatly contributed to improving the presentation of this paper.

1 Introduction

Network flows over time, also referred to as dynamic flows, extend classical static network flows by a time component. They provide a powerful tool for modeling real-world problems in traffic engineering, building evacuation, and logistics. Over the last decades, a wide range of optimization problems dealing with flows over time have been studied. The *maximum flow over time problem* was studied in the seminal work of Ford and Fulkerson [3], who showed

[1] Corresponding Author
[2] We use $\tilde{\mathcal{O}}$ to suppress polylogarithmic terms.


© Mariia Anapolska, Dario van den Boom, Christina Büsing, and Timo Gersing;
licensed under Creative Commons License CC-BY 4.0



that the problem can be solved in polynomial time by a reduction to the static minimum cost flow problem. The *quickest flow problem* asks for the minimum time-horizon T^* such that a provided demand of $D \in \mathbb{N}$ units of flow can be sent from a single source s to a single sink t. While a straightforward approach is to combine an algorithm for maximum flows over time with a parametric search algorithm [2], recent results have shown that cost scaling algorithms for minimum cost flows can be modified in order to efficiently compute quickest flows [7, 10].

The *quickest transshipment problem* generalizes the quickest flow problem by allowing for *supply* and *demand* at multiple *sources* and *sinks*. It is one of the most fundamental problems in the field of network flows over time and, as recently stated by Skutella [14], "arguably the most difficult flow over time problem that can still be solved in polynomial time." As in the quickest flow problem, our goal is to send the required flow from sources to sinks while simultaneously minimizing the time horizon.

Similar to the quickest flow problem, the quickest transshipment problem can be solved by determining the minimum time horizon via parametric search. Using this idea, Hoppe and Tardos [4] showed that the quickest transshipment problem can be solved in strongly polynomial time, that is, their algorithm's runtime is polynomially bounded in the number of nodes n, number of arcs m, and the combined number of sources and sinks k.

Recently, faster algorithms have been developed. Notably, Schlöter, Skutella, and Tran [12] proposed an algorithm with a time complexity of $\tilde{\mathcal{O}}(m^2k^5 + m^3k^3 + m^3n)$. Unfortunately, these performance improvements come at the expense of fractional solutions, which may be undesirable for applications that do not allow flow particles to be disassembled. While some of the results speed up the search for the optimal time horizon, no improvements have yet been proposed for finding integral flows over time. Hence, the state-of-the-art complexity for the integral quickest transshipment problem remains unchanged at $\tilde{\mathcal{O}}(m^4k^{15})$ [12].

Our Contribution

We propose improvements to Hoppe and Tardos' algorithm by replacing two computationally expensive subroutines in which submodular function minimization is combined with Megiddo's parametric search. By doing so, we reduce the state-of-the-art runtime for the integral quickest transshipment problem from $\tilde{\mathcal{O}}(m^4k^{15})$ to $\tilde{\mathcal{O}}(m^2k^5 + m^4k^2)$. This narrows the gap to the fractional quickest transshipment problem, which can be solved in $\tilde{O}(m^2k^5 + m^3k^3 + m^3n)$ time using the algorithm by Schlöter, Skutella, and Tran [12].

2 Preliminaries

Given a directed graph $G = (V, A)$ with vertices V and arcs A, we define a *dynamic network* as a triple $\mathcal{N} = (G, u, \tau)$ with *capacity* $u_a \in \mathbb{N}$ and *transit time* $\tau_a \in \mathbb{N}_0$ for each arc $a \in A(\mathcal{N})$. For a given dynamic network $\mathcal{N}$, the set $V(\mathcal{N})$ denotes the network's nodes, while $A(\mathcal{N})$ refers to the network's arcs. Throughout this paper, we denote the number of nodes $|V(\mathcal{N})|$ by n and the number of arcs $|A(\mathcal{N})|$ by m.

A *flow over time* is a family of functions $f_a \colon [0, T) \to \mathbb{R}_{\geq 0}$, representing the in-flow rates for each arc $a \in A(\mathcal{N})$ for every point in time until the end of the *time horizon* $T \in \mathbb{N}$. The $f_a(\theta)$-many flow units entering arc a at time $\theta \in [0, T)$ arrive at time $\theta + \tau_a$ at the end node of a. While we only require f_a to satisfy weak flow conservation, Hoppe and Tardos' algorithm computes an optimal solution satisfying even the strict flow conservation, meaning that all flow units entering a non-sink node are immediately forwarded via an outgoing arc. We refer the reader to [13] for more details on flows over time.

Note that, in contrast to Hoppe and Tardos [4] who defined a flow over time as a static flow in a time-expanded network, we use a continuous-time model. However, all prerequisites for their algorithm directly translate from the discrete- to continuous-time model as shown in [14], meaning that it can also be implemented for the continuous-time model.

For the *dynamic transshipment problem*, we have a triple $(\mathcal{N}, b, T)$ comprising a dynamic network $\mathcal{N}$, a *balance function* $b\colon V(\mathcal{N}) \to \mathbb{Z}$ with $\sum_{v \in V(\mathcal{N})} b(v) = 0$, and a time horizon T. The balances describe how supply and demand are distributed across the network. A node with positive balance $b(v) > 0$ is a *source*, while a node with negative balance is a *sink*. Let S^+ denote the set of sources, S^- the set of sinks, and $S = S^+ \cup S^-$ the set of *terminals*. A dynamic transshipment instance is *feasible* if there exists a flow over time sending the supply from the sources to the sinks such that all demands are satisfied.

▶ **Definition 1.** *Given a subset of terminals $X \subseteq S$, the* maximum out-flow $o(X)$ out of X *is the value of the maximum flow over time from the sources $S^+ \cap X$ to the sinks $S^- \setminus X$.*

The central feasibility criterion states that the *net balance* $b(X) \coloneqq \sum_{v \in X} b(v)$ must not exceed the maximum out-flow $o(X)$ for every $X \subseteq S$.

▶ **Theorem 2** (Feasibility Criterion [4])**.** *The dynamic transshipment instance $(\mathcal{N}, b, T)$ is feasible if and only if $v(X) \coloneqq o(X) - b(X) \geq 0$ for all $X \subseteq S$.*

We call a set $X \subseteq S$ with $v(X) < 0$ a *violated set*. In order to determine the feasibility of a given dynamic transshipment instance, it suffices to show that no violated set exists. However, while the value of $o(X)$, and thus of $v(X)$, can be computed with the Ford-Fulkerson algorithm for maximum flows over time, avoiding the enumeration of all subsets $X \subseteq S$ is not obvious. Fortunately, we can use the *submodularity* of $o\colon 2^S \to \mathbb{N}_0$ and $v\colon 2^S \to \mathbb{Z}$.

▶ **Definition 3** (Submodular Function)**.** *A function $f\colon 2^S \to \mathbb{R}$ over a finite* ground set S *is a* submodular function *if for all $X, Y \subseteq S$ it holds that*

$$f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y), \tag{1}$$

or, equivalently, if for all $s \in S$ and $X \subseteq Y \subseteq S \setminus \{s\}$ it holds that

$$f(Y \cup \{s\}) - f(Y) \leq f(X \cup \{s\}) - f(X). \tag{2}$$

Given a submodular function f over S, we call a set $X^* \in \operatorname{argmin}_{X \subseteq S} f(X)$ a *minimizer* of f. It is well-known that the set of minimizers of a submodular function is closed under union and intersection. Therefore, there always exists a *minimal minimizer* and a *maximal minimizer*, which are the intersection and union of all minimizers, respectively.

In the context of *submodular function minimization (SFM)*, a submodular function f is typically provided in form of an *evaluation oracle* with complexity $\mathcal{O}(\mathrm{EO})$. The performance of algorithms is measured in the number of oracle calls required for finding a minimizer. The fastest strongly polynomial algorithm for submodular function minimization is due to Lee, Sidford and Wong [6] with a runtime of $\mathcal{O}(k^3 log^2 k \cdot \mathcal{O}(\mathrm{EO}) + k^4 log^{\mathcal{O}(1)} k)$ for $k = |S|$.

For our purpose, the evaluation oracle computes a maximum out-flow $o(X)$ out of terminals $X \subseteq S$ using the Ford-Fulkerson algorithm for maximum flows over time. To this end, we introduce a super-source s^+ and a super-sink s^- and connect them to sources $s \in S^+ \cap X$ and sinks $t \in S^- \setminus X$, respectively, via infinite-capacity, zero-transit arcs. Then the maximum out-flow $o(X)$ is the value of the maximum flow over time from s^+ to s^-. This value can be computed using a static min-cost flow in $\mathcal{O}(m \log n(m + n \log n))$ or $\tilde{\mathcal{O}}(m^2)$ time via Orlin's algorithm [9]. We abbreviate this runtime as $\mathcal{O}(\mathrm{MCF}(n, m))$.

Consequently, determining whether a dynamic transshipment instance $(\mathcal{N}, b, T)$ is feasible takes $\mathcal{O}(k^3 log^2 k \cdot \mathrm{MCF}(n,m) + k^4 log^{\mathcal{O}(1)} k)$ or $\tilde{\mathcal{O}}(m^2k^3)$ time, where $k = |S|$ is the number of terminals. To improve readability, we denote the time it takes to check if an instance with k terminals, n nodes and m arcs is feasible by $\mathcal{O}(\mathrm{SFM}(k,n,m))$.

The Algorithm by Hoppe and Tardos

The algorithm by Hoppe and Tardos [4] was the first strongly polynomial algorithm computing quickest transshipments and remains the most efficient one for integral solutions. In the following, we assume that the minimum time horizon T^* is provided and focus on the algorithm's segment that computes an integral dynamic transshipment. The algorithm relies on the concept of tight orders.

▶ **Definition 4** (Tight Set and Order). *A set of terminals $X \subseteq S$ is called* tight *if $o(X) = b(X)$ holds. Let $\preceq$ be a total order on S. We call $\preceq$* tight *if $\{t' \in S \mid t' \preceq t\}$ is tight for all $t \in S$.*

The general idea by Hoppe and Tardos is to construct an equivalent dynamic transshipment instance for which a tight order exists.

▶ **Theorem 5** (Reduction to Lex-Max Flows [4]). *Given a dynamic transshipment instance $(\mathcal{N}, b, T)$ with a tight order $\preceq$ over the terminals S, an integral dynamic transshipment satisfying b can be computed as a lex-max flow over time in $\mathcal{O}(k\,\mathrm{MCF}(n,m))$ time.*

We refer to [14] for more details on lex-max flows over time. Although computing the integral flow is quite efficient, transforming $(\mathcal{N}, b, T)$ into an equivalent instance $(\mathcal{N}', b', T)$ with a tight order is computationally demanding. We propose improvements to this transformation in Section 5. Before that, we briefly discuss the approach by Hoppe and Tardos.

First, one adds a new terminal $\check{s}$ with $b(\check{s}) = b(s)$ for every terminal $s \in S$ and then sets $b(s) = 0$. Note that the set of terminals S now only contains the nodes $\check{s}$. By adding infinite-capacity, zero-transit arcs $(\check{s}, s)$ for $\check{s} \in S^+$ and $(s, \check{s})$ for $\check{s} \in S^-$, one ensures that the resulting dynamic transshipment instance is equivalent to the original instance (cf. Figure 1).

To construct an instance admitting a tight order, we iteratively shift supply / demand from terminals $\check{s}$ to new terminals $\hat{s}$. In this paper, we call the $\check{s}$ *drained terminals* and the $\hat{s}$ *filled terminals*. We refer the reader to Hoppe and Tardos [4] and the description in the extended paper [1] for a general overview of the algorithm. We focus on the two subroutines, MaximizeAlpha and MinimizeDelta, which compute the amount of supply / demand that is shifted from $\check{s}$ to $\hat{s}$.

When discussing both subroutines, we assume that a feasible dynamic transshipment instance $(\mathcal{N}, b, T)$ is given, with a set of terminals S consisting of the drained terminals $\check{s}$, added in the first step, and all filled terminals $\hat{s}$ that were introduced in previous iterations. Furthermore, we are given two tight sets $Q \subset R \subseteq S$ and a drained terminal $\check{s} \in R \setminus Q$ satisfying $o(Q \cup \{\check{s}\}) > b(Q \cup \{\check{s}\})$.

Note that the instance $(\mathcal{N}, b, T)$, its set of terminals S as well as Q, R, and $\check{s}$ are the input of our subroutines and vary between iterations. In contrast, every subroutine adds a new filled terminal. However, since both subroutines are discussed in isolation, we simplify the notation by referring to both filled terminals as $\hat{s}$. Throughout this paper, we define $\hat{Q} \coloneqq Q \cup \{\hat{s}\}$ and $\hat{R} \coloneqq R \cup \{\hat{s}\}$. Similar to Hoppe and Tardos, we only discuss the case in which $\check{s}$ is a source, since the treatment of sinks is symmetrical and is sketched in the extended paper [1].

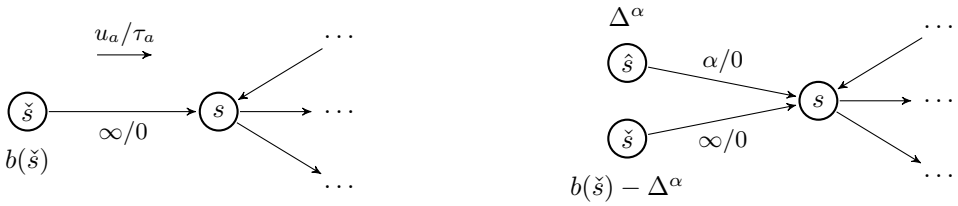


Figure 1 A dynamic transshipment instance $(\mathcal{N}, b, T)$ after initialization with terminal $\check{s}$ (left) and the corresponding α-parametric instance $(\mathcal{N}^\alpha, b^\alpha, T)$ for MaximizeAlpha (right).

Capacity-Parametric Instances

The first subroutine starts with tight sets Q and R and a drained source $\check{s} \in R \setminus Q$ for which $Q \cup \{\check{s}\}$ is not tight. Its aim is to reassign as much supply as possible to a new filled source $\hat{s}$. In doing so, we capacitate the out-flow of $\hat{s}$ such that $\hat{Q}$ is tight and the transformed instance remains feasible.

▶ **Definition 6** (α-Parametric Dynamic Network). *Given a dynamic network $\mathcal{N}$, a drained source $\check{s} \in S^+$ and a parameter $\alpha \in \mathbb{N}_0$, the* α-parametric network $\mathcal{N}^\alpha$ *is constructed by adding a new filled source $\hat{s}$ and connecting it to s via an α-capacity, zero-transit arc $(\hat{s}, s)$.*

Let $o^\alpha \colon 2^{S \cup \{\hat{s}\}} \to \mathbb{N}_0$ be the parametric counterpart of the maximum out-flow as in Definition 1 in the parametric network $\mathcal{N}^\alpha$. Using this notation, we define *α-parametric dynamic transshipment instances* analogously to Hoppe and Tardos [4]. The construction of an α-parametric instance is illustrated in Figure 1.

▶ **Definition 7** (α-Parametric Dynamic Transshipment Instance). *Given a feasible dynamic transshipment instance $(\mathcal{N}, b, T)$, two tight sets of terminals $Q \subset R \subseteq S$, a drained source $\check{s} \in R \setminus Q$ and a parameter $\alpha \in \mathbb{N}_0$, the corresponding* α-parametric dynamic transshipment instance $(\mathcal{N}^\alpha, b^\alpha, T)$ *consists of the following components.*

- *An α-parametric dynamic network $\mathcal{N}^\alpha$ as in Definition 6.*
- *An α-parametric balance function b^α with $b^\alpha(t) = b(t)$ for all terminals $t \in S \setminus \{\check{s}\}$, $b^\alpha(\hat{s}) = \Delta^\alpha$, and $b^\alpha(\check{s}) = b(\check{s}) - \Delta^\alpha$, where $\Delta^\alpha := o^\alpha(\hat{Q}) - o^\alpha(Q)$.*

We call a parameter value $\alpha \in \mathbb{N}_0$ *feasible* if the corresponding α-parametric dynamic transshipment instance $(\mathcal{N}^\alpha, b^\alpha, T)$ is feasible. Determining whether a value α is feasible is equivalent to checking if a *violated* set $X \subseteq S \cup \{\hat{s}\}$ exists. Recall that this can be done by minimizing the parametric submodular function $v^\alpha(X) = o^\alpha(X) - b^\alpha(X)$. A subroutine of the algorithm by Hoppe and Tardos finds a maximum feasible parameter value $\alpha \in \mathbb{N}_0$.

▶ **Definition 8** (MaximizeAlpha). *Given an α-parametric dynamic transshipment instance, find the maximum feasible $\alpha \in \mathbb{N}_0$.*

We denote the maximum feasible parameter value by α^*. Hoppe and Tardos concluded that, given a feasibility oracle for $(\mathcal{N}^\alpha, b^\alpha, T)$ taking $\mathcal{O}(\mathrm{SFM}(k, n, m))$ time, the value α^* can be found in $\mathcal{O}(\log(nU_{\max}) \cdot \mathrm{SFM}(k, n, m))$ time, where $U_{\max} := \max_{a \in A(\mathcal{N})} u_a$. In addition, one can combine Megiddo's parametric search [8] with the combinatorial SFM algorithm by Orlin and Iwata [5] to achieve a strongly polynomial runtime of $\tilde{\mathcal{O}}(m^4 k^{14})$. We mainly improve upon the strongly polynomial approach.

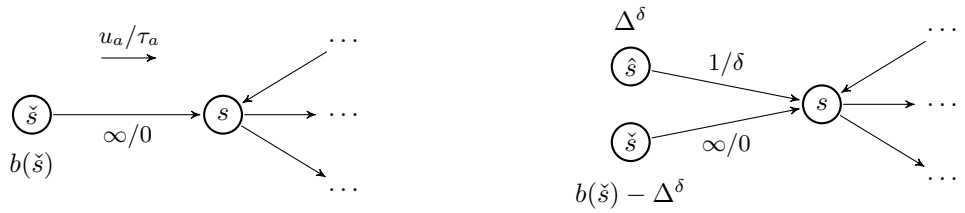


Figure 2 A dynamic transshipment instance $(\mathcal{N}, b, T)$ after initialization with terminal $\check{s}$ (left) and the corresponding δ-parametric instance $(\mathcal{N}^\delta, b^\delta, T)$ for MinimizeDelta (right).

Transit-Parametric Instances

The second subroutine is closely related to MaximizeAlpha. Again, we start with tight sets Q and R and a drained source $\check{s} \in R \setminus Q$ for which $Q \cup \{\check{s}\}$ is not tight. The aim is to reassign as much supply as possible to a new filled source $\hat{s}$. In doing so, we set the transit time for the flow out of $\hat{s}$ such that $\hat{Q}$ is tight and the transformed instance remains feasible.

▶ **Definition 9** (δ-Parametric Dynamic Network)**.** *Given a dynamic network $\mathcal{N}$, a drained source $\check{s} \in S^+$ and a parameter $\delta \in \mathbb{N}_0$, the* δ-parametric network $\mathcal{N}^\delta$ *is constructed by adding a terminal $\hat{s}$ and connecting it to s via a unit-capacity, δ-transit arc $(\hat{s}, s)$.*

Analogously to o^α, we denote the maximum out-flow of a subset of terminals $X \subset S \cup \{\hat{s}\}$ by $o^\delta(X)$. Next, we combine this parametric variant of our dynamic network with a parametric balance function to form a *δ-parametric dynamic transshipment instance* (cf. Figure 2).

▶ **Definition 10** (δ-Parametric Dynamic Transshipment Instance)**.** *Given a feasible dynamic transshipment instance $(\mathcal{N}, b, T)$, two tight sets of terminals $Q \subset R \subseteq S$, a drained source $\check{s} \in R \setminus Q$ and a parameter $\delta \in \mathbb{N}_0$, a* δ-parametric dynamic transshipment instance $(\mathcal{N}^\delta, b^\delta, T)$ *consists of the following components.*

- *A δ-parametric dynamic network $\mathcal{N}^\delta$ as given in Definition 9.*
- *A δ-parametric balance function b^δ with $b^\delta(t) = b(t)$ for all terminals $t \in S \setminus \{\check{s}\}$, $b^\delta(\hat{s}) = \Delta^\delta$, and $b^\delta(\check{s}) = b(\check{s}) - \Delta^\delta$, where $\Delta^\delta := o^\delta(\hat{Q}) - o^\delta(Q)$.*

Again, a parameter value $\delta \in \mathbb{N}_0$ is *feasible* if the corresponding parametric dynamic transshipment instance $(\mathcal{N}^\delta, b^\delta, T)$ is feasible. This can be checked by minimizing the parametric submodular function $v^\delta(X) = o^\delta(X) - b^\delta(X)$. Again, we call a set $X \subseteq S$ with $v^\delta(X) < 0$ *violated*. This yields the following parametric search problem.

▶ **Definition 11** (MinimizeDelta)**.** *Given a δ-parametric dynamic transshipment instance, find the minimum feasible $\delta \in \mathbb{N}_0$.*

Again, δ^* denotes the minimum feasible parameter value. Hoppe and Tardos showed that the optimal value δ^* can be found in $\mathcal{O}(\log(T) \cdot \mathrm{SFM}(k, n, m))$ time, or in strongly polynomial time of $\tilde{\mathcal{O}}(m^4k^{14})$ using the parametric search of Megiddo [8].

The algorithm by Hoppe and Tardos performs a total of k iterations, each of which calls both subroutines MaximizeAlpha and MinimizeDelta once. Therefore, the current version of the algorithm by Hoppe and Tardos takes $\mathcal{O}\left(k \cdot \log(nTU_{\max}) \cdot \mathrm{SFM}(k, n, m)\right)$ time if both subroutines are implemented with a binary search, while Megiddo's parametric search results in a runtime of $\tilde{\mathcal{O}}(m^4k^{15})$. In the following sections, we improve the runtime of each iteration by introducing better parametric search algorithms for both subroutines.

3 Restricting the Domains of Violated Sets

Both problems MaximizeAlpha and MinimizeDelta introduced in the previous section rely on minimizing submodular functions to determine whether a parameter value is feasible. Even with state-of-the-art algorithms for SFM, this remains a computationally expensive subroutine that scales poorly with the number of terminals in the ground set.

We show that, if our parametric instance is infeasible, there always exists a violated set $X \subseteq S \cup \{\hat{s}\}$ satisfying $\hat{Q} \subset X \subset \hat{R}$, which allows us to restrict the ground sets of our parametric submodular functions v^α and v^δ. This provides a practical improvement and establishes the foundation for the following sections.

▶ **Lemma 12.** *Let $X \subseteq S \cup \{\hat{s}\}$ be a violated set for an α-parametric dynamic transshipment instance with respect to a drained source $\check{s}$. Then $\hat{s} \in X$ and $\check{s} \notin X$. The same applies to δ-parametric dynamic transshipment instances.*

Proof. We only prove the statement for α-parametric instances, as the reasoning is analogous for δ-parametric instances. Let $X \subseteq S \cup \{\hat{s}, \check{s}\}$ be an arbitrary subset of terminals. Given that we assume the transshipment instance to be feasible, we have $v(X) \geq 0$. Remember that the maximum out-flow $o^\alpha(X)$ is the value of a maximum flow over time from a super-source s^+ to a super-sink s^-, where infinite-capacity, zero-transit arcs are used to connect s^+ to every source in $S^+ \cap X$ and to connect every sink in $S^- \setminus X$ to s^-. Using this definition, we derive the following observations regarding $o(X)$:

O1 If X contains neither $\hat{s}$ nor $\check{s}$, then maximum out-flow $o^\alpha(X)$ coincides with the out-flow $o(X)$ in the original instance.

O2 If X contains $\check{s}$, then all flow traveling from s^+ to s can bypass the α-capacity arc $(\hat{s}, s)$ and move along the infinite-capacity arcs $(s^+, \check{s})$ and $(\check{s}, s)$ instead. As a consequence, we have $o^\alpha(X) = o(X)$.

We refer the reader back to Figure 1 for a visual intuition. Combining these observations with the parametric balances $b^\alpha(\hat{s}) = \Delta^\alpha$ and $b^\alpha(\check{s}) = b(\check{s}) - \Delta^\alpha$, we prove Lemma 12 by considering the following three complementary cases.

(I) If $\hat{s} \in X$ and $\check{s} \in X$, then it follows that

$$b^\alpha(X) = b^\alpha(X \setminus \{\hat{s}, \check{s}\}) + b^\alpha(\hat{s}) + b^\alpha(\check{s}) \overset{\text{Def. of } b^\alpha}{=} b^\alpha(X \setminus \{\hat{s}, \check{s}\}) + b(\check{s}) = b(X \setminus \{\hat{s}\}) = b(X).$$

Together with Observation 2 we obtain $v^\alpha(X) = v(X) \geq 0$, meaning that X cannot be a violated set.

(II) If $\hat{s} \notin X$ and $\check{s} \in X$, then it follows that $b^\alpha(X) = b(X) - \Delta^\alpha < b(X)$, which, together with Observation 2, implies that X cannot be a violated set since $v^\alpha(X) \geq v(X) \geq 0$.

(III) If $\hat{s} \notin X$ and $\check{s} \notin X$, then $b^\alpha(X) = b(X)$ and therefore X cannot be a violated set as it follows from Observation 1 that $v^\alpha(X) = o^\alpha(X) - b^\alpha(X) = o(X) - b(X) = v(X) \geq 0$.

Hence, X can only be a violated set of $(\mathcal{N}^\alpha, b^\alpha, T)$ if $\hat{s} \in X$ and $\check{s} \notin X$. ◀

Hoppe and Tardos [4] proved the property from Lemma 12 for the special case of the infeasible parameter value $\delta^* - 1$ and used it to show that there exists a set X satisfying $\hat{Q} \subset X \subset \hat{R}$ which is violated for $\delta^* - 1$ and tight for δ^*. We employ similar arguments to show that this also holds for all infeasible $\alpha, \delta \in \mathbb{N}_0$.

▶ **Lemma 13.** *Let $\alpha \in \mathbb{N}_0$ be an infeasible parameter value for MaximizeAlpha. Then there is a minimizer X of v^α with $\hat{Q} \subset X \subset \hat{R}$. Analogously, let $\delta \in \mathbb{N}_0$ be an infeasible parameter value for MinimizeDelta. Then there is a minimizer X of v^δ with $\hat{Q} \subset X \subset \hat{R}$.*

Proof. We only prove the statement for MaximizeAlpha, as the proof for MinimizeDelta is analogous. Let X^* be an arbitrary minimizer of v^α with $v^\alpha(X) < 0$. It follows from Lemma 12 that $\hat{s} \in X^*$ and $\check{s} \notin X^*$. Next, we show that the set $X \coloneqq Q \cup (X^* \cap \hat{R})$ is also a minimizer of v^α. For this, we analyze the tightness of the sets Q, $\hat{Q}$, and $\hat{R}$:

- Q was chosen to be a tight set for $(\mathcal{N}, b, T)$. This also does not change in the parametric instance as $\hat{s}, \check{s} \notin Q$ and hence $v^\alpha(Q) = v(Q) = 0$.
- $\hat{Q}$ is tight, since $b^\alpha(\hat{Q}) = b^\alpha(Q) + \Delta^\alpha \overset{Q \text{ tight}}{=} o^\alpha(Q) + o^\alpha(\hat{Q}) - o^\alpha(Q) = o^\alpha(\hat{Q})$.
- $\hat{R}$ is tight because R is tight and $\hat{s}, \check{s} \in \hat{R}$ directly imply $o^\alpha(\hat{R}) = o(R) = b(R) = b^\alpha(\hat{R})$.

Having shown that Q and $\hat{R}$ are tight sets, we study how tight sets and minimizers of v^α behave under union and intersection. For this purpose, let $Y \in \{Q, \hat{R}\}$. It follows from submodularity of v^α that

$$v^\alpha(X^* \cup Y) + v^\alpha(X^* \cap Y) \leq v^\alpha(X^*) + v^\alpha(Y) = v^\alpha(X^*) < 0. \tag{3}$$

Note that both summands on the left hand side are non-positive, since otherwise one of them would be smaller than the minimum value $v^\alpha(X^*)$. Hence, either both summands are negative, or one is equal to zero while the other equals the minimum value of v^α. In other words, exactly one of the following properties must hold:

(1) Both $X^* \cup Y$ and $X^* \cap Y$ are violated sets of v^α.
(2) Either $X^* \cup Y$ or $X^* \cap Y$ is a minimizer, while the other set is tight.

We apply this case distinction to the cases where $Y = Q$ and $Y = \hat{R}$:

- Let $Y = \hat{R}$, then Lemma 12 states that a violated set cannot contain $\check{s}$, implying that $v^\alpha(X^* \cup \hat{R}) \geq 0$ since $\check{s} \in \hat{R}$. This means that $X^* \cap \hat{R}$ is a minimizer of v^α.
- Consider $Y = Q$ and the minimizer $X^* \cap \hat{R}$. We rule out $Q \cap (X^* \cap \hat{R}) = Q \cap X^*$ as a violated set since $\hat{s} \notin Q \cap X^*$. Therefore, $Q \cup (X^* \cap \hat{R})$ is a minimizer.

Finally, observe that the minimizer $X = Q \cup (X^* \cap \hat{R})$ not only satisfies $\hat{Q} \subseteq X \subseteq \hat{R}$ because $\hat{s} \in X^* \cap \hat{R}$ but also $\hat{Q} \subset X \subset \hat{R}$ since $\hat{Q}$ and $\hat{R}$ are tight. ◀

Lemma 13 allows us to determine feasibility of a parameter value by minimizing the *restricted functions* $\tilde{v}^\alpha \colon 2^{\hat{R} \setminus \hat{Q}} \to \mathbb{Z}$ and $\tilde{v}^\delta \colon 2^{\hat{R} \setminus \hat{Q}} \to \mathbb{Z}$ with $\tilde{v}^\alpha(X) \coloneqq v^\alpha(\hat{Q} \cup X)$ and $\tilde{v}^\delta(X) \coloneqq v^\delta(\hat{Q} \cup X)$. For future use, we define the functions $\tilde{o}^\alpha$, $\tilde{o}^\delta$, $\tilde{b}^\alpha$, and $\tilde{b}^\delta$ analogously.

This brings us to the main result in this section.

▶ **Corollary 14.** *A parameter value $\alpha \in \mathbb{N}_0$ is feasible for* MaximizeAlpha *if and only if $\tilde{v}^\alpha(X) \geq 0$ for every $X \subseteq \hat{R} \setminus \hat{Q}$. Analogously, a parameter value $\delta \in \mathbb{N}_0$ is feasible for* MinimizeDelta *if and only if $\tilde{v}^\delta(X) \geq 0$ for every $X \subseteq \hat{R} \setminus \hat{Q}$.*

The obvious advantage of this result is the reduction of the domain of the submodular functions of interest to $\hat{R} \setminus \hat{Q}$. Although we have $R = S$ and $Q = \emptyset$ at the beginning, the difference becomes smaller over the course of Hoppe and Tardos' algorithm, until both sets are eventually equal. This yields a significant practical speed-up in later iterations of the algorithm. Moreover, the reduction brings further practical and theoretical advantages, which we discuss in the following section.

4 Strong Map Sequences

We concluded the previous section with a useful restriction of our submodular functions v^α and v^δ to sets X with $\hat{Q} \subset X \subset \hat{R}$, while maintaining the guarantee that our parametric instance is feasible if and only if the minimizer of our restricted function is not violated. Building on this, we show that both restricted functions satisfy the *strong map property* (also known as *decreasing differences property*).

▶ **Definition 15** (Strong Map Property [11]). *Let $f_1, f_2 \colon 2^E \to \mathbb{R}$ be two submodular functions defined over the same finite ground set E. We write $f_1 \sqsupset f_2$, or $f_2 \sqsubset f_1$, if $X \subseteq Y \subseteq E$ implies*

$$f_1(Y) - f_1(X) \leq f_2(Y) - f_2(X).$$

The relation is called a strong map. *Submodular functions $f_1, f_2, \ldots, f_k$ form a* strong map sequence *if $f_1 \sqsupset f_2 \sqsupset \cdots \sqsupset f_k$.*

Recall that the minimizers of submodular functions are closed under union and intersection, meaning that there exists a unique minimal and maximal minimizer for every submodular function. A result by Topkis [15] relates the minimal and maximal minimizers of functions that form strong map sequences.

▶ **Lemma 16** (Minimizers for Strong Map Sequences [11]). *Let $f_1, f_2 \colon 2^E \to \mathbb{R}$ be two submodular functions over the same ground set E with $f_1 \sqsubset f_2$. Denote by X_1^{min} and X_1^{max} the minimal and maximal minimizer of f_1, respectively, while X_2^{min} and X_2^{max} are the minimal and maximal minimizer of f_2, respectively. Then $X_1^{min} \subseteq X_2^{min}$ and $X_1^{max} \subseteq X_2^{max}$.*

Lemma 16 already gives an intuition for how the strong map property can be used for the parametric search problems MaximizeAlpha and MinimizeDelta: each choice of α or δ gives us a new submodular function, and, if these functions form strong map sequences, the strong map property implies that we can only encounter at most $|S|$ distinct minimal minimizers for a monotonic sequence of parameter values.

Unfortunately, neither v^α nor v^δ satisfy the strong map property as is[2]. However, this changes when considering the restricted functions $\tilde{v}^\alpha$ and $\tilde{v}^\delta$ defined in Section 3. Recall that these functions are defined over the ground set $\hat{R} \setminus \hat{Q}$, and that their definition ensures that $\hat{s}$ is implicitly added, while $\check{s}$ is excluded.

▶ **Lemma 17.** *Let $0 \leq \alpha \leq \alpha'$. Then $\tilde{v}^\alpha \sqsubset \tilde{v}^{\alpha'}$.*

Proof. Our argument is structured as follows: We first prove that $\tilde{o}^\alpha \sqsubset \tilde{o}^{\alpha+1}$ holds for all $\alpha \in \mathbb{N}_0$ and then use this result to prove that $\tilde{v}^\alpha$ also forms a strong map sequence with $\tilde{v}^\alpha \sqsubset \tilde{v}^{\alpha+1}$. The general claim $\tilde{v}^\alpha \sqsubset \tilde{v}^{\alpha'}$ then immediately follows by transitivity of $\leq$.

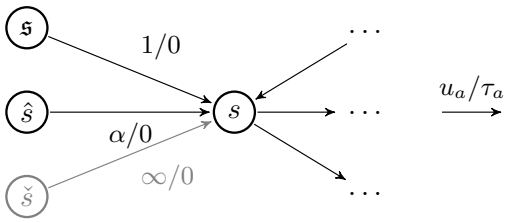


Figure 3 The auxiliary network $\mathfrak{N}$ for MaximizeAlpha.

Let $\alpha \in \mathbb{N}_0$ be arbitrary but fixed. We construct an auxiliary network $\mathfrak{N}$ with nodes $V(\mathfrak{N}) \coloneqq V(\mathcal{N}^\alpha) \cup \{\mathfrak{s}\}$ and arcs $A(\mathfrak{N}) \coloneqq A(\mathcal{N}^\alpha) \cup \{\mathfrak{a} \coloneqq (\mathfrak{s}, s)\}$, where s is the source that was replaced by $\check{s}$ in the first phase of the algorithm. For the new arc, we set $u_\mathfrak{a} = 1$ and $\tau_\mathfrak{a} = 0$. An example network for a given source $\hat{s}$ is depicted in Figure 3.

[2] In nontrivial cases, the nested sets $\emptyset$, $\{\hat{s}\}$, and $\{\hat{s}, \check{s}\}$ satisfy $o^{\alpha+1}(\{\hat{s}\}) - o^{\alpha+1}(\emptyset) > o^\alpha(\{\hat{s}\}) - o^\alpha(\emptyset)$ and $o^{\alpha+1}(\{\hat{s}, \check{s}\}) - o^{\alpha+1}(\{\hat{s}\}) < o^\alpha(\{\hat{s}, \check{s}\}) - o^\alpha(\{\hat{s}\})$. The same holds for o^δ. Consequently, our proofs for Lemma 17 and Lemma 18 fail for the general functions v^α and v^δ.

Let $\mathfrak{o}(X)$ be the max out-flow function for network $\mathfrak{N}$ restricted to the domain $\hat{R} \setminus \hat{Q} \cup \{\mathfrak{s}\}$. Notice how adding $\mathfrak{s}$ to a set of terminals $X \subseteq \hat{R} \setminus \hat{Q} \cup \{\mathfrak{s}\}$ with $\hat{s} \in X$ and $\check{s} \notin X$ is equivalent to increasing α by one, as flow cannot bypass the arcs $(\mathfrak{s}, s)$ and $(\hat{s}, s)$ through $(\check{s}, s)$. Formally, it holds for every set X with $X \subseteq \hat{R} \setminus \hat{Q}$ that

1. $\mathfrak{o}(X) = \tilde{o}^{\alpha}(X)$, and
2. $\mathfrak{o}(X \cup \{\mathfrak{s}\}) = \tilde{o}^{\alpha+1}(X)$.

Clearly, the function $\mathfrak{o}$ is also submodular. Hence, given two sets $X \subseteq Y \subset \hat{R} \setminus \hat{Q}$, the definition of submodularity in Equation (2) can be restated as

$$\mathfrak{o}(Y \cup \{\mathfrak{s}\}) - \mathfrak{o}(X \cup \{\mathfrak{s}\}) \leq \mathfrak{o}(Y) - \mathfrak{o}(X). \tag{4}$$

Combining Equation (4) with all our previous observations, it follows that the sets X and Y satisfy

$$\begin{aligned} \tilde{o}^{\alpha+1}(Y) - \tilde{o}^{\alpha+1}(X) &\overset{\text{Obs. } 2}{=} \mathfrak{o}(Y \cup \{\mathfrak{s}\}) - \mathfrak{o}(X \cup \{\mathfrak{s}\}) \\ &\overset{\text{Eq. (4)}}{\leq} \mathfrak{o}(Y) - \mathfrak{o}(X) \\ &\overset{\text{Obs. } 1}{=} \tilde{o}^{\alpha}(Y) - \tilde{o}^{\alpha}(X). \end{aligned}$$

Hence, we have shown that $\tilde{o}^{\alpha} \sqsubset \tilde{o}^{\alpha+1}$ holds for every parameter value α. Finally, recall our definition of $\tilde{v}^{\alpha}$ as $\tilde{v}^{\alpha}(X) = \tilde{o}^{\alpha}(X) - \tilde{b}^{\alpha}(X)$ for any $X \subseteq \hat{R} \setminus \hat{Q}$. Due to the strong map property of $\tilde{o}^{\alpha}$, we get for all $X \subseteq Y \subseteq \hat{R} \setminus \hat{Q}$ that

$$\begin{aligned} \tilde{v}^{\alpha+1}(Y) - \tilde{v}^{\alpha}(Y) &= \tilde{o}^{\alpha+1}(Y) - \tilde{o}^{\alpha}(Y) + \tilde{b}^{\alpha}(Y) - \tilde{b}^{\alpha+1}(Y) \\ &= \tilde{o}^{\alpha+1}(Y) - \tilde{o}^{\alpha}(Y) + \Delta^{\alpha} - \Delta^{\alpha+1} \\ &\leq \tilde{o}^{\alpha+1}(X) - \tilde{o}^{\alpha}(X) + \Delta^{\alpha} - \Delta^{\alpha+1} \\ &= \tilde{o}^{\alpha+1}(X) - \tilde{o}^{\alpha}(X) + \tilde{b}^{\alpha}(X) - \tilde{b}^{\alpha+1}(X) \\ &= \tilde{v}^{\alpha+1}(X) - \tilde{v}^{\alpha}(X), \end{aligned}$$

therefore proving that $\tilde{v}^{\alpha} \sqsubset \tilde{v}^{\alpha+1}$. Thus, by transitivity of $\leq$, our claim $\tilde{v}^{\alpha} \sqsubset \tilde{v}^{\alpha'}$ holds. ◀

Next, we utilize a similar argument to prove that $\tilde{v}^{\delta}$ forms a strong map sequence. Here, we rely on our specific definition of the δ-parametric network $\mathcal{N}^{\delta}$.

▶ **Lemma 18.** *Let $0 \leq \delta' \leq \delta$. Then $\tilde{v}^{\delta} \sqsubset \tilde{v}^{\delta'}$.*

Proof. The general approach is identical to the proof of Theorem 17. That is, we show that $\tilde{o}^{\delta}$ forms a strong map sequence with $\tilde{o}^{\delta-1} \sqsupset \tilde{o}^{\delta}$ for $\delta \geq 1$. Afterwards, we transfer this result to $\tilde{v}^{\delta-1}$ and $\tilde{v}^{\delta}$. The general claim then directly follows from the transitivity of $\leq$.

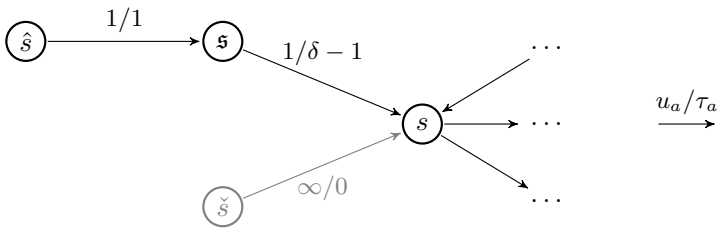


Figure 4 The auxiliary network $\mathfrak{N}$ for MinimizeDelta.

Again, let $\delta \in \mathbb{N}$ be arbitrary but fixed. We construct an auxiliary network $\mathfrak{N}$ with nodes $V(\mathfrak{N}) \coloneqq V(\mathcal{N}^\delta) \cup \{\mathfrak{s}\}$ and arcs $A(\mathfrak{N}) \coloneqq (A(\mathcal{N}^\delta) \setminus \{(\hat{s}, s)\}) \cup \{\mathfrak{a} \coloneqq (\mathfrak{s}, s), \hat{a} \coloneqq (\hat{s}, \mathfrak{s})\}$, s is the source that was replaced by $\check{s}$ in the first step of the algorithm. Additionally, let $u_{\hat{a}} = u_{\mathfrak{a}} = 1$, $\tau_{\mathfrak{a}} = \delta - 1$ and $\tau_{\hat{a}} = 1$. An example for a source $\hat{s}$ is given in Figure 4.

Again, $\mathfrak{o}(X)$ denotes the maximum out-flow function for this network restricted to the domain $\hat{R} \setminus \hat{Q} \cup \{\mathfrak{s}\}$. In this construction, we have replaced the arc $(\hat{s}, s)$ by a path consisting of arcs $(\hat{s}, \mathfrak{s})$ and $(\mathfrak{s}, s)$ with a combined transit time of δ. As a consequence, the maximum out-flow $\mathfrak{o}(X)$ remains unchanged for sets $X \subseteq \hat{R} \setminus \hat{Q}$. If $\mathfrak{s} \in X$ and $\hat{s} \in X$, on the other hand, both sources compete for access to the arc $(\mathfrak{s}, s)$. Then, the flow out of X is maximized by sending all flow from $\mathfrak{s}$. Formally, it holds for every set $X \subseteq \hat{R} \setminus \hat{Q}$ that

1. $\mathfrak{o}(X) = \tilde{o}^\delta(X)$, and
2. $\mathfrak{o}(X \cup \{\mathfrak{s}\}) = \tilde{o}^{\delta-1}(X)$

These observations plus the second definition of submodularity from Equation (2) imply for all sets $X \subseteq Y \subseteq \hat{R} \setminus \hat{Q}$ that

$$\begin{aligned}\tilde{o}^{\delta-1}(Y) - \tilde{o}^{\delta-1}(X) &\overset{\text{Obs. } 2}{=} \mathfrak{o}(Y \cup \{\mathfrak{s}\}) - \mathfrak{o}(X \cup \{\mathfrak{s}\}) \\ &\overset{\text{Eq. (2)}}{\leq} \mathfrak{o}(Y) - \mathfrak{o}(X) \\ &\overset{\text{Obs. } 1}{=} \tilde{o}^\delta(Y) - \tilde{o}^\delta(X).\end{aligned}$$

Therefore, the function $\tilde{o}^\delta$ forms a strong map sequence with $\tilde{o}^{\delta-1} \sqsupset \tilde{o}^\delta$. Finally, by the same arguments as in the proof of Lemma 17, the parametric function $\tilde{v}^\delta$ forms a strong map sequence with $\tilde{v}^\delta \sqsupset \tilde{v}^{\delta'}$ as claimed. ◀

In the following section, we use the strong map property and the resulting nested sequences of minimizers to construct new parametric search algorithms for v^α and v^δ .

5 An Improved Parametric Search

In this section, we adapt Schlöter's [11] parametric search algorithm for the single-source or single-sink quickest transshipment problem to MaximizeAlpha and MinimizeDelta. We start by showing that $\tilde{v}^\alpha$ and $\tilde{v}^\delta$ are monotonic in their respective parameters.

▶ **Lemma 19.** *Given a set of terminals $X \subseteq \hat{R} \setminus \hat{Q}$, both maps $\alpha \mapsto \tilde{v}^\alpha(X)$ and $\delta \mapsto \tilde{v}^\delta(X)$ are monotonic in the parameters α and δ, respectively. That is, we have $\tilde{v}^\alpha(X) \geq \tilde{v}^{\alpha+1}(X)$ and $\tilde{v}^\delta(X) \leq \tilde{v}^{\delta+1}(X)$ for all $\alpha, \delta \in \mathbb{N}_0$.*

Proof. Let $\alpha, \delta \in \mathbb{N}_0$ be arbitrary but fixed parameter values. We set $Z \coloneqq X \cup \hat{Q}$ and decompose $\tilde{v}^\alpha(X)$ (and, analogously, $\tilde{v}^\delta(X)$) as

$$\begin{aligned}\tilde{v}^\alpha(X) &= \tilde{o}^\alpha(X) - \tilde{b}^\alpha(X) \\ &= \tilde{o}^\alpha(X) - \big(b(Z \setminus \{\hat{s}\}) + \Delta^\alpha\big) \\ &\overset{\text{Def. } 7}{=} \tilde{o}^\alpha(X) - \big(b(Z \setminus \{\hat{s}\}) + o^\alpha(\hat{Q}) - o^\alpha(Q)\big) \\ &= \tilde{o}^\alpha(X) - \big(b(Z \setminus \{\hat{s}\}) + \tilde{o}^\alpha(\emptyset) - o^\alpha(Q)\big) \\ &= \tilde{o}^\alpha(X) - \tilde{o}^\alpha(\emptyset) + \big(o^\alpha(Q) - b(Z \setminus \{\hat{s}\})\big).\end{aligned} \tag{5}$$

Observe that we can treat the term $o^\alpha(Q) - b(Z \setminus \{\hat{s}\})$ as constant: since $\hat{s}, \check{s} \notin Q$, the out-flow $o^\alpha(Q)$ is identical for all $\alpha \in \mathbb{N}_0$ (or $\delta \in \mathbb{N}_0$). Similarly, the value of $b(Z \setminus \{\hat{s}\})$ does not depend on α or δ.

We now prove monotonicity using this simplification. For MaximizeAlpha, the monotonicity $\tilde{v}^\alpha(X) \geq \tilde{v}^{\alpha+1}(X)$ holds if and only if $\tilde{o}^\alpha(X) - \tilde{o}^\alpha(\emptyset) \geq \tilde{o}^{\alpha+1}(X) - \tilde{o}^{\alpha+1}(\emptyset)$ is true, which follows directly from the strong map property $\tilde{o}^\alpha \sqsubset \tilde{o}^{\alpha+1}$ shown in the proof of Lemma 17. Similarly, replacing α with δ in Equation (5), it follows from $\tilde{o}^\delta \sqsupset \tilde{o}^{\delta+1}$ that $\tilde{o}^\delta(X) - \tilde{o}^\delta(\emptyset) \geq \tilde{o}^{\delta+1}(X) - \tilde{o}^{\delta+1}(\emptyset)$ and thus $\tilde{v}^\delta(X) \leq \tilde{v}^{\delta+1}(X)$ holds. ◀

The monotonicity and strong map property of $\tilde{v}^\alpha$ and $\tilde{v}^\delta$ allow us to introduce new parametric search algorithms for MaximizeAlpha (cf. Algorithm 1) and MinimizeDelta (cf. Algorithm 2). In their core, they follow a rather simple approach similar to algorithms by Schlöter, Skutella, and Tran [12, 11]: we start with an infeasible parameter value α_1 or δ_1 and a corresponding minimizer X_1 of $\tilde{v}^{\alpha_1}$ or $\tilde{v}^{\delta_1}$. Possible initial values are $\alpha_1 \coloneqq \alpha_{\max} = nU_{\max}$ and $\delta_1 \coloneqq 0$ [4]. Next, we alternate between two steps *jump* and *check* until the current minimizer X_i is no longer violated. In the *jump* step, we compute the largest parameter value α_{i+1} or the smallest parameter value δ_{i+1} such that the previous minimizer X_i is no longer violated. Note that this step is, in essence, an integral parametric min-cost flow problem with one parametrized arc. Afterwards, the *check* step finds a minimizer X_{i+1} for the new value α_{i+1} and δ_{i+1}.

In the check step, we use the results of the previous sections in order to restrict the search for minimizers to $X_{i+1} \subset X_i$. We will show in the following that this restriction is actually feasible. Note that the restriction not only reduces the search space in each iteration to $|X_i|$ terminals, but also limits MaximizeAlpha and MinimizeDelta to at most $|S|$ iterations, since the length of the chain $X_1 \supset X_2 \supset \dots$ is limited by $|\hat{R} \setminus \hat{Q}| \leq |S|$.

▶ **Theorem 20.** *Algorithm 1 computes the maximum feasible $\alpha^* \in \mathbb{N}_0$ for MaximizeAlpha; Algorithm 2 computes the minimum feasible $\delta^* \in \mathbb{N}_0$ for MinimizeDelta. Both algorithms terminate after at most $|S|$ iterations of the while loop.*

Proof. We first show by induction that if α_i is infeasible, then there exists a minimizer X_i of $\tilde{v}^{\alpha_i}$ with $X_i \subset X_{i-1}$, where $X_0 = \hat{R} \setminus \hat{Q}$.

By definition, X_1 is a minimizer of $\tilde{v}^{\alpha_1}$. Let α_i and α_{i+1} be infeasible, and assume that X_i is a minimizer of $\tilde{v}^{\alpha_i}$. We have $\alpha_{i+1} < \alpha_i$, since otherwise $\tilde{v}^{\alpha_{i+1}}(X_i) \overset{\text{Lem. 19}}{\leq} \tilde{v}^{\alpha_i}(X_i) < 0$ contradicts the choice of α_{i+1} as feasible for X_i. Together with Lemma 17, this implies the relation $\tilde{v}^{\alpha_i} \sqsupset \tilde{v}^{\alpha_{i+1}}$. Therefore, we have $X_{i+1}^{\min} \subseteq X_i^{\min} \subseteq X_i$ due to Lemma 16, with $X_j^{\min}$ being the minimal minimizer of $\tilde{v}^{\alpha_j}$ for $j \in \{i, i+1\}$. Since α_{i+1} is infeasible with $\tilde{v}^{\alpha_{i+1}}(X_{i+1}^{\min}) < 0$, and, by definition, $\tilde{v}^{\alpha_{i+1}}(X_i) \geq 0$, we even have $X_{i+1}^{\min} \subset X_i$. Hence, $X_{i+1}^{\min}$ is a feasible choice for X_{i+1} in Algorithm 1.

Algorithm 1 Parametric Search for MaximizeAlpha.

Data: Tight sets $\hat{Q} \subset \hat{R} \subseteq S \cup \{\hat{s}\}$, submodular function $\tilde{v}^\alpha \colon 2^{\hat{R} \setminus \hat{Q}} \to \mathbb{Z}$, infeasible upper bound $\alpha_{\max}$
Result: Maximum feasible $\alpha \in \mathbb{N}_0$

```
α_1 ← α_max, X_1 ← Minimizer of ṽ^{α_1}
i ← 1
while ṽ^{α_i}(X_i) < 0 do
    α_{i+1} ← Maximum α ≥ 0 with ṽ^α(X_i) ≥ 0        ▷ Jump step
    X_{i+1} ← Minimizer X ⊂ X_i of ṽ^{α_{i+1}}       ▷ Check step
    i ← i + 1
end
return α_i
```

Algorithm 2 Parametric Search for MinimizeDelta.

Data: Tight sets $\hat{Q} \subset \hat{R} \subseteq S \cup \{\hat{s}\}$, submodular function $\tilde{v}^\delta \colon 2^{\hat{R}\setminus\hat{Q}} \to \mathbb{Z}$
Result: Minimum feasible $\delta \in \mathbb{N}$
$\delta_1 \leftarrow 0$, $X_1 \leftarrow$ Minimizer of $\tilde{v}^{\delta_1}$
$i \leftarrow 1$
while $\tilde{v}^{\delta_i}(X_i) < 0$ **do**
  $\delta_{i+1} \leftarrow$ Minimum $\delta \geq 0$ with $\tilde{v}^{\delta}(X_i) \geq 0$ ▷ Jump step
  $X_{i+1} \leftarrow$ Minimizer $X \subset X_i$ of $\tilde{v}^{\delta_{i+1}}$ ▷ Check step
  $i \leftarrow i + 1$
end
return δ_i

Recall that Algorithm 1 terminates after at most $|S|$ iterations. Let α_{i^*} be the value returned by the algorithm, and let X_{i^*} be the minimizer generated in the final iteration i^*. If α_{i^*} were not feasible, then X_{i^*} would be a minimizer with $v^{\alpha_{i^*}}(X_{i^*}) < 0$. However, since the algorithm terminates, it follows that α_{i^*} is feasible.

To see that α_{i^*} is maximum, recall that $\alpha_1 = \alpha_{\max}$ is infeasible, and thus the jump step is executed at least once. By construction in the jump, the value α_{i^*} is maximum such that the previous minimizer X_{i^*-1} is no longer violated. Hence, for any $\alpha > \alpha_{i^*}$, we have $v^{\alpha}(X_{i^*-1}) < 0$, so α is infeasible. Therefore, α_{i^*} is the optimal solution to MaximizeAlpha.

The proof for Algorithm 2 is analogous. ◀

The remainder of the section is devoted to the runtime analysis. Our main improvement is due to the upper bound $k = |S|$ on the number of iterations the algorithms execute.

▶ **Theorem 21.** *Algorithm 1 can be implemented to solve MaximizeAlpha in strongly polynomial time of $\mathcal{O}(k[\mathrm{SFM}(k,n,m) + \mathrm{MCF}(n,m)^2])$ and in weakly polynomial time of $\mathcal{O}(k[\mathrm{SFM}(k,n,m) + \log(nU_{max}) \cdot \mathrm{MCF}(n,m)])$.*

Proof. The runtime is determined by the two main steps performed in each iteration:

- Jump can be implemented using Megiddo's parametric search [8] or binary search over the range $[0, nU_{\max}]$ in conjunction with a minimum cost flow algorithm. The former results in a runtime of $\mathcal{O}(\mathrm{MCF}(n,m)^2)$, the latter in $\mathcal{O}(\log(nU_{\max}) \cdot \mathrm{MCF}(n,m))$.
- Check minimizes the submodular function v^α on the restricted domain. In the worst case, this takes $\mathcal{O}(\mathrm{SFM}(k,n,m))$ time.

Together with the upper bound of k on the iterations of the while loop, the final runtime is $\mathcal{O}(k[\mathrm{SFM}(k,n,m) + \mathrm{MCF}(n,m)^2])$ or $\mathcal{O}(k[\mathrm{SFM}(k,n,m) + \log(nU_{\max}) \cdot \mathrm{MCF}(n,m)])$. ◀

We obtain an analogous runtime for Algorithm 2.

▶ **Theorem 22.** *Algorithm 2 can be implemented to solve MinimizeDelta in strongly polynomial time of $\mathcal{O}(k[\mathrm{SFM}(k,n,m) + \mathrm{MCF}(n,m)^2])$ and in weakly polynomial time of $\mathcal{O}(k[\mathrm{SFM}(k,n,m) + \log(T) \cdot \mathrm{MCF}(n,m)])$.*

Proof. The proof is analogous to that of Theorem 21. The binary search for the minimum feasible value of δ is done on the range $[0, T]$, which yields the different logarithmic term. ◀

We conclude this section with an improved runtime for computing integral dynamic transshipments compared to the state-of-the-art runtime of $\tilde{\mathcal{O}}(m^4k^{15})$.

▶ **Theorem 23.** *Given a dynamic transshipment instance $(\mathcal{N}, b, T)$, an integral quickest transshipment can be computed in $\tilde{\mathcal{O}}(m^2k^5 + m^4k^2)$ time.*

Proof. Hoppe and Tardos [4] already proved that their algorithm for dynamic transshipment terminates after $\mathcal{O}(k)$ iterations, each of which consists of one call of Algorithms 1 and 2. Given the strongly polynomial runtime of $\mathcal{O}\big(k(\mathrm{SFM}(k,n,m) + \mathrm{MCF}(n,m)^2)\big)$ for both subroutines, we obtain a worst-case complexity of $\mathcal{O}(k^2(\mathrm{SFM}(k,n,m) + \mathrm{MCF}(n,m)^2))$.

Suppressing the polylogarithmic terms, we have a time complexity of $\tilde{\mathcal{O}}(m^2k^3)$ for $\mathcal{O}(\mathrm{SFM}(k,n,m))$ and of $\tilde{\mathcal{O}}(m^2)$ for $\mathcal{O}(\mathrm{MCF}(n,m))$. Overall, we obtain an improved runtime of $\tilde{\mathcal{O}}(m^2k^5 + m^4k^3)$ time for the integral dynamic transshipment problem.

In order to compute a quickest transshipment, we have to determine the minimum time horizon first. This can be done by the method by Schlöter, Skutella, and Tran [12] in $\tilde{\mathcal{O}}(m^2k^5 + m^3k^3 + m^3n)$ time. This runtime is dominated by that of our algorithm for computing the corresponding transshipment: since we assume that the network is connected, we have $m \geq n \geq k$, and thus $\tilde{\mathcal{O}}(m^3n) \subset \tilde{\mathcal{O}}(m^4)$ and $\tilde{\mathcal{O}}(m^3k^3) \subset \tilde{\mathcal{O}}(m^4k^2)$. Overall, the time required to compute a quickest integral transshipment is

$$\tilde{\mathcal{O}}\big((m^2k^5 + m^3k^3 + m^3n) + (m^2k^5 + m^4k^2)\big) = \tilde{\mathcal{O}}(m^2k^5 + m^4k^2 + m^4) = \tilde{\mathcal{O}}(m^2k^5 + m^4k^2),$$

which shows Theorem 23. ◀

6 Conclusion and Outlook

In this paper, we propose an improved version of the algorithm by Hoppe and Tardos for the integral quickest transshipment problem. Our approach is based on more efficient parametric search algorithms using the strong map property and yields a substantial reduction of the runtime from $\tilde{\mathcal{O}}(m^4k^{15})$ to $\tilde{\mathcal{O}}(m^2k^5 + m^4k^2)$.

Our findings open room for ensuing research. In particular, the restrictions of submodular functions to suitable domains introduced in this paper may provide even better bounds on the runtime for our algorithms. Furthermore, we see potential improvements to the jump steps in Algorithms 1 and 2 that are currently based on Megiddo's parametric search and contribute a factor of $\tilde{\mathcal{O}}(m^4)$ to the runtime. This factor constitutes the remaining gap in runtime between the integral and the fractional quickest transshipment problem. In order to close this gap, future studies may focus on adapting parametric minimum cost flow algorithms akin to the algorithms by Lin and Jaillet [7] and Saho and Shigeno [10].

References

1 Mariia Anapolska, Dario van den Boom, Christina Büsing, and Timo Gersing. A faster parametric search for the integral quickest transshipment problem. *arXiv preprint arXiv:2505.12975*, 2025. doi:10.48550/arXiv.2505.12975.

2 Rainer E. Burkard, Karin Dlaska, and Bettina Klinz. The quickest flow problem. *Zeitschrift für Operations Research*, 37(1):31–58, 1993. doi:10.1007/BF01415527.

3 Lester R. Ford and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

4 Bruce Hoppe and Éva Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25(1):36–62, 2000. doi:10.1287/moor.25.1.36.15211.

5 Satoru Iwata and James B Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 1230–1237. SIAM, 2009. doi:10.1137/1.9781611973068.133.

6 Yin Tat Lee, Aaron Sidford, and Sam Chiu-Wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1049–1065. IEEE, 2015. doi:10.1109/FOCS.2015.68.

7 Maokai Lin and Patrick Jaillet. On the quickest flow problem in dynamic networks–a parametric min-cost flow approach. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1343–1356. SIAM, SIAM, 2015. doi:10.1137/1.9781611973730.89.

8 Nimrod Megiddo. Combinatorial optimization with rational objective functions. In *Tenth annual ACM symposium on Theory of computing (STOC)*, pages 1–12, 1978. doi:10.1145/800133.804326.

9 James Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Twentieth annual ACM symposium on Theory of Computing (STOC)*, pages 377–387, 1988. doi:10.1145/62212.62249.

10 Masahide Saho and Maiko Shigeno. Cancel-and-tighten algorithm for quickest flow problems. *Networks*, 69(2):179–188, 2017. doi:10.1002/net.21726.

11 Miriam Schlöter. *Flows over time and submodular function minimization.* PhD thesis, Dissertation, Berlin, Technische Universität Berlin, 2018, 2018.

12 Miriam Schlöter, Martin Skutella, and Khai Van Tran. A faster algorithm for quickest transshipments via an extended discrete newton method. In *2022 Symposium on Discrete Algorithms (SODA)*, pages 90–102. SIAM, 2022. doi:10.1137/1.9781611977073.5.

13 Martin Skutella. An introduction to network flows over time. *Research Trends in Combinatorial Optimization: Bonn 2008*, pages 451–482, 2009. doi:10.1007/978-3-540-76796-1_21.

14 Martin Skutella. An introduction to transshipments over time. *arXiv preprint arXiv:2312.04991*, 2023. doi:10.48550/arXiv.2312.04991.

15 Donald M. Topkis. Minimizing a Submodular Function on a Lattice. *Operations Research*, 26(2):305–321, 1978. doi:10.1287/opre.26.2.305.

Bootstrapping Dynamic APSP via Sparsification

Rasmus Kyng
ETH Zurich, Switzerland

Simon Meierhans
ETH Zurich, Switzerland

Gernot Zöcklein
ETH Zurich, Switzerland

Abstract

We give a simple algorithm for the dynamic approximate All-Pairs Shortest Paths (APSP) problem. Given a graph $G = (V, E, \boldsymbol{l})$ with polynomially bounded edge lengths, our data structure processes $|E|$ edge insertions and deletions in total time $|E|^{1+o(1)}$ and provides query access to $|E|^{o(1)}$-approximate distances in time $\widetilde{O}(1)$ per query.

We produce a data structure that mimics Thorup-Zwick distance oracles [17], but is dynamic and deterministic. Our algorithm selects a small number of pivot vertices. Then, for every other vertex, it reduces distance computation to maintaining distances to a small neighborhood around that vertex and to the nearest pivot. We maintain distances between pivots efficiently by representing them in a smaller graph and recursing. We maintain these smaller graphs by (a) reducing vertex count using the dynamic distance-preserving core graphs of Kyng-Meierhans-Probst Gutenberg [16] in a black-box manner and (b) reducing edge-count using a dynamic spanner akin to Chen-Kyng-Liu-Meierhans-Probst Gutenberg [6]. Our dynamic spanner internally uses an APSP data structure. Choosing a large enough size reduction factor in the first step allows us to simultaneously bootstrap a spanner and a dynamic APSP data structure. Notably, our approach does not need expander graphs, an otherwise ubiquitous tool in derandomization.

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

Keywords and phrases Dynamic Graph Algorithms, Spanners, Vertex Sparsification, Bootstrapping

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.113

Related Version *Full Version*: `https://arxiv.org/abs/2408.11375`

Funding *Rasmus Kyng*: The research leading to these results has received funding from grant no. 200021 204787 and from the starting grant "A New Paradigm for Flow and Cut Algorithms" (no. TMSGI2_218022) of the Swiss National Science Foundation.
Simon Meierhans: The research leading to these results has received funding from grant no. 200021 204787 of the Swiss National Science Foundation. Simon Meierhans is supported by a Google PhD Fellowship.

1 Introduction

The dynamic *All-Pairs Shortest Paths* (APSP) problem asks to maintain query access to pairwise distances for a dynamic graph $G = (V, E)$. Recently, the setting of maintaining crude distance estimates at low computational cost has received significant attention [15, 5, 11, 9, 8, 2, 12, 13] culminating in deterministic algorithms with sub-polynomial update and query time for graphs receiving both edge insertions and deletions [10, 16, 14]. In contrast to other deterministic approaches, [16] allows efficient *implicit* access to approximate shortest paths via a small collection of dynamic trees. This is crucial for applications to decremental single source shortest paths and incremental minimum cost flows [6].


© Rasmus Kyng, Simon Meierhans, and Gernot Zöcklein;
licensed under Creative Commons License CC-BY 4.0



All existing dynamic approximate APSP data structures that work against adaptive adversaries are highly complex [10, 16, 14].

The algorithm [16] establishes the following *Thorup-Zwick-distance oracle*-inspired framework: To maintain approximate APSP in the graph, we select a small (dynamic) set of pivot vertices. For every other vertex, we can reduce distance computation to (a) computing distances in a small neighborhood around the vertex and (b) computing the distance to the nearest pivot vertex. Then, [16] maintains a smaller graph where distances between the pivot vertices can be computed, and recursively maintains these with the same approach. To maintain this smaller graph, [16] designs a dynamic vertex sparsification procedure and performs edge sparsification using the dynamic spanner of [7, 4]. This spanner internally relies on complex data structures for decremental APSP on expanders [9] and uses expander embedding techniques in a non-trivial way. We will use the overall dynamic Thorup-Zwick approach of [16], and we will maintain a small graph for computing distances between pivots in a simpler way.

A key ingredient for us is a new, more powerful fully-dynamic spanner from [6]. This spanner requires both dynamic APSP and expander embedding techniques to maintain. In this work, we observe that this spanner can be simplified so that it only requires dynamic APSP to maintain. Thus, given a dynamic APSP data structure, we can maintain a fully-dynamic spanner. But, [16] also showed that given a fully-dynamic spanner, we can obtain dynamic APSP. This presents a striking opportunity: we can simultaneously bootstrap a dynamic APSP data structure and spanner. This gives our main result, a dynamic approximate APSP data structure.

▶ **Theorem 1.** *For a graph $G = (V, E, \boldsymbol{l})$ with polynomially bounded integral edge lengths $\boldsymbol{l} \in \mathbb{R}^E_{>0}$ undergoing up to m edge updates (insertions and deletions), there is an algorithm with total update time $m^{1+o(1)}$ that maintains query access to approximate distances $\widetilde{\text{dist}}(u, v)$ such that*

$$\text{dist}(u, v) \leq \widetilde{\text{dist}}(u, v) \leq m^{o(1)} \cdot \text{dist}(u, v).$$

The query time is $\widetilde{O}(1)$.

While we present our theorem in the setting of processing m updates in time $m^{1+o(1)}$ to simplify the presentation, we believe that this algorithm can be extended directly to the worst-case update time setting of [16]. Furthermore, the witness paths can be implicitly represented as forest paths as in [16], which enables the cheap flow maintenance via dynamic tree data structures necessary for applications.

1.1 Roadmap

We first define the necessary preliminaries in Section 2. Then, we present a simplified version of our algorithm as a warm-up in Section 3. Afterwards, we present our bootstrapped algorithm based on the KMG-vertex sparsifier [16] and a dynamic spanner in Section 4.

2 Preliminaries

2.1 Graph Notation

We use $G = (V, E, \boldsymbol{l})$ to refer to a graph G with vertex set V, edge set E and length vector $\boldsymbol{l} \in \mathbb{R}^E_{\geq 1}$. We will restrict our attention to polynomially bounded integral lengths, and will denote their upper bound by L.

2.2 Paths, Distances and Balls

For two vertices u, v, we let $\pi^G_{u,v}$ denote the shortest path from u to v in G (lowest sum of edge lengths). Furthermore, we let $\text{dist}_G(u, v)$ be the length of said path.

We let $B_G(u, v) \stackrel{\text{def}}{=} \{w : \text{dist}(u, w) < \text{dist}(u, v)\}$ and $\overline{B}_G(u, v) \stackrel{\text{def}}{=} \{w : \text{dist}(u, w) \leq \text{dist}(u, v)\}$. Then, for a set $A \subseteq V$ we let $B_G(u, A) \stackrel{\text{def}}{=} \bigcup_{a\in A} B_G(u, a)$ and $\overline{B}_G(u, A) \stackrel{\text{def}}{=} \bigcup_{a\in A} \overline{B}_G(u, a)$ respectively.

Finally, we define the neighborhood of a vertex u as $\mathcal{N}(u) \stackrel{\text{def}}{=} \{v : (u, v) \in E\} \cup \{u\}$ and of a set $A \subseteq V$ as $\mathcal{N}(A) \stackrel{\text{def}}{=} \bigcup_{a\in A} \mathcal{N}(a)$. For a vertex v, we let $E_G(v) \stackrel{\text{def}}{=} \{(v, u) : u \in \mathcal{N}(v)\} \subseteq E(G)$.

When it is clear from the context, we sometimes omit the subscript/superscript G.

▶ **Definition 2** (Spanner). *Given a graph $G = (V, E, \boldsymbol{l})$, we say that a subgraph $H \subseteq G$ is an α-spanner of G if for any $u, v \in V$ we have $\text{dist}_G(u, v) \leq \text{dist}_H(u, v) \leq \alpha \cdot \text{dist}_G(u, v)$.*

2.3 Dynamic Graphs

Vertex splits are a crucial operation in our algorithms. They arise since we contract graph components and represent them as vertices. When such a component needs to be divided, this naturally corresponds to a vertex split defined as follows.

▶ **Definition 3** (Vertex Split). *For a graph $G = (V, E, \boldsymbol{l})$, a vertex split takes a vertex $v \in V$ and a set $U \subseteq \mathcal{N}(v)$, and replaces the vertex v with two vertices v', v'' such that $\mathcal{N}(v') = U$ and $\mathcal{N}(v'') = \mathcal{N}(v) \setminus U$.*

▶ **Definition 4** (Master Nodes). *Initially, each $v \in V(G^{(0)})$ is associated with a unique master node $\text{master}(v) = \{v\}$. Then, at any time t, if some $v \in V(G^{(t)})$ is split into v' and v'', we set $\text{master}(v') = \text{master}(v'') = \{v\} \cup \text{master}(v)$.*

We can also define our notion of fully-dynamic graphs, which includes vertex splits as an operation.

▶ **Definition 5** (Fully-Dynamic Graph). *A fully-dynamic graph is a graph undergoing edge insertions/deletions, vertex splits and isolated vertex insertions.*

Whenever we disallow vertex splits, we refer to edge-dynamic graphs instead.

▶ **Definition 6** (Edge-Dynamic Graph). *An edge-dynamic graph is a graph undergoing edge insertions/deletions and isolated vertex insertions.*

2.4 Dynamic Objects and Recourse

For a dynamic object $(F(t))_{t\in[T]}$ defined on an index set $[T] = \{1, \ldots, T\}$, we refer to $F^{(T')} \stackrel{\text{def}}{=} (F(t))_{t\in[T']}$ as the full sequence up to time T'. We can then apply functions $g(\cdot)$ that operate on sequences to $F^{(T')}$, e.g. $g(F^{(T')})$. This is for example very useful to keep track of how an object changes over time. By providing a difference function $\triangle(F(t), F(t-1))$ for a dynamic object F, one may define the recourse as the sum of differences $\left\|F^{(t)}\right\|_{\rightarrow} \stackrel{\text{def}}{=} \sum_{i=0}^{t-1} \triangle(F(i+1), F(i))$.

For functions f that operate on a static object G instead of on a sequence, we overload the notation by saying $f(G^{(t)})$ is simply f applied to the last object in the sequence, that is, $f(G^{(t)}) = f(G(t))$. For example, $\deg_{\max}(G^{(t)})$ is defined to be the maximum degree of the graph G at time t.

Whenever we make a statements about a dynamic object F without making reference to a specific time t, we implicitly mean that the statement holds at all times. For example, $\|H\|_{\to} \leq \|G\|_{\to}$ formally means for all $t : \left\|H^{(t)}\right\|_{\to} \leq \left\|G^{(t)}\right\|_{\to}$. Meanwhile, $|E(H)| \leq |E(G)|$ formally means for all $t : |E(H^{(t)})| \leq |E(G^{(t)})|$, and by our convention for applying functions of static objects to a dynamic object, we have that $|E(H^{(t)})| \leq |E(G^{(t)})|$ means $|E(H(t))| \leq |E(G(t))|$.

The time scales we use are fine-grained as we will consider multiple dynamic objects that change more frequently than the input graph G does. We will choose an index set for the dynamic sequence which is fine-grained enough to capture changes to all the objects relevant to a given proof.

A dynamic object H might not change in a time span in which another object F changes multiple times. In this case, to have sequences that are indexed by the same set, we simply pad the sequence of H with the same object, so that both dynamic objects would share the same index set T.

For fully-dynamic graphs G, we let $\triangle(G(t), G(t-1))$ denote the number of edge insertions, edge deletions, vertex splits and isolated vertex insertions that happened between $G(t)$ and $G(t-1)$. Note that generally, vertex splits cannot be implemented in constant time, and thus recourse bounds are not directly relevant to bounding running time. Nevertheless, this convention for measuring recourse is useful, as it is tailored precisely to the type of recourse bounds we prove.

We can store anything that occurs at various times throughout our algorithm in a dynamic object and define what recourse means for it. For example, we will repeatedly call a procedure ReduceDegree(X). If at time t we call the function with input $X(t)$, we might want to track the quantity $\sum_{t' \leq t} |X(t')|$, so we simply let R_X be the dynamic object such that $R_X(t) = \sum_{t' \leq t} |X(t')|$. By defining the difference $\triangle(R_X(t), R_X(t-1)) = R_X(t) - R_X(t-1)$, we can find out how much the value changed during two points in time. A statement such as $\|H\|_{\to} \leq \|G\|_{\to} + \|R_X\|_{\to}$ then means that at any time t (suitably fine-grained), the total amount of changes that H underwent are not more than the amount of G, plus some extra number of updates determined by the calls to ReduceDegree$(\cdot)$ made up to that time.

By viewing the run-time of dynamic algorithm as evolving over a sequence of updates and thinking of it as a dynamic object, we also give meaning to the statement, say, that an algorithm runs in time $n + \gamma \|G\|_{\to}$.

2.5 Dynamic APSP

We introduce APSP data structures for edge-dynamic graphs.

▶ **Definition 7** (Dynamic APSP). *For an edge-dynamic graph $G = (V, E, \boldsymbol{l})$, a γ-approximate β-query APSP data structure supports the following operations.*

- AddEdge(u, v) / RemoveEdge(u, v)*: Add/Remove edge (u, v) from/to G.*
- Dist(u, v)*: Returns in time β a distance estimate $\widetilde{\text{dist}}(u, v)$ such that at all times*

$$\text{dist}_G(u, v) \leq \widetilde{\text{dist}}(u, v) \leq \gamma \cdot \text{dist}_G(u, v).$$

- Path(u, v)*: Returns a uv-path P in G of length $|P| \leq$ Dist(u, v) in time $\beta \cdot |P|$.*

Given an APSP data structure, we let $APSP(|E|, \Delta, u)$ denote the total run time of initialization and u calls to AddEdge and RemoveEdge on an edge-dynamic graph with $\deg_{\max}(G) \leq \Delta$ and $|E|$ initial edges.

We sometimes also call γ the (worst-case) stretch of the data structure.

2.6 Degree Reduction Through Binary Search Trees

Given an *edge-dynamic* graph G, we can maintain in time $\widetilde{O}(1) \cdot \|G\|_{\to}$ another edge-dynamic graph G' in which we replace vertices by Binary Search Trees with one leaf node per edge of the vertex in the original graph. It is immediate that if we can find shortest paths in G', we can translate them into shortest paths in G. The advantage of this technique is that while $|E(G')| = O(|E(G)|)$, $|V(G')| = O(|E(G)|)$ and $\|G'\|_{\to} = \widetilde{O}(\|G\|_{\to})$, we have $\deg_{\max}(G') \leq 3$.

3 Warm up: Low-Recourse Thorup-Zwick'05

Before we describe our algorithm, we turn our attention to a much more modest goal that still showcases the main ideas used in our full construction: maintaining a relaxed version of Thorup-Zwick pivots [17] in a fully-dynamic graph with low recourse. This allows us to do away with the careful controlling of maximum degrees in intermediate graphs, which is the root cause of most of the technicalities when making the construction computationally efficient. Furthermore, we do not need to bootstrap our data structure in this setting.

3.1 Pivot Hierarchies

We first introduce pivot hierarchies.

▶ **Definition 8** (Pivot Hierarchy). *For a graph $G = (V, E, \boldsymbol{l})$, we let a pivot hierarchy of depth Λ and cluster size γ be*

- *a collection of vertex sets $\mathcal{C} = A_0, \ldots, A_\Lambda$ with $A_i \subseteq V$ for all i, $A_0 = V$ and $|A_\Lambda| = 1$,*
- *pivot maps $p_i : A_i \to A_{i+1}$ for all $0 \leq i < \Lambda$ and*
- *cluster maps $C_i : A_i \to 2^{A_i}$ for all $0 \leq i < \Lambda$ such that for all $u \in A_i$ $|C_i(u)| \leq \gamma$ and $u \in C_i(u)$.*

Pivot maps should be thought of as mapping vertices to smaller and smaller sets of representatives, and the cluster maps should be thought of as small neighborhoods of pivots for which (approximate) distances are stored. In classical Thorup-Zwick, the cluster $C_i(v)$ consists of all vertices in A_i that are closer to v than $p_i(v)$. In our algorithms, these balls become somewhat distorted due to sparsification. We then define the approximate distances maintained by the hierarchy.

▶ **Definition 9** (Distances). *For a pivot hierarchy of depth Λ, we define $\widetilde{\operatorname{dist}}^{(\Lambda)}(u, v) \stackrel{\text{def}}{=} 0$ for $u, v \in A_\Lambda$ and*

$$\widetilde{\operatorname{dist}}^{(i)}(u, v) \stackrel{\text{def}}{=} \begin{cases} \operatorname{dist}_G(u, v) & \text{if } u \in C_i(v) \\ \operatorname{dist}_G(u, p_i(u)) + \widetilde{\operatorname{dist}}^{(i+1)}(p_i(u), p_i(v)) + \operatorname{dist}_G(p_i(v), v) & \text{otherwise} \end{cases}.$$

We let $\widetilde{\operatorname{dist}}(u, v) \stackrel{\text{def}}{=} \widetilde{\operatorname{dist}}^{(0)}(u, v)$ for $u, v \in V$ and call a pivot hierarchy α distance preserving if $\widetilde{\operatorname{dist}}(u, v) \leq \alpha \cdot \operatorname{dist}_G(u, v)$ for all $u, v \in V$.

We note that efficient distance queries can be implemented as long as the distances within clusters and the distances to pivots are efficiently maintained. In this warm up, we only focus on maintaining the collection $\mathcal{C}$ with low recourse. However, our full algorithm will maintain this additional information.

Formally, the goal of this warm-up section is to prove the following theorem.

▶ **Theorem 10.** *For an edge-dynamic graph $G = (V, E)$ with $n = |V|$ vertices, a $\sqrt{\log n}$-depth $n^{o(1)}$-cluster size pivot hierarchy can be maintained in polynomial time per update with total recourse $\|\mathcal{C}\|_{\to} \leq n^{o(1)} \cdot \|G\|_{\to}$ on the pivot sets.*

We emphasize that this theorem is not particularly useful, as it comes without meaningful guarantees on the update time, and one could just use Dijkstras algorithm to obtain exact distances in this regime. Even the recourse guarantee is also of limited use, as we do not control the recourse of the pivot maps or the cluster maps of the pivot hierarchy. Nevertheless, our algorithm for proving this theorem is instructive and closely mirrors our final dynamic APSP algorithm.

The illustrative algorithm for maintaining pivot hierarchies repeatedly decreases the vertex count via low-recourse core graphs, and the edge count via a low-recourse spanner. We present these two pieces separately.

3.2 Low-Recourse KMG-vertex sparsifier

In this section, we state a theorem about maintaining a KMG-vertex sparsifier with low recourse [16]. Although their algorithm only works for edge-dynamic graphs, it can be extended to fully-dynamic graphs rather straightforwardly in the recourse setting. Recall that we count a vertex split as a single operation when defining the recourse of a fully-dynamic graph.

▶ **Definition 11** (Vertex Sparsifier)**.** *Given $G = (V, E, \boldsymbol{l})$ and $A \subseteq V$, we call a graph $\widetilde{G}$ with vertex set $V(\widetilde{G}) \supseteq A$ a β-approximate vertex sparsifier of G with respect to the set A if for every two vertices $u, v \in V(\widetilde{G})$ we have $\operatorname{dist}_G(u, v) \leq \operatorname{dist}_{\widetilde{G}}(u, v)$ and if $u, v \in A$ we further have $\operatorname{dist}_{\widetilde{G}}(u, v) \leq \beta \cdot \operatorname{dist}_G(u, v)$.*

▶ Remark 12. We always associate a map from $V(\widetilde{G})$ to V with a vertex sparsifier, where multiple vertices in $V(\widetilde{G})$ might map to the same vertex in V. This clarifies what we mean with $\operatorname{dist}_G(u, v) \leq \operatorname{dist}_{\widetilde{G}}(u, v)$ and $\operatorname{dist}_{\widetilde{G}}(u, v) \leq \beta \cdot \operatorname{dist}_G(u, v)$ in Definition 11. When we apply multiple vertex sparsifiers to a graph G, we refer to recursively applying this map as mapping vertices back to G.

▶ **Theorem 13** (See Theorem 3.1 in [16])**.** *Consider a size reduction parameter $k > 1$ and a fully-dynamic graph G with lengths in $[1, L]$. Then, there is a deterministic algorithm that explicitly maintains,*

1. *a monotonically growing pivot set $A \subseteq V(G)$ and pivot function $p : V \to A$, such that $|A| \leq n/k + 2\|G\|_{\to}$ and $|C(u)| \leq \widetilde{O}(k)$ for all $u \in V$ where $C(u) = \{v \in V : \operatorname{dist}_G(u, v) < \operatorname{dist}_G(u, p(u))\}$.*
2. *a fully-dynamic graph $\widetilde{G}$, such that $\widetilde{G}$ is a γ_{VS}-approximate vertex sparsifier of G with respect to A. We have that the initial graph $\widetilde{G}^{(0)}$ has at most $m \cdot \gamma_{VS}$ edges and $\gamma_{VS} \cdot m/k$ vertices, where $\gamma_{VS} = \widetilde{O}(1)$. The following properties hold:*
   a. *$\widetilde{G}$ has lengths $\boldsymbol{l}_{\widetilde{G}}$ in $[1, nL]$, and*
   b. *given any edge $e = (u, v) \in \widetilde{G}$, there exists a uv-path P in G with $l_G(P) \leq l_{\widetilde{G}}(e)$.*
   c. *$\left\|\widetilde{G}\right\|_{\to} \leq \gamma_{VS} \cdot (\|G\|_{\to} + m)$ and $\left\|D_{\widetilde{G}}\right\|_{\to} \leq \gamma_{VS} \cdot \|G\|_{\to}$ where $D_{\widetilde{G}}$ is the dynamic object containing the decremental operations (deletions and vertex splits) on $\widetilde{G}$.*

The algorithm runs in polynomial time.

3.3 Low-Recourse Dynamic Spanner

In this section, we describe a simple algorithm for maintaining a spanner H of a fully-dynamic graph G with remarkably low recourse. The construction is based on the greedy spanner of [1]. The low-recourse property of a variant of this algorithm has been observed in [3].

▶ **Theorem 14** (Dynamic Spanner). *There exists a data structure that given a fully-dynamic graph $G = (V, E, \boldsymbol{l})$ with polynomially bounded edge lengths, maintains a fully-dynamic $O(\log |V|)$-spanner H of G such that $|E(H)| \leq \gamma_{DS} \cdot |V(H)|$ and $\|H\|_{\rightarrow} \leq \gamma_{DS} \cdot (\|D_G\|_{\rightarrow} + |V|)$ for $\gamma_{DS} \in \widetilde{O}(1)$, where D_G is a dynamic object that contains all decremental updates to G (deletions and vertex splits).*

H can be maintained in polynomial time per update.

Proof. We assume unit lengths via maintaining separate spanners for length ranges $[2^i, 2^{i+1})$ and returning the direct product of the appropriately scaled spanners. Furthermore, we let $|V^{(0)}| = n$ and re-start after n decremental updates.

Then, there can be at most n vertex splits and therefore G contains at most $2n$ vertices before re-starting. We initialize $H^{(0)}$ to the greedy spanner of $G^{(0)}$, i.e. we consider the edges $(u, v) \in E(G^{(0)})$ in arbitrary order and add them to $E(H^{(0)})$ if $\text{dist}_{H^{(0)}}(u, v) > 2 \log 2n$. This ensure that the graph $H^{(0)}$ is a $O(\log |V|)$-spanner of $G^{(0)}$, and $|E^{(0)}| \leq O(n)$ directly follows from the fact that a girth $2 \log |V| + 1$ graph contains at most $O(|V|)$ edges.[1]

Whenever G is updated, we perform the corresponding operation on H in case of edge deletions and vertex splits such that $H \subseteq G$. Then, we iteratively check for every edge $(u, v) \in E(G) \setminus E(H)$ if $\text{dist}_H(u, v) > 2 \log 2n$ and insert the edges for which the condition is true. In particular, for the insertion of an edge (u, v), this corresponds to inserting the edge whenever $\text{dist}_H(u, v) > 2 \log 2n$.

Clearly H is an $O(\log n)$ spanner of G throughout because every edge is stretched by at most $2 \log 2n$. Furthermore, vertex splits and edge deletions only increase the girth of H, and therefore H contains at most $O(n)$ edges throughout. Finally, the total recourse bound $\|H\|_{\rightarrow} \leq O(n)$ follows until the data structure is re-started because an edge only leaves the graph H when it is deleted.

To conclude, it suffices to observe that the total number of restarts is bounded by $\|D_G\|_{\rightarrow} / n$ and there are $O(\log n)$ data structures in total, so $\gamma_{\text{DS}} = O(\log n)$. ◀

3.4 Low-Recourse Pivot Hierarchies via Iterated Sparsification

Given the low-recourse vertex and edge sparsification routines described above, we are ready to give the algorithm for Theorem 10.

Our algorithm consists of layers $0, \ldots, \Lambda - 1$ where $\Lambda = \sqrt{\log n}$.[2] Each layer consists of two graphs G_i and H_i, where $H_i \subseteq G_i$. We let $G_0 = G$. Then we obtain H_i from G_i and G_i from H_{i-1} as follows:

1. G_i is the output of the vertex sparsifier from Theorem 13 on the graph H_{i-1} for size reduction parameter $k \stackrel{\text{def}}{=} 2^{\sqrt{\log n}}$. For simplicity, we assume that k is an integer without loss of generality.
2. H_i is the dynamic spanner from Theorem 14 on the graph G_i.

[1] In [1] this folklore bound was already used to obtain a spanner.

[2] We remark that our full algorithm uses a much more drastic size reduction to enable bootstrapping, which results in only $O(\log^{\epsilon} \log n)$ layers for some small constant $\epsilon < 1$. This is necessary because the stretch is powered every time we recurse.

Whenever an update to G occurs, we pass the updates up the hierarchy until we reach the first layer j that has recieved more than $\gamma_{DS}^j \gamma_{VS}^j 2^{\Lambda-j}$ since it was initialized. Then, we re-initialize all the layers $j, \ldots, \Lambda$ via the steps described above. Notice in particular that the recourse of each layer is much smaller than the size reduction, leading to manageable recourse over all.

We let the sets A_i be the vertex sets of the graphs G_i mapped back to G, and we let A_Λ contain an arbitrary vertex r in $A_{\Lambda-1}$ (also mapped back to G). Furthermore, we let the pivot map $p_i(\cdot)$ be the pivot function maintained by the data structure from Theorem 13 at layer $i+1$ for $i = 0, \ldots, \Lambda-1$, and the cluster map C_i is the cluster map from vertex sparsifier data structure at layer $i+1$. Finally, we define the cluster map $C_\Lambda(v) \stackrel{\text{def}}{=} A_\Lambda$ and pivot map $p_\Lambda(v)$ for every vertex $v \in A_\Lambda$.

Proof of Theorem 10. We first bound the sizes and recourse of the sets $A_0, \ldots, A_\Lambda$ by induction. The recourse of the set A_0 and the recourse of G_0 is $\|G\|_\to$ by definition, and the recourse of H_0 is at most $\gamma_{\text{DS}}\big(\|G\|_\to + |V|\big)$.

We then observe that recourse caused by non-decremental updates on G_i can be ignored because they do not show up in the recourse of the subsequent spanner H_i by Theorem 14. The recourse of the decremental updates to G_i is $\gamma_{VS} \cdot \|H_{i-1}\|_\to$. The recourse of H_i is $\|H_i\|_\to \leq \gamma_{DS}(\|D_{G_i}\|_\to + |V(G_i)|)$. Therefore, t updates to G cause at most $O(t \cdot \gamma_{VS}^i \gamma_{DS}^i)$ vertices to be added to G_i. Since we re-start every layer i after $\gamma_{DS}^i \gamma_{VS}^i 2^{\Lambda-i}$ updates have arrived and the total amount of updates that reach layer i is $\gamma_{VS}^i \gamma_{DS}^i n$, the total recourse of the layer is $O(2^\Lambda \cdot n \cdot \gamma_{VS}^{i+1} \gamma_{DS}^{i+1})$ and the total recourse of all the sets is at most $O(k \cdot \gamma_{VS}^{\Lambda+1} \gamma_{DS}^{\Lambda+1} \cdot n \cdot \Lambda)$ by Theorem 13 and Theorem 14.

Then, we show that the final set $A_{\Lambda-1}$ is of small enough size for bounding the size of the clusters $C_{\Lambda-1}(\cdot)$. By the recourse bound above, there are at most $n^{o(1)} n / k^{\Lambda-i+1}$ updates to a level before it gets rebuilt. Therefore, the size of level $\Lambda-1$ is at most $n^{o(1)}$ at all times. The bound on the other cluster sizes directly follows from Theorem 13 and the description of our algorithm.

It remains to show that the distance estimates $\widetilde{\text{dist}}$ are good. To do so, we first notice that for every $u, v \in A_i$ we have $\text{dist}_G(u,v) \leq \text{dist}_{G_i}(u,v) \leq n^{o(1)} \text{dist}_G(u,v)$ since each layer only loses $\widetilde{O}(1)$ in distance approximation by Theorem 13 and Theorem 14 and there are $\Lambda = \sqrt{\log n}$ levels. Then, the distance approximation follows by induction since the length of the detour to the pivots can lose at most a poly-logarithmic factor per level because the pivot is closer in the graph G_i and $\text{dist}_G(u,v) \leq \text{dist}_{G_i}(u,v) \leq n^{o(1)} \cdot \text{dist}_G(u,v)$.[3] ◀

4 The Algorithm

We solve the APSP problem on a large graph G using a pivot hierarchy obtained by iterated vertex and edge sparsification, just like in the warm up in Section 3. Again, we will select a set of pivots for the first level of a pivot hierarchy, and perform edge and vertex sparsification to represent the distances between the pivots using a smaller dynamic graph, and then recurse on this graph. The main difference to the previous section is that (a) we want to bootstrap dynamic APSP along the way and (b) we want to develop and use a computationally efficient spanner. This spanner creates a few technical complications. In particular, to make the spanner efficient, we need all our graphs to have somewhat bounded maximum degree.

[3] The bounds on the detour are analogous to [17] after taking into account the extra distortion of the distances in G_i as compared to the original graph G. This extra distortion causes the sub-polynomial overhead.

4.1 Vertex sparsification

First, our algorithm reduces the APSP problem to a graph $\widetilde{G}$ with significantly fewer vertices using the KMG-vertex sparsifier (See Section 4.3). When compared to the low-recourse version presented in Section 3, our vertex sparsifier has an additional routine called ReduceDegree$(\cdot)$ which forces some of the vertices in the vertex sparsifier to be split and therefore have smaller degree. We further elaborate this point in the section below.

The vertex sparsification algorithm is presented in Section 4.3.

4.2 Edge sparsification

Although $\widetilde{G}$ is supported on a much smaller vertex set, it may still contain most of the edges from G. To achieve a proper size reduction, we additionally employ edge sparsification on top of the vertex sparsifier $\widetilde{G}$.

There are two key differences between the warm up and the efficient spanner algorithm.

- To quickly check for detours in the spanner, the efficient algorithm recursively uses APSP datastructures on small instances (See Section 4.4).
- To efficiently maintain the dynamic spanners, it is crucial that the degree of the graph remains bounded. Although we can assume that the degree of the initial graph G is bounded by a constant, repeatedly sparsifying the graph could seriously increase its maximum degree. Fortunately, our dynamic spanner guarantees that the average degree after edge sparsification is low. However, the same cannot be achieved for the maximum degree, which becomes apparent when trying to sparsify a star graph.
  To remedy this issue, we use that the underlying graph has bounded degree and call the ReduceDegree$(\cdot)$ routine of the KMG-vertex sparsifier to reduce the degree of high-degree vertices in the spanner. This operation can lead to changes in $\widetilde{G}$ and its spanner, but we show that repeatedly splitting vertices with high degree in the spanner quickly converges because the progress of splits is larger than the recourse they cause to the spanner.

The edge sparsification algorithm is presented in Section 4.4.

4.3 Pivots and Vertex Sparsification

We use the KMG-vertex sparsifier algorithm [16], which is based on dynamic core graphs, which in turn are based on static low-stretch spanning trees.

We first introduce certified pivots. These are mapping vertices that get sparsified away to vertices in $\widetilde{G}$, and certify the distance for vertices that are closer to each other than to their respective pivots.

▶ **Definition 15** (Certified Pivot). *Consider a graph $G = (V, E)$ with edge lengths $\boldsymbol{l}$ and a set $A \subseteq V$. We say that a function $p : V \to A$ is a pivot function for A, if for all $v \in V$, we have that $p(v)$ is the nearest vertex in A (ties broken arbitrarily but consistently).*

Additionally, we say that p is a certified pivot function if for all vertices $v \in V$, for all $u \in B_G(v, p(v)) \cup \{p(v)\}$ we have computed the exact (u, v) shortest paths and distances.

To motivate Definition 15, we show that pivots can be used to approximate the distance between sparsified vertices whenever their exact distance isn't stored already.

▷ Claim 16. Consider a graph $G = (V, E, l)$ with pivot set A and corresponding pivot function p. Then for any $u, v \in V$, if $v \notin B_G(u, p(u))$, we have $\mathrm{dist}_G(p(u), p(v)) \leq 4 \cdot \mathrm{dist}_G(u, v)$.

Proof. By $v \notin B_G(u, p(u))$ we directly obtain $\text{dist}_G(u, p(u)) \leq \text{dist}_G(u, v)$ and since $p(v)$ is (one of) the neareast vertices to v in A, we have $\text{dist}_G(v, p(v)) \leq \text{dist}_G(v, p(u)) \leq \text{dist}_G(v, u) + \text{dist}_G(u, p(u)) \leq 2\,\text{dist}_G(u, v)$ as well. By using the triangle inequality we get $\text{dist}_G(p(u), p(v)) \leq \text{dist}_G(p(u), u) + \text{dist}_G(u, v) + \text{dist}_G(v, p(v)) \leq 4\,\text{dist}_G(u, v)$. ◁

Next, we state a version of the KMG-vertex sparsifier [16], which is based on dynamic core graphs. We refer the reader to the full version of our article for a more detailed discussion of their vertex sparsifier.

The following theorem should be thought of as an efficient analog to Theorem 13 with the extra operation ReduceDegree(·). As mentioned at the start of this section, this routine comes into play when the spanner produces a high-degree vertex down the line.

▶ **Theorem 17** (KMG-vertex sparsifier, See Theorem 3.1 in [16])**.** *Consider a size reduction parameter $k > 1$ and an edge-dynamic graph G, with $\deg_{\max}(G) \leq \Delta$ and lengths in $[1, L]$. Then, there is a deterministic algorithm that explicitly maintains,*

1. *a monotonically growing pivot set $A \subseteq V(G)$ and certified pivot function $p : A \to V$, such that $|A| \leq n/k + 2\,\|G\|_{\to}$.*
2. *a fully-dynamic graph $\widetilde{G}$, such that $\widetilde{G}$ is a γ_{VS}-approximate vertex sparsifier of G with respect to A. We have that $\widetilde{G}^{(0)}$ has at most $m \cdot \gamma_{VS}$ edges and $\gamma_{VS} \cdot m/k$ vertices, where $\gamma_{VS} = \widetilde{O}(1)$. The following properties hold:*
   - **a.** *$\deg_{\max}(\widetilde{G}) \leq \Delta \cdot \gamma_{VS} \cdot k$, and $\widetilde{G}$ has lengths $\boldsymbol{l}_{\widetilde{G}}$ in $[1, nL]$, and*
   - **b.** *given any edge $e = (u, v) \in \widetilde{G}$, the algorithm can return a uv-path P in G with $l_G(P) \leq l_{\widetilde{G}}(e)$ in time $O(|P|)$.*
   - **c.** *a procedure ReduceDegree(E', z), where $E' \subseteq \{e \in E(\widetilde{G}) : v \in e\}$ for some $v \in V(\widetilde{G})$, and z is a positive integer. It updates the graph $\widetilde{G}$, performing at most $\gamma_{VS} \cdot |E'|/z$ vertex splits, such that afterwards any vertex $v' \in V(\widetilde{G})$ with $v \in \text{master}(v')$ is adjacent to at most $z \cdot \Delta$ edges in E'. This procedure runs in time at most $\gamma_{VS} \cdot k \cdot |E'|$.*
   - **d.** *If R_V is the dynamic object containing the total number of vertex splits performed by ReduceDegree, then $\left\|\widetilde{G}\right\|_{\to} \leq \gamma_{VS} \cdot \|G\|_{\to} + \|R_V\|_{\to}$.*

The algorithm runs in time $m \cdot \gamma_{VS} \Delta k^4 + \|G\|_{\to} \cdot \gamma_{VS} k^4 \Delta$.

4.4 Edge Sparsification

We state our main dynamic spanner theorem whose proof is deferred to the full version of our article. The theorem assumes access to a $\gamma_{apxAPSP}$-approximate, $\widetilde{O}(1)$-query APSP data structure (See Definition 7.)

▶ **Theorem 18** (Dynamic Spanner)**.** *There exists a data structure that given a fully-dynamic graph G with unit lengths and $|V(G^{(0)})| = n$, $\|G\|_{\to} \leq n$ and a parameter $1 \leq K \leq O(\log^{1/3} n)$, maintains a fully-dynamic $\gamma_l^{O(K)}$-spanner S of G such that $|E(S)| \leq \gamma_{ES} \cdot n$ and $\|S\|_{\to} \leq \gamma_{ES} \cdot \|G\|_{\to}$, where $\gamma_l = \widetilde{O}(\gamma_{apxAPSP})$ and $\gamma_{ES} = \widetilde{O}(n^{1/K})$. It runs in time $n \deg_{\max}(G^{(0)}) \gamma_{ES} \gamma_l^{O(K^2)} + APSP(\gamma_{ES} n, 3, \gamma_{ES} n)$.*

▶ Remark 19. We can directly extend Theorem 18 to graphs with edge lengths in $[1, L]$ by bucketing edges in the intervals $[2^i, 2^{i+1})$ for $i = 1, \ldots, O(\log L)$.

Note also that as at all times H is a subgraph of G, it does not undergo more vertex splits and isolated vertex insertions than G, i.e., all extra updates to maintain H are edge insertions/deletions.

4.5 Bootstrapping via Edge and Vertex Sparsification

We are given an edge-dynamic graph G on initially n vertices and m edges, that at all times has maximum degree at most Δ. Applying Theorem 17 to G reduces the problem to finding short paths in $\widetilde{G}$, a graph of $\widetilde{O}(m/k)$ vertices. Although the number of vertices significantly decreases, the graph still contains up to $\widetilde{O}(m)$ edges, preventing efficient recursion. To address this, we apply Theorem 18 on $\widetilde{G}$ to produce a graph H on $\gamma \cdot m/k$ vertices *and* edges, where γ crucially depends only on the parameter K of Theorem 18.

Note that the APSP data structure that Theorem 18 requires also only needs to run on a graph of at most $\gamma \cdot m/k$ vertices and edges, so by choosing a large enough size reduction factor k, the problem size reduces by a factor of γ/k, while the approximation factor increases from $\gamma_{apxAPSP<m}$ to $\widetilde{O}(\gamma_{apxAPSP<m})^{O(K)}$.[4] Balancing both parameters k and K allows us to bootstrap an APSP data structure with the guarantees from Theorem 1.

While H is sparse, it might still have high maximum degree, preventing efficient recursion. The sparsity still implies, however, that there cannot be too many high degree vertices. Hence, we carefully perform a few vertex splits in $\widetilde{G}$ by leveraging the ReduceDegree$(\cdot)$ functionality from Theorem 17 to enforce that H also has small maximum degree.

Algorithm 1 initDynamicAPSP().

1 Maintain $\widetilde{G}$ from Theorem 17.
2 Let H be the graph maintained by applying Theorem 18 to graph $\widetilde{G}$ with parameter K.
`/* Here` $\gamma_{degConstr}$ `is a constant fixed later. */`
3 **while** $\exists v \in V(H)$ *with* $\deg_H(v) > 8\gamma_{degConstr} \cdot \Delta$ **do**
4 | ReduceDegree$(E_H(v), \gamma_{degConstr})$.
5 Initialize and maintain a recursive dynamic APSP data structure on H.

The recursive APSP instance that is running on H expects an edge-dynamic graph, while the spanner from Theorem 18 is fully-dynamic. To circumvent this issue, we simulate the vertex splits by suitable edge insertions/removals and vertex insertions. As the maximum degree of H gets controlled during the while loops, this simulation can lead to at most an extra factor of $O(\gamma_{degConstr} \cdot \Delta)$ in the number of updates.

Algorithm 2 MaintainDynamicAPSP(G, t).

1 Update $\widetilde{G}$ and H accordingly.
2 **while** $\exists v \in V(H)$ *with* $\deg_H(v) > 8\gamma_{degConstr} \cdot \Delta$ **do**
3 | ReduceDegree$(E_H(v), \gamma_{degConstr})$.
4 | Update H accordingly.
5 Forward all updates made to H to the recursive dynamic APSP datastructure that runs on H by simulating the vertex splits accordingly

When queried for the distance between two vertices u and v, our algorithm checks if the certified pivot function stores the distance, and if this is not the case it recursively asks the smaller instance DynamicAPSP$_H$ for the distance between $p(u)$ and $p(v)$ in H.

[4] We only have a valid size reduction if also $\gamma \cdot m/k < n$. We will see that this is indeed the case as we can assume that initially $m = O(n)$.

Algorithm 3 $\textsc{Dist}(u, v)$.

```
if v ∈ B_G(u, A) then
    return dist(u, v)
else
    return dist(u, p(u)) + DynamicAPSP_H.Dist(p(u), p(v)) + dist(p(v), v)
```

4.6 Analysis

We first show that the while loops in Algorithm 1 and Algorithm 2 terminate sufficiently fast. This is the most crucial claim for the analysis.

▷ Claim 20. If R_V is the dynamic object counting the total number of vertex splits performed by ReduceDegree() in Algorithm 1 and Algorithm 2, then $\|R_V\|_{\to} \leq 2\gamma_{VS}(m/k + \|G\|_{\to})$. In particular, the while loops always terminate and immediately thereafter H has maximum degree at most $8\gamma_{degConstr} \cdot \Delta$. The total runtime of all calls to ReduceDegree() is $6\gamma_{VS}^2\gamma_{ES}(m + k\|G\|_{\to})$.

Proof. For tracking progress achieved by vertex splits, we introduce the following potential function

$$\Phi(H) \stackrel{\text{def}}{=} \sum_{v \in V(H)} \max\{\deg_H(v) - \Delta \cdot \gamma_{degConstr}, 0\}.$$

Let t be the time before we enter the while loop in Algorithm 1. Then by Theorem 17 we have that $|V(\widetilde{G}^{(t)})| \leq \gamma_{VS}m/k$ and thus by Theorem 18, we have that $|E(H^{(t)})| \leq \gamma_{VS}\gamma_{ES}m/k$.

By the Handshake-Lemma, $\Phi(H^{(t)}) \leq 2|E(H^{(t)})| \leq 2 \cdot \gamma_{ES} \cdot \gamma_{VS} \cdot m/k$. Note that whenever an update is made to H, the potential can increase by at most 2. This is as edge deletions and vertex splits can only decrease it, and an edge insertion can increase it by at most 2.

Let R_E be the dynamic object counting the number of edges that were passed to ReduceDegree, and note that if $\textsc{ReduceDegree}(E', \gamma_{degConstr})$ is called, we have $|E'| > 8\gamma_{degConstr} \cdot \Delta$ by the while loop condition.

By the properties of ReduceDegree() in Theorem 17, we have that $\|R_V\|_{\to} \leq \gamma_{VS}/\gamma_{degConstr} \cdot \|R_E\|_{\to}$, and that none of the vertices v' that result from the splits to v will be adjacent to more than $\gamma_{degConstr} \cdot \Delta$ of the edges in E'. So if we just forwarded these splits to H, we would have $\deg_H(v') \leq \gamma_{degConstr} \cdot \Delta$, and the potential decreases by at least $7/8 \cdot |E'|$.

However, forwarding these vertex splits to H, Theorem 18 will cause further updates to H to maintain the spanner. Fortunately by Item 2d of Theorem 17 we know that $\|H\|_{\to} \leq \gamma_{ES} \cdot \left\|\widetilde{G}\right\|_{\to} \leq \gamma_{ES} \cdot \gamma_{VS} \cdot \|G\|_{\to} + \gamma_{ES} \cdot \|R_V\|_{\to}$. Let $\gamma = \gamma_{ES} \cdot \gamma_{VS}$. Putting everything together we show that the potential decreases as follows.

$$\begin{aligned}\Phi(H) &\leq 2\gamma m/k - \frac{7}{8}\|R_E\|_{\to} + 2\|H\|_{\to} \leq 2\gamma m/k - \frac{7}{8}\|R_E\|_{\to} + 2\gamma\|G\|_{\to} + 2\gamma_{ES}\|R_V\|_{\to} \\ &\leq 2\gamma m/k - \frac{7}{8}\|R_E\|_{\to} + 2\gamma\|G\|_{\to} + \frac{2\gamma_{ES}\gamma_{VS}}{\gamma_{degConstr}}\|R_E\|_{\to}.\end{aligned}$$

Thus, choosing $\gamma_{degConstr} = \lceil 4\gamma_{VS}\gamma_{ES} \rceil$, we get $\Phi(H) \leq 2\gamma m/k - \frac{3}{8}\|R_E\|_{\to} + 2\gamma\|G\|_{\to}$. As $0 \leq \Phi(H)$, this directly implies that $\|R_E\|_{\to} \leq 6\gamma_{ES}\gamma_{VS}(m/k + \|G\|_{\to})$. Remembering that $\|R_V\|_{\to} \leq \frac{\gamma_{VS}}{\gamma_{degConstr}}\|R_E\|_{\to}$ then yields the bound on $\|R_V\|_{\to}$.

Since the total runtime is given by $\gamma_{VS} \cdot k \cdot \|R_E\|_{\to}$, this bound follows directly. ◁

We then show a bound on the total runtime of our dynamic APSP algorithm in terms of costs for smaller instances, setting us up for our final recursion.

To do so, assume the recursive APSP instance that is running on H and used in Theorem 18 is a $\gamma_{apxAPSP<m}$-approximate, $\widetilde{O}(1)$-query APSP. Let $\gamma_l = \widetilde{O}(\gamma_{apxAPSP<m})$ be the stretch parameter from Theorem 18.

▶ **Theorem 21.** *The total combined runtime of Algorithm 1 and Algorithm 2 over a sequence of up to m/k updates is $m \cdot \gamma_{RC} \cdot \gamma_l^{O(K^2)} \Delta k^4 + APSP(\gamma_{RC} m/k, \gamma_{RC}\Delta, \gamma_{RC}\Delta m/k)$ for some $\gamma_{RC} = \widetilde{O}(m^{3/K})$.*

Proof. We first show that for at most m/k updates, initializing and maintaining H is efficient. Note that by the previous claim, we can bound the total costs incurred in the while loops by $6\gamma_{VS}^2\gamma_{ES}(m + k\|G\|_{\to}) \leq 12\gamma_{VS}^2\gamma_{ES}m$. The total costs of initializing and maintaining $\widetilde{G}$ are $2m \cdot \gamma_{VS}\Delta k^4$ by Theorem 17. Finally, by the previous lemma $\|\widetilde{G}\| \leq \gamma_{VS}\|G\|_{\to} + \|R_V\|_{\to} \leq 3\gamma_{VS}\|G\|_{\to} + 2\gamma_{VS}m/k \leq 4\gamma_{VS}m/k$, so that we can further bound the runtime of Theorem 18 by $\gamma_{VS}^2\gamma_{ES}n\Delta k\gamma_l^{O(K^2)} + APSP(\gamma_{ES} \cdot n, 3, \gamma_{ES} \cdot n)$. Note that here we also crucially used that by Theorem 17, $\Delta_{\max}(\widetilde{G}) \leq k\gamma_{VS}\Delta$.

It remains to consider the APSP data structure that we have to run on H. First note that by the gurantees of Theorem 17 and Theorem 18, initially $|E(H)| \leq \gamma_{VS}\gamma_{ES}m/k$. Secondly, note that as $\|\widetilde{G}\| \leq 4\gamma_{VS}m/k$, we in particular have $\|H\|_{\to} \leq 4\gamma_{ES}\gamma_{VS}m/k$. However, as H is vertex dynamic, we also have to simulate all the vertex splits occuring in H, which can lead to up to $8\gamma_{degConstr}\Delta$ extra updates per split. Hence the total amount of updates that the APSP data structure receives are up to $O(\gamma_{VS}^2\gamma_{ES}^2 m/k)$. By the while loop conditions, we do know that the maximum degree of the graph is always bounded by $8\gamma_{degConstr}\Delta$. Therefore, the total runtime bound follows. ◀

We show that the stretch does not increase too much as the size of the graph increases.

▷ Claim 22. Algorithm 1 provides a worst case stretch of $\widetilde{O}(\gamma_{apxAPSP<m})^{O(K)}$.

Proof. If $v \in B_G(u, A)$ there is nothing to show as then by Theorem 17 we have already pre-computed the exact distance between u and v.

But if $v \notin B_G(u, A)$, then by Claim 16, we have that $\text{dist}_G(p(u), p(v)) \leq 4 \cdot \text{dist}_G(u, v)$. As $p(u), p(v) \in A$, we have $\text{dist}_H(p(u), p(v)) \leq \gamma_l^{O(K)} \cdot \text{dist}_{\widetilde{G}}(p(u), p(v)) \leq \gamma_l^{O(K)} \cdot \gamma_{VS} \cdot \text{dist}_G(p(u), p(v)) = \widetilde{O}(1) \cdot (\widetilde{O}(1) \cdot \gamma_{apxAPSP})^{O(K)} \cdot \text{dist}_G(p(u), p(v)) = \widetilde{O}(\gamma_{apxAPSP})^{O(K)} \cdot \text{dist}_G(p(u), p(v))$. ◁

▶ Remark 23. We can directly extend this to receive a procedure $\textsc{Path}(u, v)$ that provides a u-v path P with $|P| \leq \widetilde{O}(\gamma_{apxAPSP<m})^{O(K)} \cdot \text{dist}_G(u, v)$ and runs in time $\widetilde{O}(|P|)$. This is because Theorem 17 also explicitly stores exact shortest paths between vertices u and their pivots $p(u)$.

In order to prove Theorem 1, we need to carefully assemble the previous insights, which we defer to the full version of our article.

References

1 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993. doi:10.1007/BF02189308.

2 A. Bernstein, M. Gutenberg, and T. Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1000–1008, Los Alamitos, CA, USA, February 2022. IEEE Computer Society. doi:10.1109/FOCS52979.2021.00100.

3 Sayan Bhattacharya, Thatchaphol Saranurak, and Pattara Sukprasert. Simple Dynamic Spanners with Near-Optimal Recourse Against an Adaptive Adversary. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.17.

4 J. Brand, L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. Gutenberg, S. Sachdeva, and A. Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514, Los Alamitos, CA, USA, November 2023. IEEE Computer Society. doi:10.1109/FOCS57990.2023.00037.

5 Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 170–181, 2018. doi:10.1109/FOCS.2018.00025.

6 Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, s-t shortest path, and minimum-cost flow. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, pages 1165–1173, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3618260.3649745.

7 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022. doi:10.1109/FOCS54457.2022.00064.

8 Julia Chuzhoy. Decremental all-pairs shortest paths in deterministic near-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 626–639, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451025.

9 Julia Chuzhoy and Thatchaphol Saranurak. Deterministic Algorithms for Decremental Shortest Paths via Layered Core Decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 2478–2496. Society for Industrial and Applied Mathematics, January 2021. doi:10.1137/1.9781611976465.147.

10 Julia Chuzhoy and Ruimin Zhang. A new deterministic algorithm for fully dynamic all-pairs shortest paths. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1159–1172, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585196.

11 Sebastian Forster, Gramoz Goranci, and Monika Henzinger. Dynamic maintenance of low-stretch probabilistic tree embeddings with applications. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1226–1245, 2021. doi:10.1137/1.9781611976465.75.

12 Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Bootstrapping Dynamic Distance Oracles. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.50.

13 Sebastian Forster, Yasamin Nazari, and Maximilian Probst Gutenberg. Deterministic incremental apsp with polylogarithmic update time and stretch. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1173–1186, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585213.

14 Bernhard Haeupler, Yaowei Long, and Thatchaphol Saranurak. Dynamic deterministic constant-approximate distance oracles with n^{ϵ} worst-case update time, 2024. doi:10.48550/arXiv.2402.18541.

15 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. *J. ACM*, 65(6), November 2018. doi:10.1145/3218657.

16 Rasmus Kyng, Simon Meierhans, and Maximilian Probst Gutenberg. A dynamic shortest paths toolbox: Low-congestion vertex sparsifiers and their applications. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, pages 1174–1183, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3618260.3649767.

17 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, January 2005. doi:10.1145/1044731.1044732.

A Deterministic Partition Tree and Applications

Haitao Wang
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

In this paper, we present a deterministic variant of Chan's randomized partition tree [Discret. Comput. Geom., 2012]. This result leads to numerous applications. In particular, for d-dimensional simplex range counting (for any constant $d \geq 2$), we construct a data structure using $O(n)$ space and $O(n^{1+\epsilon})$ preprocessing time, such that each query can be answered in $o(n^{1-1/d})$ time (specifically, $O(n^{1-1/d}/\log^{\Omega(1)} n)$ time), thereby breaking an $\Omega(n^{1-1/d})$ lower bound known for the semigroup setting. Notably, our approach does not rely on any bit-packing techniques. We also obtain deterministic improvements for several other classical problems, including simplex range stabbing counting and reporting, segment intersection detection, counting and reporting, ray-shooting among segments, and more. Similar to Chan's original randomized partition tree, we expect that additional applications will emerge in the future, especially in situations where deterministic results are preferred.

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

Keywords and phrases partition trees, simplex range searching, segment intersection queries, ray-shootings, multi-level data structures

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.114

Related Version *Full Version*: https://arxiv.org/abs/2507.01775

Acknowledgements We would like to thank Timothy Chan for the discussions on derandomizing his partition tree.

1 Introduction

Simplex range searching is a fundamental problem in computational geometry. Given a set P of n points in the d-dimensional space $\mathbb{R}^d$ for a constant $d \geq 2$, the goal is to build a data structure so that points of P inside a query simplex can be found efficiently. The problem has been extensively studied (see [1, 2, 19] for some excellent surveys). For solving the problem with small space (e.g., near linear), one powerful technique is *partition trees*, e.g., [7, 14, 16–18, 23–25]. In particular, using a partition tree Matoušek [17] built a data structure of $O(n)$ space in $O(n \log n)$ time and each simplex range query can be answered in $O(n^{1-1/d} \log^{O(1)} n)$ time. Subsequently Matoušek [18] gave another more complicated data structure of $O(n)$ space with $O(n^{1+\epsilon})$ preprocessing time and $O(n^{1-1/d})$ query time; throughout the paper let ϵ be an arbitrarily small positive constant. Chazelle [10] proved that $\Omega(n^{1-1/d}/\log n)$ (and $\Omega(\sqrt{n})$ for $d = 2$) is a lower bound on the query time for an $O(n)$-space data structure; it is widely believed that the $\log n$ factor is an artifact of the proof. Although the result of [18] seems to achieve optimal query time with linear space, it is not quite satisfactory. One reason is that partition trees are often used as backbone for designing multi-level data structures and certain properties of the result of [18] makes this challenging, e.g., it has a special root of $O(n^{1/d} \log n)$ degree whose children may have overlapping cells and it does not guarantee the optimal crossing number except at the bottom most level of the tree. To address these issues, Chan [7] proposed a randomized partition tree of $O(n)$ space that can be built in $O(n \log n)$ expected time and the query time is bounded by $O(n^{1-1/d})$ w.h.p. Comparing to Matoušek's work [18], Chan's result has many nice properties, e.g.,


© Haitao Wang;
licensed under Creative Commons License CC-BY 4.0

each node has $O(1)$ children, crossing number is optimal (with w.h.p) at almost all layers (except the top few layers), and children's cells at each node are pairwise disjoint. These properties make Chan's partition tree quite amenable to multi-level data structures [7–9].

Simplex range searching has several versions: (1) Counting: compute the number of points of P inside the query simplex Δ; (2) reporting: report all points of P inside Δ; (3) semigroup query (which generalizes the counting query): compute the sum of the weights of the points of P inside Δ, assuming that each point of P is assigned a weight from a semigroup. The above results [7, 17, 18] are applicable to semigroup queries and can also be modified to solve range reporting with an additive term k in the query time, where k is the output size. In particular, Chazelle's lower bound [10] is for the semigroup setting.

Since Chan's partition tree is randomized, for those who need deterministic algorithms, Matoušek's partition trees [17, 18] are still the main resort. In this paper, we "partially" derandomize Chan's partition tree and obtain a deterministic tool, especially for designing multi-level data structures. More specifically, our partition tree is similar to Chan's (e.g., $O(n)$ space, optimal crossing number at all levels except the top few levels, disjointness of children's cells of each node); however, the degree of each node is logarithmic instead of constant (that is why we used "partially" above). Albeit this drawback, our tree is still powerful enough to have many applications, as discussed below.

Simplex range counting. For simplex range counting, we construct a data structure of $O(n)$ space that can answer each query in $O(n^{1-1/d}/\log^{\Omega(1)} n)$ time. Note that this does not violate Chazelle's lower bound [10] as it is for the more general semigroup queries. The preprocessing time is $O(n^{1+\epsilon})$. To achieve the result, we construct our partition tree so that each leaf has $O(\log^{\tau} n)$ points of P for an arbitrarily small constant $\tau > 0$. Because this number is small, we can afford to preprocess all leaves in $O(n)$ time by considering all possible configurations for queries; in this way, each query on a single leaf v can be answered in $O(\log\log n)$ time. While this kind of technique may not be quite surprising, it has never been used in simplex range searching, perhaps because previous work has been focusing on the semigroup queries while this technique does not work in that setting. Note that our above result is also applicable to simplex range emptiness queries.

It should be noted that very recently Chan and Zheng [9] obtained similar randomized results (i.e., $O(n)$ words of space and $o(n^{1-1/d})$ query time w.h.p.) for other data structures and also mentioned a possibility of achieving such result for simplex range counting. One difference is that their technique uses bit-packing by assuming each word has $\Omega(\log n)$ bits (note that the $\log n$-bit word RAM is also a conventional computational model), while ours does not use bit-packing. In addition, their result is randomized while ours is deterministic.

Simplex range stabbing. Given a set of n simplices in $\mathbb{R}^d$, the *simplex range stabbing counting problem* is to build a data structure to compute the number of simplices containing a query point. Using Chan's partition tree, Chan and Zheng [9] built a randomized data structure of $O(n\log\log n)$ space (or $O(n)$ words of space using the bit-packing tricks) in $O(n\log n)$ expected preprocessing time and the query time is $O(n^{1-1/d})$ w.h.p. Using our new deterministic partition tree, we build a deterministic data structure of $O(n\log\log n)$ space in $O(n^{1+\epsilon})$ preprocessing time and the query time is $O(n^{1-1/d}/\log^{\Omega(1)} n)$. Previously, the best deterministic results [17, 18] have $O(n^{1+\epsilon})$ preprocessing time, $O(n\log^{O(1)} n)$ space, and $O(n^{1-1/d}\log^{O(1)} n)$ query time; or $O(n2^{\sqrt{\log n}})$ preprocessing time and space, and $n^{1-1/d}\cdot 2^{O(\sqrt{\log n})}$ query time. For the *reporting problem* (i.e., report all simplices containing a query point), the randomized data structure of [9] has the same performance as above except that the query time becomes $O(k + n^{1-1/d})$ w.h.p., where k is the output size. We also obtain the same deterministic result as above with $O(k + n^{1-1/d}/\log^{\Omega(1)} n)$ query time.

Segment intersection searching. Given a set of n (possibly intersecting) line segments in the plane, the *segment intersection counting problem* is to build a data structure to compute the number of segments intersecting a query segment. Chan and Zhen [9] built a randomized data structure of $O(n \log\log n)$ space (which again can be reduced to $O(n)$ words of space if bit-packing tricks are allowed) in $O(n \log n)$ expected preprocessing time and the query time is $O(\sqrt{n})$ w.h.p. Using our new deterministic partition tree, we build a deterministic data structure of $O(n \log\log n)$ space in $O(n^{1+\epsilon})$ preprocessing time and the query time is $O(\sqrt{n}/\log^{\Omega(1)} n)$. The previously best deterministic result [6] built a data structure of $O(n \log^2 n)$ space in $O(n^{3/2})$ time that can answer each query in $O(\sqrt{n} \log n)$ time.

For the *reporting problem* (i.e., report all segments intersecting a query segment), the randomized data structure of [9] has the same performance as above except that the query time becomes $O(k + \sqrt{n})$ w.h.p., where k is the output size. We also obtain the same deterministic result as above with $O(k + \sqrt{n}/\log^{\Omega(1)} n)$ query time.

Segment intersection detection. Given n (possibly intersecting) line segments in the plane, we wish to build a data structure to decide whether a query line intersects any segment [12,22]. The previous deterministic result [22] builds an $O(n)$-space data structure in $O(n^{3/2})$ time, with $O(\sqrt{n} \log n)$ query time. Using our new partition tree, we build an $O(n)$-space data structure with $O(\sqrt{n}/\log^{\Omega(1)} n)$ query time and $O(n^{1+\epsilon})$ preprocessing time.

Ray-shooting among non-intersecting segments. Given n non-intersecting line segments in the plane, we wish to build a data structure to find the first segment hit by a query ray [3,6,20]. The previous best deterministic result [22] builds an $O(n)$-space data structure in $O(n^{3/2})$ time, with $O(\sqrt{n} \log n)$ query time. With our partition tree, we build an $O(n)$-space data structure with $O(\sqrt{n}/\log^{\Omega(1)} n)$ query time and $O(n^{1+\epsilon})$ preprocessing time.

If the segments are allowed to intersect, then the problem has also been studied [3, 4, 6, 7, 9, 12, 15, 20, 22]. The previously best deterministic result [22] builds an $O(n \log n)$-space data structure in $O(n^{3/2})$ time and each query can be answered in $O(\sqrt{n} \log n)$ time.

Outline. After introducing notation in Section 2, we present our partition tree in Section 3. The simplex range counting problem is treated in Section 4. The simplex range stabbing and the segment intersection searching problems are discussed in Section 5. We solve the segment intersection detection and the ray-shooting problems in Sections 6 and 7, respectively. Due to the space limit, many proofs and details are omitted but can be found in the full paper.

2 Preliminaries

Let H be a set of n hyperplanes in $\mathbb{R}^d$. We use $\mathcal{A}(H)$ to denote the arrangement of H, which can be computed in $O(n^d)$ time [13]. For a compact region $R \in \mathbb{R}^d$, we use H_R to denote the subset of hyperplanes of H that intersect the relative interior of R but does not contain R (we also say that these hyperplanes *cross* R). A *cutting* for H is a collection Ξ of closed cells (each of which is a simplex, possibly unbounded) with disjoint interiors, which together cover the entire space $\mathbb{R}^d$ [11,18]. The *size* of Ξ is the number of cells of Ξ. For a parameter r with $1 \le r \le n$, a $(1/r)$-*cutting* for H is a cutting Ξ satisfying $|H_\sigma| \le n/r$ for every cell $\sigma \in \Xi$.

For any $1 \le r \le n$, a $(1/r)$-cutting of size $O(r^d)$ for H can be computed in $O(nr^{d-1})$ time [11]. We further have Lemma 1 (similar results were mentioned before [5, 7, 21]). Throughout the paper, β always refers to the one in the lemma.

▶ **Lemma 1.** (Cutting Lemma) *Let H be a set of n hyperplanes and Δ a simplex in $\mathbb{R}^d$. For any $1 \le r \le n$, we can compute a $(1/r)$-cutting of $O(K \cdot (r/n)^d + r^{d-1+\beta})$ cells for H whose union is Δ in $O(K \cdot (r/n)^{d-1} + nr^{d-2+\beta})$ time, where K is the number of vertices of $\mathcal{A}(H)$ inside Δ and β is an arbitrarily small positive constant.*

Proof. We apply Chazelle's algorithm [11] but only on the region inside Δ (i.e., starting with $C_0 = \Delta$ following the notation of [11]). A detailed analysis for the 2D case is given in [21, Appendix A]. Below we sketch how to modify the analysis in [11] accordingly by following the notation there.

The analysis follows the same approach except that we use K to replace $\binom{n}{d}$ in the formula $\sum_{s \in C_{k-1}} v(H_{|s}; s) \le \binom{n}{d}$ in [11, Page 153]. As such, the subsequent formula becomes

$$|C_k| \le c \left(\frac{r_0^k \log r_0}{n} \right)^d \cdot K + c r_0^{d-1} (\log r_0)^d |C_{k-1}|.$$

Then, one can prove by induction that $|C_k| \le r_0^{d(k+1)} \cdot K/n^d + r_0^{(k+1)\cdot(d-1+\beta)}$, for an arbitrarily small constant $\beta > 0$. Therefore, the total number of cells of the cutting inside Δ is as stated in the lemma.

For the time analysis, following the same formula $\sum_{0 \le k \le \lceil \log_{r_0} r \rceil} \frac{n}{r_0^k} |C_k|$ in [11, Page 153] and using the above inequality for $|C_k|$, we can derive the time complexity as stated in the lemma. ◀

3 Deterministic partition tree

In this section, we present our deterministic partition tree. The following lemma "partially" derandomizes Chan's partition refinement theorem (i.e., Theorem 3.1 [7]).

▶ **Lemma 2.** *Let P be a set of n points and H a set of m hyperplanes in $\mathbb{R}^d$. Suppose there are t interior-disjoint cells whose union covers P, such that each cell contains at most $2n/t$ points of P and each hyperplane crosses at most κ cells. Then, for any $b \ge 4$, we can divide every cell into $O(b)$ disjoint subcells each containing at most $2n/(bt)$ points of P, for a total of at most bt subcells, so that the total number of subcells crossed by any hyperplane in H is bounded by*

$$O((b \cdot t)^{1-1/d} + b^{1-1/(d-1+\beta)} \cdot \kappa \cdot \log t + b \cdot \log m). \tag{1}$$

Remark. Comparing to Chan's partition refinement theorem, there is an extra $\log t$ in the second term of (1). As will be seen next, due to this extra factor, we have to make each node of our partition tree have logarithmically many children instead of constant. That is why we said before that we "partially" derandomized Chan's result.

3.1 Proving Lemma 2

This subsection is devoted to the proof of Lemma 2.

Let S denote the set of all t given cells in Lemma 2. Let H' be a multiset containing C copies of each hyperplane in H, for a parameter C that is a sufficiently large power of b (so that all future multiplicities are integers; the actual value of C is not important). For any multiset H'' of H, the *size* of H'', denoted by $|H''|$, is the sum of the multiplicities of the hyperplanes in H''. For a cell Δ, let $N_\Delta(H'')$ denote the number of vertices of the

arrangement $\mathcal{A}(H'')$ of H'' inside Δ, counting multiplicities (note that the multiplicity of a vertex is the product of the multiplicities of its defining hyperplanes); let $H''(\Delta)$ denote the (multi)-subset of hyperplanes of H'' crossing Δ.

We process the t cells of S iteratively one by one. The order they will be processed is carefully chosen (in contrast, a random order is used in the proof of [7], which is a major difference between our proof and that in [7]). We will assign indices to the cells following the reverse order they are processed. Suppose we have processed $t-i$ cells, which have been assigned indices and denoted by $\Delta_t, \Delta_{t-1}, \ldots, \Delta_{i+1}$. In the next iteration, we will find a cell among the unprocessed cells of S and process it (and the cell will be assigned index i as Δ_i). Suppose we now have a multiset H_i after cell Δ_{i+1} is processed (initially let $H_{t+1} = H'$). Let S_i denote the subset of unprocessed cells of S. Hence, $|S_i| = i$.

For each cell $\Delta \in S_i$, define

$$A_\Delta = \left(\frac{N_\Delta(H_i)}{b}\right)^{1/d}, \quad B_\Delta = \frac{|H_i(\Delta)|}{b^{1/(d-1+\beta)}}.$$

Define S_{i1} as the subset of cells $\Delta \in S_i$ with $A_\Delta \geq B_\Delta$. Let $S_{i2} = S \setminus S_{i1}$. If $|S_{i1}| \geq i/2$, then we define Δ_i as the cell Δ of S_{i1} that minimizes the value A_Δ; otherwise define Δ_i as the cell Δ of S_{i2} that minimizes B_Δ. Note that the above way of defining Δ_i is a key difference from Chan's approach [7], where Δ_i is chosen from S_i randomly. We now process Δ_i in the following three steps (which is similar to Chan's approach).

1. Construct a $(1/r_i)$-cutting for H_i inside Δ_i with
$$r_i = c \cdot \min\left\{|H_i(\Delta_i)| \cdot \left(\frac{b}{N_{\Delta_i}(H_i)}\right)^{1/d}, b^{1/(d-1+\beta)}\right\},$$
for some constant c. By the cutting lemma, the number of subcells inside Δ_i is $O(N_{\Delta_i}(H_i) \cdot (r_i/|H_i(\Delta_i)|)^d + r_i^{d-1+\beta})$, which can be made at most $b/4$ for a sufficiently small c.
2. We further subdivide each subcell of Δ_i (e.g., using vertical cuts) so that each subcell contains at most $2n/(tb)$ points of P. The number of extra cuts is $O(b)$ as Δ_i contains at most $2n/t$ points of P. The total number of extra cuts for processing all t cells of S is at most $\frac{n}{2n/(tb)} + t = bt/2 + t$, and thus the total number of subcells after processing all t cells is at most $bt/4 + bt/2 + t = 3bt/4 + t$, which is at most bt for $b \geq 4$.
3. For each distinct hyperplane $h \in H_i$, multiply the multiplicity of h in H_i by $(1+1/b)^{\lambda_i(h)}$, where $\lambda_i(h)$ is the number of subcells of Δ_i crossed by h. Let H_{i-1} be the resulting multiset after this.

To prove Lemma 2, it remains to prove Bound (1).

In the third step, since each of the $O(b)$ subcells of Δ_i is crossed by at most $|H_i(\Delta_i)|/r_i$ hyperplanes of H_i, we have $\sum_{h \in H_i} \lambda_i(h) = O(b \cdot |H_i(\Delta_i)|/r_i)$. Since $\lambda_i(h) = O(b)$, we have

$$|H_{i-1}| = \sum_{h \in H_i} (1+1/b)^{\lambda_i(h)} = \sum_{h \in H_i} (1 + O(\frac{\lambda_i(h)}{b})) = |H_i| + \sum_{h \in H_i} O(\frac{\lambda_i(h))}{b}).$$

We thus obtain

$$|H_{i-1}| - |H_i| = O(\frac{1}{b} \cdot \sum_{h \in H_i} \lambda_i(h)) = O(|H_i(\Delta_i)|/r_i) = O(\alpha_i) \cdot |H_i|, \text{ where}$$

$$\alpha_i = \frac{|H_i(\Delta_i)|}{r_i \cdot |H_i|} = O\left(\frac{1}{|H_i|} \cdot \max\left\{\left(\frac{N_{\Delta_i}(H_i)}{b}\right)^{1/d}, \frac{|H_i(\Delta_i)|}{b^{1/(d-1+\beta)}}\right\}\right)$$
$$= O\left(\frac{1}{|H_i|} \cdot \max\{A_{\Delta_i}, B_{\Delta_i}\}\right).$$

After all t cells of S are processed, we have a multiset H_0. Recall that $H_{t+1} = H'$ and $|H'| = C \cdot m$. According to the above analysis, we have

$$|H_0| = |H_{t+1}| \cdot \Pi_{i=1}^{t}(1 + O(\alpha_i)) \leq C \cdot m \cdot \exp\left(O(\sum_{i=1}^{t} \alpha_i)\right). \tag{2}$$

The following lemma gives an upper bound for $\sum_{i=1}^{t} \alpha_i$.

▶ **Lemma 3.** $\sum_{i=1}^{t} \alpha_i = O\left(\frac{t^{1-1/d}}{b^{1/d}} + \frac{\kappa \cdot \log t}{b^{1/(d-1+\beta)}}\right)$.

By Lemma 3 and (2), we have

$$|H_0| \leq C \cdot m \cdot \exp\left(O\left(\frac{t^{1-1/d}}{b^{1/d}} + \frac{\kappa \cdot \log t}{b^{1/(d-1+\beta)}}\right)\right). \tag{3}$$

For any hyperplane $h \in H$, let $\lambda(h)$ be the total number of subcells crossed by h. Our goal is to prove that $\lambda(h)$ is bounded by (1). By the way H_0 is produced, we have $C \cdot (1+1/b)^{\lambda(h)} \leq |H_0|$. Hence,

$$\lambda(h) \leq \log_{1+1/b} \frac{|H_0|}{C} = O(b \cdot \log \frac{|H_0|}{C}) = O(b \log m + (bt)^{1-1/d} + b^{1-1/(d-1+\beta)} \cdot \kappa \cdot \log t).$$

This proves Lemma 2.

3.2 Constructing the partition tree

Using Lemma 2, we can construct a partition tree in the following theorem.

▶ **Theorem 4.** (Hierarchical Partition Theorem) *Given a set P of n points in $\mathbb{R}^d$ and a parameter r with $r_0 \leq r \leq n$ for a sufficiently large constant r_0, for $b = \log^{\rho} r$ for a sufficiently large constant $\rho > 0$, there exists a sequence $\Pi_0, \Pi_1, \ldots, \Pi_{k+1}$ with $k = \lfloor \log_b r \rfloor$ and $b' = \lceil r/b^k \rceil$, where each Π_i is a collection of disjoint simplicial cells (i.e., each cell is a simplex) whose union covers P with the following properties:*

1. *Π_0 has only one cell that is $\mathbb{R}^d$ and the number of cells of Π_i is $O(b' \cdot b^{i-1})$ for $i \geq 1$ (in particular, Π_{k+1} has $O(r)$ cells; the total number of cells in all collections is also $O(r)$).*
2. *Each cell of Π_i contains at least one point and at most $2n/(b' \cdot b^{i-1})$ points of P for all $i \geq 0$ (in particular, each cell of Π_{k+1} contains at most $2n/r$ points).*
3. *Each cell in Π_{i+1} is contained in a single cell of Π_i.*
4. *Each cell of Π_i contains $O(b)$ cells of Π_{i+1} for $i \geq 1$ and the cell of Π_0 contains $O(b')$ cells of Π_1.*
5. *Any hyperplane crosses at most $O((b' \cdot b^{i-1})^{1-1/d} + \log^{O(1)} n)$ cells of Π_i for all $i \geq 1$.*

All collections of cells can be constructed in $O(n^{d^2} \cdot (r^{2-2/d} + \log^{O(1)} n))$ time.

Proof. We only sketch the proof here. The details can be found in the full paper.

Let H be a set of $m = n^{O(1)}$ hyperplanes in $\mathbb{R}^d$. We can prove the following *critical lemma*: With respect to H, one can construct a sequence $\Pi_0, \Pi_1, \ldots, \Pi_{k+1}$ as stated in the theorem with the same properties except that Property (5) only works for any hyperplane in H; also, the construction time is bounded by $O(m^d \cdot (r^{2-2/d} + \log^{O(1)} n) + n \log n + r^{3-1/d} + r^2 \cdot \log^{O(1)} n + m \cdot r^{3-2/d})$.

To prove the above lemma, we first assume that r is a power of b. Then, $r = b^k$ and $b' = 1$. We apply Lemma 2 with H to construct the collections Π'_t iteratively for $t = 1, b, b^2, \ldots$ until $b^k = r$. More specifically, Π'_1 consists of a single cell that is $\mathbb{R}^d$, and Π'_{bt} is obtained by applying Lemma 2 on all cells of Π'_t. We then let $\Pi_0 = \Pi_1 = \Pi'_1$ and $\Pi_i = \Pi'_{b^{i-1}}$. By

Lemma 2, the properties (1)-(4) hold. The proof for Property (5) as well as the case where r is not a power of b are given in the full paper. The algorithm implementation for constructing all collections of cells is also presented in the full paper.

The above critical lemma only guarantees the crossing numbers for the hyperplanes in H. To make it work for any hyperplane in $\mathbb{R}^d$, we resort to the following lemma, which was known before, e.g., [7].

▶ **Lemma 5.** (Test Set Lemma [7]) *For a set P of n points in $\mathbb{R}^d$, we can compute in $O(n^d)$ time a set H (called* test set*) of $O(n^d)$ hyperplanes with the following property: for any set of cells each containing at least one point of P, if κ is the maximum number of cells crossed by any hyperplane of H, then the maximum number of cells crossed by any hyperplane is $O(\kappa)$.*

Now to prove Theorem 4, we just apply the above critical lemma with H as the test set given in Lemma 5, which can be computed in $O(n^d)$ time. Since $m = O(n^d)$ and $r \leq n$, we obtain the theorem from the above critical lemma. ◀

The collections of cells of Theorem 4 naturally form a tree structure, called a *hierarchical partition tree*, in which each node corresponds to a cell. Specifically, the only cell in Π_0 is the root. Cells of Π_{k+1} are the leaves. If a cell $\Delta \in \Pi_i$ contains another cell $\Delta' \in \Pi_{i+1}$, then Δ is the parent of Δ' and Δ' is a child of Δ. As such, each node of the tree has $O(b)$ children (the root has $O(b')$ children). Although the construction time of Theorem 4 is large, it can usually be reduced (e.g., to $O(n^{1+\epsilon})$ time) in applications by constructing an "upper" partition tree using other techniques (e.g., [17]) and then processing the leaves (each containing a small number of points) using Theorem 4, as will be demonstrated in the subsequent sections.

4 Simplex range counting

Given a set P of n points in $\mathbb{R}^d$, we wish to construct a data structure so that the number of points of P inside a query simplex can be quickly computed. If we set $r = n$ in Theorem 4, we can have a data structure of $O(n)$ space and $O(b \cdot n^{1-1/d})$ query time, which is not $O(n^{1-1/d})$ as $b = \Theta(\log n)$. In the following, we reduce the query time to $O(n^{1-1/d}/\log^{\Omega(1)} n)$.

For a region R in the plane, let $P(R)$ denote the subset of points of P in R.

Setting $r = n/b^{2d/(d-1)}$ with $b = \log^{\rho} n$, we apply Theorem 4 to obtain a partition tree T with collections Π_i, $0 \leq i \leq k+1$. The total number of cells is $O(r)$. For each cell Δ, we store $|P(\Delta)|$. By Theorem 4, each cell in Π_{k+1} contains $O(n/r) = O(b^{2d/(d-1)}) = O(\log^{2d\rho/(d-1)} n)$ points of P. The runtime to construct T is $O(n^{d^2+2})$.

Given a query simplex σ, starting from the root of T, for each cell Δ of T, if Δ is contained in σ, then we add $|P(\Delta)|$ to a total count. If Δ is completely outside σ, then we ignore it. Otherwise, a bounding halfplane of σ must cross Δ and we proceed on all children of Δ unless Δ is a leaf. In this way, we obtain a set V of $O(r^{1-1/d})$ leaf cells that are crossed by the bounding halfplanes of σ. The runtime is $O((r/b)^{1-1/d} \cdot b)$, which is $O(r^{1-1/d} \cdot b^{1/d})$.

We now build the data structure recursively on the points in $P(\Delta)$ for each leaf cell Δ of T. If $Q(n)$ is the query time, we have the following recurrence relation:

$$Q(n) = O(r^{1-1/d} \cdot b^{1/d}) + O(r^{1-1/d}) \cdot Q_1(n/r), \tag{4}$$

where $Q_1(\cdot)$ is the query time for each leaf cell Δ, which contains $O(n/r)$ points of P.

To solve $Q_1(n/r)$, we preprocess $P(\Delta)$ for each leaf cell $\Delta \in T$ recursively as above by using different parameters. Specifically, let $n_1 = |P(\Delta)|$, which is $O(n/r)$. Let $\tau > 0$ be an arbitrarily small constant to be set later. Setting $r_1 = n_1/\log^{\tau} n$ with $b_1 = \log^{\rho} n_1$, we

apply Theorem 4 to construct a partition tree $T(\Delta)$ in $O(n_1^{d^2+2})$ time. The total time for constructing $T(\Delta)$ for all leaf cells Δ of T is bounded by $O(n^{d^2+2})$. Using $T(\Delta)$ to handle queries on $P(\Delta)$ and following the same analysis as above, we obtain

$$Q_1(n_1) = O(r_1^{1-1/d} \cdot b_1^{1/d}) + O(r_1^{1-1/d}) \cdot Q_2(n_1/r_1), \tag{5}$$

where $Q_2(\cdot)$ is the query time for each leaf of $T(\Delta)$, which contains $O(n_1/r_1)$ points of P.

Combining (4) and (5) leads to (see the full paper for the detailed proof):

$$\begin{aligned} Q(n) &= O(r^{1-1/d} \cdot b^{1/d}) + O(r^{1-1/d}) \cdot Q_1(n/r) \\ &= O(r^{1-1/d} \cdot b^{1/d}) + O(r^{1-1/d} \cdot r_1^{1-1/d} \cdot b_1^{1/d}) + O(r^{1-1/d} \cdot r_1^{1-1/d}) \cdot Q_2(n_1/r_1) \\ &= O\left(\frac{n^{1-1/d}}{\log^{\Omega(1)} n}\right) + O\left(\left(\frac{n}{t}\right)^{1-1/d}\right) \cdot Q_2(t), \text{ where } t = \log^{\tau} n. \end{aligned} \tag{6}$$

In summary, the above first builds a partition tree T and then builds a partition tree $T(\Delta)$ for each leaf Δ of T (so there are two recursive steps but with different parameters). For notational convenience, we still use T to refer to the entire tree (by attaching $T(\Delta)$ for all leaves Δ), which has $O(n/t)$ leaves, each containing $O(t)$ points. The total space is bounded by $O(n)$ since there are only two recursive steps. In Section 4.1, by using the property that t is very small, we show that after $O(n)$ space and $O(n \log n)$ time preprocessing, each simplex range counting query on any leaf cell of T can be answered in $O(\log t)$ time (i.e., $Q_2(t) = O(\log t)$, which is $O(\log \log n)$; we consider the query on each leaf cell a *subproblem*). As such, we obtain that $Q(n) = O(n^{1-1/d}/\log^{\Omega(1)} n)$. The total preprocessing time is $O(n^{d^2+2})$, dominated by the time for constructing T. We thus have the following result.

▶ **Lemma 6.** *Given a set P of n points in $\mathbb{R}^d$, there is a data structure of $O(n)$ space that can compute the number of points of P in any query simplex in $O(n^{1-1/d}/\log^{\Omega(1)} n)$ time. The data structure can be built in $O(n^{d^2+2})$ time.*

We will reduce the preprocessing time to $O(n^{1+\epsilon})$ in Section 4.2.

4.1 Solving the subproblems

We first consider halfspace queries and then extend the techniques to the simplex case.

A basic data structure. Let A be a set of t points in $\mathbb{R}^d$. We first build a straightforward data structure (called a *basic data structure*) of $O(t^d)$ space in $O(t^{d+1})$ time that can answer each halfspace range counting query on A in $O(\log t)$ time.

Let H be the set of dual hyperplanes of A. We compute the arrangement $\mathcal{A}$ of H in $O(t^d)$ time and space [13]. We then build a point location data structure on $\mathcal{A}$, which can be done in $O(t^d)$ time and supports $O(\log t)$-time point location queries [11]. In addition, for each face f of $\mathcal{A}$, we compute the number of hyperplanes above f (resp., below f) and store these two numbers at f, e.g., by checking every hyperplane of A. This finishes our preprocessing, which can be easily done in $O(t^{d+1})$ time and $O(t^d)$ space. The preprocessing time can be reduced to $O(t^d)$, e.g., by taking an Eulerian tour of the dual graph of the arrangement.

Given a query halfspace σ, the goal is to compute the number of points of A inside σ. Without loss of generality, we assume that σ is an upper halfspace. Then, it is equivalent to computing the number of hyperplanes of H below p, where p is the dual point of the bounding hyperplane of σ. Using the point location data structure, we find the face f of $\mathcal{A}$ that contains p; f stores the number of hyperplanes of H below it and we return that number as our answer to the query. The query time is thus $O(\log t)$.

Handling halfspace queries. Next, we show that after $O(n \log n)$ time and $O(n)$ space preprorcessing we can answer each halfspace range counting query in $O(\log t)$ time on $P(\Delta)$ for each leaf cell Δ of our partition tree T.

We build an algebraic decision tree T_D for the arrangement construction algorithm [13] on a set of t hyperplanes in $\mathbb{R}^d$ so that each node of T_D corresponds to a comparison in the algorithm. The height of T_D is $O(t^d)$ and T_D has $2^{O(t^d)}$ leaves. Each leaf v of T_D corresponds to a "configuration" of t hyperplanes in the following sense. Let A be a set of t hyperplanes with indices $1, 2, \ldots, t$. Following T_D in a top-down manner, we can reach a leaf v such that all comparisons of the nodes in the path of T_D from the root to v are consistent with A; we say that A has the same configuration as v. Let A and B are two sets of t hyperplanes each. Let $\mathcal{A}_A$ and $\mathcal{A}_B$ be the arrangements of A and B, respectively. If the configurations of A and B are both the same as a leaf v of T_D, then there is a one-to-one correspondence between faces of $\mathcal{A}_A$ and faces of $\mathcal{A}_B$ such that if a face f of $\mathcal{A}_A$ corresponds to a face f' of $\mathcal{A}_B$, then the i-th hyperplane of A is above f if and only if the i-th hyperplane of B is above f'.

By the above observation, we do the following preprocessing. For each leaf v of T_D, let A_v be a set of t hyperplanes whose configuration corresponds to v (note that the sets A_v's for all leaves v can be constructed in $t^d \cdot 2^{O(t^d)}$ time by following T_D, which basically enumuerates all possible configurations for the arrangements of a set of t hyperplanes). We construct the above basic data structure on A_v, denoted by $\mathcal{D}_v$. Doing this for all leaves of T_D takes $t^d \cdot 2^{O(t^d)}$ time and space, which is $O(n)$ since $t = \log^\tau n$ if we choose a small enough τ.

For each leaf cell Δ of T, recall $|P(\Delta)| \leq t$ (if $|P(\Delta)| < t$, we add $t - |P(\Delta)|$ dummy points to $P(\Delta)$ so that $P(\Delta)$ has exactly t points). Let $H(\Delta)$ be the set of dual hyperplanes of the points of $P(\Delta)$. We arbitrarily assign indices to the hyperplanes of $H(\Delta)$. Following T_D, we find the leaf v of T_D that has the same configuration as $H(\Delta)$, which can be done in time linear in the height of T_D, i.e., $O(t^d)$; we associate v with Δ. Since T has $O(n/t)$ leaves, doing this for all leaves of T takes $O(n/t \cdot t^d)$ time, which is $O(n \log n)$ if τ is small enough.

Given a query halfspace σ, suppose we want to compute the number of points of $P(\Delta)$ inside σ for a leaf cell Δ of T. This can be done in $O(\log t)$ time as follows. Let v be the leaf of T_D associated with Δ. Let p be the dual point of the bounding hyperplane of σ. Without loss of generality, we assume that σ is an upper halfspace. Hence it is equivalent to finding the number of hyperplanes of $H(\Delta)$ below p. We apply the point location query algorithm using the data structure $\mathcal{D}_v$ with p, but whenever the algorithm attempts to use the i-th hyperplane of A_v to make a comparison, we use the i-th hyperplane of $H(\Delta)$ instead. The point location algorithm will eventually return a face, which stores the number of hyperplanes below it; we return that number as our answer to the query of σ.

Handling simplex queries. We can easily extend the above idea to simplex queries, by using a multilevel data structure. The details are given in the full paper. In summary, with $O(n)$ space and $O(n \log n)$ time preprocessing, a simplex range counting query on $P(\Delta)$ for any leaf cell Δ of T can be answered in $O(\log t)$ time.

4.2 Reducing the preprocessing time

We now reduce the preprocessing time of Lemma 6 to $O(n^{1+\epsilon})$. The idea is to build an "upper partition tree" of $O(1)$ depth using Matoušek's method [17] so that each leaf has $O(n^\delta)$ points of P for a small constant $\delta > 0$ and then construct our data structure in Lemma 6 on each leaf (which form the "lower" part of the partition tree). The details are given below.

A *simplicial partition* for P is a collection $\Pi = \{(P_1, \sigma_1), (P_2, \sigma_2), \ldots, (P_m, \sigma_m)\}$, where the P_i's are pairwise disjoint subsets forming a partition of P, and each σ_i is a simplex containing all points of P_i. The subsets P_i's are called the *classes* and the simplices σ_i's are called *cells*, which may overlap. The *crossing number* of Π is the maximum number of cells crossed by any hyperplane. The following result is from [17].

▶ **Lemma 7.** ([17]) *For a set P of n points in $\mathbb{R}^d$ and a parameter $r \leq n^{\epsilon'}$ for any constant $\epsilon' < 1$, a simplicial partition $\Pi = \{(P_1, \sigma_1), (P_2, \sigma_2), \ldots, (P_{O(r)}, \sigma_{O(r)})\}$ whose classes satisfy $|P_i| \leq n/r$ and whose crossing number is $O(r^{1-1/d})$ can be constructed in $O(n \log r)$ time.*

We build a partition tree T by Lemma 7 recursively, until we obtain a partition of P into subsets of sizes $O(n^\delta)$ for a small enough constant $\delta > 0$ to be fixed later, which form the leaves of T. Each inner node v of T corresponds to a subset P_v of P as well as a simplicial partition Π_v of P_v, which form the children of v. At each child u of v, we store the cell σ_u of Π_v containing P_u and also store the cardinality $|P_u|$. The simplicial partition Π_v is constructed using Lemma 7 with $r = n^{(1-\delta)/k}$ for a constant integer k to be fixed later, i.e., every internal node of T has $O(r)$ children. If we recurse k times, i.e., the depth of T is k, then the subset size of each leaf of T is $O(n^\delta)$. Hence, the number of leaves of T is $O(r^k)$. Next, for each leaf v of T, we construct the data structure of Lemma 6 on P_v, denoted by $\mathcal{D}_v$, in $O(|P_v|^{d^2+2})$ time. Since $|P_v| = O(n^\delta)$, we can make δ small enough so that the total time of Lemma 6 on all leaves of T is $O(n^{1+\epsilon})$. This finishes the preprocessing, which takes $O(n^{1+\epsilon})$ time. The space of the data structure is $O(n)$ since the depth of T is $O(1)$.

Given a query simplex σ, starting from the root of T, for each node v, we check whether σ contains the cell σ_v. If yes, we add $|P_v|$ to the total count. Otherwise, if the boundary of σ crosses σ_v, we proceed to the children of v. In this way, we reach a set V of leaves v of T whose cells σ_v are crossed by the bounding hyperplanes of σ. Since the number of leaves of T is $O(r^k)$ and the depth of T is k, which is a constant, the size of V is $O(r^{k\cdot(1-1/d)})$. If $Q(m)$ is the query time for a subset of size m, then we have the following recurrence

$$Q(m) = O(r) + O(r^{1-1/d}) \cdot Q(m/r), \text{ where } r = n^{(1-\delta)/k} \text{ with } n \text{ as the global input size.}$$

Starting with $m = n$ and recursing k times (using the same r) gives us

$$Q(n) = O(r^{(k-1)\cdot(1-1/d)+1}) + O(r^{k\cdot(1-1/d)}) \cdot Q(n/r^k).$$

Using $r = n^{(1-\delta)/k}$, by setting k to a constant integer larger than $(1/\delta - 1)/(d-1)$, we have

$$Q(n) = O(n^{1-1/d-\delta'}) + O(r^{k\cdot(1-1/d)}) \cdot Q(n/r^k), \tag{7}$$

for another small constant $\delta' > 0$.

Finally, for each leaf node $v \in V$ (i.e., those subproblems $Q(n/r^k)$ in (7)), we use the data structure $\mathcal{D}_v$ to compute the number of points of P_v in σ, in $O(|P_v|^{1-1/d}/\log^{\Omega(1)}|P_v|)$ time by Lemma 6, which is $O((n/r^k)^{1-1/d}/\log^{\Omega(1)} n)$ as $|P_v| = O(n/r^k)$. Plugging this into (7) gives us $Q(n) = O(n^{1-1/d}/\log^{\Omega(1)} n)$. We thus conclude as follows.

▶ **Theorem 8.** *Given a set of n points in $\mathbb{R}^d$, there is a data structure of $O(n)$ space that can compute the number of points in any query simplex in $O(n^{1-1/d}/\log^{\Omega(1)} n)$ time. The data structure can be built in $O(n^{1+\epsilon})$ time for any $\epsilon > 0$.*

5 Simplex range stabbing and segment intersection searching

Given a set S of n simplices in $\mathbb{R}^d$, the problem is to construct a data structure to compute the number of simplices that contain a query point (we also say that those simplices are *stabbed* by the query point). In their data structure, Chan and Zheng [9] utilized a randomized hierarchical partition. We follow their approach and instead use our deterministic hierarchical

partition in Theorem 4. Extra effort needs to be taken because the degree of the partition tree in [7] is $O(1)$ while ours is logarithmic. In the following, we first briefly review Chan and Zheng's approach [9] and then discuss how to make changes.

A review of Chan and Zheng's randomized approach [9]. Each simplex is bounded by $d+1$ hyperplanes. A point p stabs a simplex s if p is in the "correct" side of every bounding hyperplane of s (to simplify the discussion, we assume these are lower halfplanes). In the dual space, this is equivalent to having the dual hyperplane of p above the dual point of each bounding hyperplane of s. Therefore, we have the following problem in the dual space. Given a set S^* of n $(d+1)$-tuples of points in $\mathbb{R}^d$, we wish to construct a data structure to compute the number of tuples whose points are all below a query hyperplane p^*. We can solve the problem using a multi-level data structure as follows.

Let P be the set of the $(d+1)$-th points of all tuples of S^*. Apply the simplicial partition of Lemma 7 to P to obtain a partition $\Pi = \{(P_i, \sigma_i)\}$, with a parameter r to be fixed later. Given a query hyperplane p^*, for every cell σ_i of Π that is crossed by p^*, we recurse on the subset of tuples whose $(d+1)$-th points are in P_i (i.e., apply Lemma 7 on P_i recursively). For each cell σ_i completely below p^*, we recurse on the subset of tuples whose $(d+1)$-th points are in P_i but as a level-d problem (the original problem is a level-$(d+1)$ problem; the recurrence stops at a level-0 problem in which we only need to return the size of the subset). As analyzed in [7], choosing $r = n^\epsilon$ with n as the global input size (i.e., value r is fixed for all levels of the recursion and this makes the recursion depth $O(1)$) can obtain an $O(n)$ space data structure with $O(n^{1-1/d+\epsilon})$ query time, for any $\epsilon > 0$. The preprocessing time is $O(n \log n)$. We summarize this (deterministic) result in the lemma below.

▶ **Lemma 9.** ([9]) *Given a set of n simplices in $\mathbb{R}^d$, one can build a data structure of $O(n)$ space in $O(n \log n)$ time so that the number of simplices containing a query point can be computed in $O(n^{1-1/d+\epsilon})$ time, for any $\epsilon > 0$.*

To improve the query time, Chan and Zheng [9] reduced the problem to a special case where the first d points in each tuple of S^* all lie in $\mathbb{R}^{d-1}$ (equivalently, in the primal setting all but one bounding halfplanes of each input simplex are parallel to the d-th axis). As such, we can reduce some subproblems in the recurrence to the $(d-1)$-dimensional problem and then solve these subproblems by Lemma 9 (where d becomes $d-1$). If $Q(n)$ (resp., $Q'(n)$) is the query time for the problem in $\mathbb{R}^d$ (resp., $\mathbb{R}^{d-1}$), then we have the following recurrence:

$$Q(n) = O(r^{1-1/d}) \cdot Q(n/r) + O(r) \cdot Q'(n/r). \tag{8}$$

By Lemma 9, $Q'(n) = O(n^{1-1/(d-1)+\epsilon})$. If we choose $r = n^\delta$ for a small constant $\delta > 0$, the recursion depth is $O(\log \log n)$ and thus the space is $O(n \log \log n)$. The query time is $O(n^{1-1/d} \log^{O(1)} n)$ due to a constant-factor blowup. To improve the query time to $O(n^{1-1/d})$, a randomized hierarchical partition was used in [9] to replace all recursive simplicial partitions in the preprocessing algorithm for Lemma 9. Here to achieve a deterministic result, we instead use our deterministic hierarchical partition in Theorem 4, as follows.

Our deterministic result. With $r = n/b^c$ for a sufficiently large constant c and $b = \log^\rho n$, we apply Theorem 4 to P to obtain a partition tree T consisting of Π_j, $0 \le j \le k+1$. By Theorem 4, Π_j has $O(b' \cdot b^{j-1})$ cells and each cell has at most $2n/(b' \cdot b^{j-1})$ points of P. We define a sequence of numbers $t_0 < t_1 < \cdots < t_l$ as follows. For each i, $0 \le i \le l$, define $t_i = b' \cdot b^{j_i - 1}$ (and thus $|\Pi_{j_i}| = O(t_i)$), with $j_0 = 1$ and

$$j_i = \left\lceil \log_b \frac{r^{1-(1-\epsilon)^i}}{b'} \right\rceil + 1, \quad 1 \le i \le l,$$

where l is the smallest integer so that $r/b < b' \cdot b^{j_l-1}$. Hence, $l = O(\log\log r)$, which is $O(\log\log n)$. Note that $b' \cdot b^{j_l-1} \leq r$ always holds. By definition, $t_0 = b'$ and t_i is equal to $r^{1-(1-\epsilon)^i}$ within a factor of b for $i \geq 1$.

We replace the recursive partitions in the preprocessing algorithm of Lemma 9 with this sequence of partitions of T: $\Pi_0, \Pi_{j_0}, \Pi_{j_1}, \ldots, \Pi_{j_l}$. As $l = O(\log\log n)$, the space of the data structure is still $O(n\log\log n)$. The cells of the last partition Π_{j_l} will be further preprocessed later. The following lemma analyzes the query time.

▶ **Lemma 10.** *The query time excluding the time spent on the cells of Π_{j_l} is bounded by $O(b \cdot r^{1-1/d} \cdot (n/r)^{1-1/(d-1)+\epsilon})$.*

Lemma 10 leads to the following.

▶ **Corollary 11.** *For a sufficiently large constant c, the query time excluding the time spent on the cells of Π_{j_l} is bounded by $O(n^{1-1/d}/b)$.*

Proof. By Lemma 10, it suffices to show that $b \cdot r^{1-1/d} \cdot (n/r)^{1-1/(d-1)+\epsilon} = O(n^{1-1/d}/b)$, which is equivalent to $b^2 \cdot (n/r)^{1-1/(d-1)+\epsilon} = O((n/r)^{1-1/d})$. This in turn is equivalent to $b^2 = O((n/r)^{1/(d-1)-1/d-\epsilon})$, which holds for a sufficiently large c since $n/r = b^c$. ◀

Preprocessing cells of Π_{j_l}. Recall that Π_{j_l} has $O(b' \cdot b^{t_l-1})$ cells each containing $2n/(b' \cdot b^{t_l-1})$ points of P, and $r/b < b' \cdot b^{t_l-1} \leq r$. Let $r' = b' \cdot b^{t_l-1}$. By Corollary 11, we obtain the following recurrence on the query time $Q(n)$: $Q(n) = O(n^{1-1/d}/b) + O(r'^{1-1/d}) \cdot Q_1(n/r')$, where $Q_1(n/r')$ is the query time for each subproblem of size $O(n/r')$ since each cell of Π_{j_l} contains $O(n/r')$ points of P. As $b = \log^{\rho} n$ and $r/b < r' \leq r$, we can write

$$Q(n) = O(n^{1-1/d}/\log^{\Omega(1)} n) + O((n/n_1)^{1-1/d}) \cdot Q_1(n_1), \tag{9}$$

with $n_1 = \log^{\Theta(1)} n$.

For notational convenience, let T refer to the sequence of partitions $\Pi_0, \Pi_{j_0}, \Pi_{j_1}, \ldots, \Pi_{j_l}$, and cells of Π_{j_l} form the leaves of T.

To solve $Q_1(n_1)$ in (9), we preprocess $P(\Delta)$ for each leaf cell $\Delta \in T$ recursively as above by using different parameters. Specifically, let $\tau > 0$ be an arbitrarily small constant to be set later. Setting $r_1 = n_1/\log^{\tau} n$, we apply Theorem 4 to construct a partition tree $T(\Delta)$ with $b_1 = \log^{\rho} n_1$ in $O(n_1^{d^2+2})$ time. As above (using the new parameters r_1 and b_1), we only use a subset of partitions of $T(\Delta)$ in our query data structure. For notational convenience, we use $T(\Delta)$ to refer to those partitions. The total time for constructing $T(\Delta)$ for all leaves $\Delta \in T$ is bounded by $O(n^{d^2+2})$ and the total space is still $O(n\log\log n)$.

With the same analysis as Lemma 10, we can obtain that the query time on $T(\Delta)$ excluding the time spent on the leaf cells of $T(\Delta)$ is $O(b_1 \cdot r_1^{1-1/d} \cdot (n_1/r_1)^{1-1/(d-1)+\epsilon})$. Define r_1' with respect to r_1 in the same way as above for r' with respect to r, i.e., $T(\Delta)$ has $O(r_1')$ leaf cells and each cell contains $O(n_1/r_1')$ points of P. As above, $r_1/b_1 < r_1' \leq r_1$ holds. Consequently, we obtain $Q_1(n) = O(b_1 \cdot r_1^{1-1/d} \cdot (n_1/r_1)^{1-1/(d-1)+\epsilon}) + O(r_1'^{1-1/d}) \cdot Q_2(n_1/r_1')$, where $Q_2(n_1/r_1')$ is the query time for each leaf cell of $T(\Delta)$. Let $t = n_1/r_1'$. We have

$$Q_1(n) = O(b_1 \cdot r_1^{1-1/d} \cdot (n_1/r_1)^{1-1/(d-1)+\epsilon}) + O((n_1/t)^{1-1/d}) \cdot Q_2(t). \tag{10}$$

Combining (9) and (10) leads to:

$$Q(n) = O\left(\frac{n^{1-1/d}}{\log^{\Omega(1)} n}\right) + O\left(\left(\frac{n}{t}\right)^{1-1/d}\right) \cdot Q_2(t). \tag{11}$$

Since $t = n_1/r'_1$ and $r_1/b_1 < r'_1 \leq r_1$, we have $n/t = n/n_1 \cdot r'_1 \leq n/n_1 \cdot r_1 = n/\log^{\tau} n$. Therefore, we further obtain from the above

$$\begin{aligned} Q(n) &= O\left(\frac{n^{1-1/d}}{\log^{\Omega(1)} n}\right) + O\left(\left(\frac{n}{\log^{\tau} n}\right)^{1-1/d}\right) \cdot Q_2(t) \\ &= O\left(\frac{n^{1-1/d}}{\log^{\Omega(1)} n}\right) + O\left(\frac{n^{1-1/d}}{\log^{\Omega(1)} n}\right) \cdot Q_2(t). \end{aligned} \tag{12}$$

Since $t = n_1/r'_1$ and $r_1/b_1 < r'_1 \leq r_1$, we have $t \leq n_1/r_1 \cdot b_1 = O(\log^{\tau} n \cdot \log^{\rho} \log n)$. Therefore, we can write $t = O(\log^{\tau'} n)$ for another arbitrarily small constant τ'.

In summary, the above first builds a partition tree T and then builds partition trees $T(\Delta)$ for all leaf cells Δ of T (using different parameters). For notational convenience, we still use T to refer to the entire tree (i.e., attaching all leaf trees $T(\Delta)$ to T), which has $O(n/t)$ leaves, each containing $O(t)$ points. The space is bounded by $O(n \log\log n)$ and the total preprocessing time is $O(n^{d^2+2})$. Since t is tiny, we show in the full paper that after additional $O(n \log n)$ time and $O(n)$ space preprocessing, each query on any leaf cell of T can be answered in $O(\log t)$ time (i.e., $Q_2(t) = O(\log\log n)$). Consequently, the query time is bounded by $O(n^{1-1/d}/\log^{\Omega(1)} n)$. We can reduce the preprocessing time to $O(n^{1+\epsilon})$ using idea similar to Section 4.2 (i.e., build an "upper partition tree" of O(1) depth on top using simplicial partitions [17]). We conclude with the following theorem.

▶ **Theorem 12.** *Given a set of n simplices in $\mathbb{R}^d$, one can build a data structure of $O(n \log\log n)$ space so that the number of simplices containing a query point can be computed in $O(n^{1-1/d}/\log^{\Omega(1)} n)$ time. The preprocessing time is $O(n^{1+\epsilon})$ for any $\epsilon > 0$.*

5.1 Other related problems

As studied in [9], some related problems (as stated in the following two theorems) can be solved by similar techniques. For each problem, we basically follow the same high-level algorithmic framework as [9] but use our deterministic partition tree instead of the randomized one in [7]; this is very similar to the above simplex stabbing counting problem, so we omit these details. For each problem, however, we still need to come up with a method to answer queries in $O(\log t)$ time for subproblems of small sizes $t = \log^{\tau} n$ (for an arbitrarily small constant $\tau > 0$) and these are discussed in the full paper.

▶ **Theorem 13.** *Given a set of simplices in $\mathbb{R}^d$, one can build a data structure of $O(n \log\log n)$ space so that the simplices containing a query point can be reported in $O(n^{1-1/d}/\log^{\Omega(1)} n + k)$ time, where k is the output size. The preprocessing time is $O(n^{1+\epsilon})$ for any $\epsilon > 0$.*

▶ **Theorem 14.** *Given a set of n segments in the plane, one can build a data structure of $O(n \log\log n)$ space so that the number of segments intersecting a query segment can be computed in $O(\sqrt{n}/\log^{\Omega(1)} n)$ time (these segments can be reported in additional $O(k)$ time, where k is the output size). The preprocessing time is $O(n^{1+\epsilon})$ for any $\epsilon > 0$.*

6 Segment intersection detection

Let S be a set of n line segments in $\mathbb{R}^2$. We wish to build a data structure to decide whether a query line intersects any segment of S. The problem can be solved by Theorem 14. This section presents a new method that only needs $O(n)$ space while the query time is the same.

Let P be the set of $2n$ endpoints of the segments of S. With $r = 4n/b^3$ and $b = \log^\rho n$, we apply Theorem 4 to P to obtain a partition tree T consisting of collections Π_i, $0 \le i \le k+1$. By Theorem 4, the total number of cells is $O(r)$, each cell in Π_{k+1} contains at most $4n/r = \log^{3\rho} n$ points of P, and the runtime to construct T is $O(n^6)$. For each node $v \in T$, let $\Delta(v)$ denote the corresponding cell of v, which is a triangle in $\mathbb{R}^2$.

The following is a result from the previous work [22] for a special case of the problem. We will use the result as a subroutine in our approach.

▶ **Lemma 15.** ([22]) *If all segments of S intersect a given line segment, then one can build a data structure of $O(n)$ space in $O(n \log n)$ time so that whether a query line intersects any segment of S can be determined in $O(\log n)$ time.*

We store the segments of S in the partition tree T as follows (the idea is similar to [22]). For each segment $s \in S$, starting from the root, for each node v whose cell $\Delta(v)$ contains s (which is true initially when v is the root), if v is a leaf, then we store s at v (let S_v denote the set of all such segments stored at v). Otherwise, we check every child of v. If v has a child u whose cell $\Delta(u)$ contains s, then we proceed on u. Otherwise, for each child u, if $\Delta(u)$ contains an endpoint of s, then since $\Delta(u)$ does not contain s, s must intersect an edge e of $\Delta(u)$; we store s at e (let S_e denote the set of segments stored at e); note that since s has two endpoints, there are two such edges e but it suffices to store s in one such edge. This finishes the algorithm for storing s, which takes $O(b \log n)$ time. Because s is stored at either a leaf or a cell edge, the total space for storing all segments is $O(n)$. The time is $O(nb \log n)$.

Next, for each edge e of each cell of T, since all segments of S_e intersect e, we preprocess S_e using Lemma 15. Doing this for all cell edges e of T takes $O(n \log n)$ time and $O(n)$ space. For those segments stored in S_v for all leaves v, we will preprocess them into a data structure $\mathcal{D}_v$. Before discussing $\mathcal{D}_v$, we describe the query algorithm and analyze the time complexity.

Given a query line ℓ, starting from the root of T, for each node v, assume that ℓ intersects the boundary of $\Delta(v)$, which is true initially when v is the root. If v is a leaf, then we call the data structure $\mathcal{D}_v$ to check whether ℓ intersects a segment of S_v. Otherwise, for each child u of v, for each edge e of $\Delta(u)$, we apply the query algorithm of Lemma 15 to check whether ℓ intersects a segment of S_e; further, if ℓ crosses $\Delta(u)$, then we proceed on u. The following lemma justifies the correctness of the query algorithm.

▶ **Lemma 16.** *The query algorithm works correctly.*

Proof. If the query algorithm detects an intersection, then it is obviously true that ℓ intersects a segment of S. On the other hand, suppose ℓ intersects a segment s, say, at a point p. We argue that the query algorithm must detect an intersection. Indeed, according to our query algorithm, all nodes u of T whose cells $\Delta(u)$ are crossed by ℓ will be processed. If s is stored at a leaf v, then s is contained in $\Delta(v)$. Since ℓ intersects s, ℓ must cross $\Delta(v)$, and thus v must be processed and the data structure $\mathcal{D}_v$ will detect an intersection between ℓ and S_v.

If s is not stored at a leaf, then there must exist an internal node v such that $s \in \Delta(v)$ and s is not in $\Delta(u)$ for any child u of v. According to our preprocessing algorithm, s must be stored in $S_{e'}$ for an edge e' of some cell $\Delta(u)$ of a child u of v. Since $p \in s \subseteq \Delta(v)$ and $p \in \ell$, ℓ must cross $\Delta(v)$. Therefore, our query algorithm will process v by applying the query algorithm of Lemma 15 on S_e for every edge e of every child cell of v. When it is applied to $S_{e'}$, the intersection will be detected. ◀

We now analyze the query time. Recall that T has $O(r)$ leaves. By Theorem 4, the total number internal nodes of T whose cell boundaries are crossed by ℓ is $O(\sqrt{r/b})$, and for each such node, we need to call the query algorithm of Lemma 15 $O(b)$ times and each call

takes $O(\log n)$ time. As such, the query time other than the time spent on calling $\mathcal{D}_v$ for those leaves v whose cell boundaries are crossed by ℓ (let V be the set of all such cells) is $O(\sqrt{r/b} \cdot b \cdot \log n) = O(\sqrt{n}/b \cdot \log n) = O(\sqrt{n}/\log^{\rho-1} n)$ since $r = 4n/b^3$ and $b = \log^\rho n$. By Theorem 4, $|V| = O(\sqrt{r}) = O(\sqrt{n/\log^{3\rho} n})$ and $|S_v| \leq 4n/r = \log^{3\rho} n$ for each leaf v of T.

Let $Q(n)$ be the query time. Following the above analysis, we obtain the following

$$Q(n) = O(\sqrt{rb} \cdot \log n) + O(\sqrt{r}) \cdot Q_1(n/r), \tag{13}$$

where $Q_1(\cdot)$ is the query time for each leaf $v \in V$, whose cell contains $O(n/r)$ points of P.

To solve $Q_1(n/r)$, we recursively process S_v for each leaf cell v of T as above by using different parameters. Specifically, let n_1 be the number of endpoints of S_v; hence $n_1 = O(n/r) = O(\log^{3\rho} n)$. Let $\tau > 0$ be an arbitrarily small constant to be set later. Setting $r_1 = n_1/\log^\tau n$, we apply Theorem 4 to construct a partition tree $T(v)$ with $b_1 = \log^\rho n_1$ in $O(n_1^6)$ time. The total time for constructing $T(\Delta)$ for all leaf cells Δ of T is thus bounded by $O(n^6)$. Using $T(v)$ to handle queries on S_v and following the above analysis, we obtain

$$Q(n_1) = O(\sqrt{r_1 b_1} \cdot \log n_1) + O(\sqrt{r_1}) \cdot Q_2(n_1/r_1), \tag{14}$$

where $Q_2(\cdot)$ is the query time for each leaf cell of $T(v)$, which contains $O(n_1/r_1)$ points of P.

Combining (13) and (14) leads to:

$$Q(n) = O\left(\frac{\sqrt{n}}{\log^{\Omega(1)} n}\right) + O\left(\sqrt{\frac{n}{t}}\right) \cdot Q_2(t), \text{ where } t = \log^\tau n. \tag{15}$$

In summary, the above first builds a partition tree T and then builds partition trees $T(v)$ for all leaves v of T (using different parameters). For notational convenience, we use T to refer to the entire tree (by attaching all trees $T(v)$ to T), which has $O(n/t)$ leaves, each containing $O(t)$ points. The space is bounded by $O(n)$. As t is small, with $O(n \log n)$ additional time and $O(n)$ space preprocessing, each subproblem $Q(t)$ in (15) can be solved in $O(\log t)$ time. This makes the total query time $Q(n)$ bounded by $O(\sqrt{n}/\log^{\Omega(1)} n)$. We can also reduce the preprocesing time to $O(n^{1+\epsilon})$. See the full paper for the details.

▶ **Theorem 17.** *Given a set of n segments in the plane, there is a data structure of $O(n)$ space that can determine whether a query line intersects any segment in $O(\sqrt{n}/\log^{\Omega(1)} n)$ time. The data structure can be built in $O(n^{1+\epsilon})$ time for any $\epsilon > 0$.*

7 Ray-shooting among non-intersecting segments

Let S be a set of n line segments in the plane such that no two segments intersect. The problem is to build a data structure to compute the first segment hit by a query ray.

The following is a result from the previous work [22] for a special case of the problem. We will use it as a subroutine in our approach.

▶ **Lemma 18.** ([22]) *If all segments of S intersect a given line segment, then one can build a data structure of $O(n)$ space in $O(n \log n)$ time so that a ray-shooting query can be answered in $O(\log n)$ time.*

We build the partition tree T and store S in T in a way similar to the segment intersection detection problem in Section 6 except that we use Lemma 18 to preprocess S_e for each cell edge e of T. In addition, for each leaf v of T, we will build a data structure $\mathcal{D}_v$ on S_v.

Given a query ray ρ, starting from the root of T, for each node v, assume that ρ intersects the boundary of $\Delta(v)$, which is true initially when v is the root. If v is a leaf, then we use the data structure $\mathcal{D}_v$ to find the first segment of S_v hit by ρ as our candidate solution segment. Otherwise, for each child u of v, for each edge e of $\Delta(u)$, apply the query algorithm of Lemma 18 to find the first ray of S_e hit by ρ as a candidate; further, if ρ crosses $\Delta(u)$, then we proceed on u. Finally, among all candidate segments, we return the one whose intersection with ρ is closest to the origin of ρ. The correctness follows a similar argument as Lemma 16.

As in Section 6, for each leaf v of T, we construct the data structure $\mathcal{D}_v$ by preprocessing S_v recursively once. The query time analysis follows exactly the same method as in Section 6 since the query time of Lemma 18 is the same as that of Lemma 15, and thus we can also obtain the recurrence (15) with the same value of t. As t is small, with additional $O(n \log n)$ time and $O(n)$ space preprocessing, each subproblem $Q(t)$ in (15) can be solved in $O(\log t)$ time. This makes the total query time $Q(n)$ bounded by $O(\sqrt{n}/\log^{\Omega(1)} n)$. We thus have the following result.

▶ **Lemma 19.** *Given a set of n segments in the plane, there is a data structure of $O(n)$ space that can compute the first segment hit by a query ray in $O(\sqrt{n}/\log^{\Omega(1)} n)$ time. The data structure can be built in $O(n^6)$ time.*

As before, we can reduce the preprocesing time to $O(n^{1+\epsilon})$. See the full paper for the details.

▶ **Theorem 20.** *Given a set of n segments in the plane, there is a data structure of $O(n)$ space that can compute the first segment hit by a query ray in $O(\sqrt{n}/\log^{\Omega(1)} n)$ time. The data structure can be built in $O(n^{1+\epsilon})$ time for any $\epsilon > 0$.*

References

1 Pankaj K. Agarwal. Range searching, in *Handbook of Discrete and Computational Geometry,* C.D. Tóth, J. O'Rourke, and J.E. Goodman (eds.), pages 1057–1092. CRC Press, 3rd edition, 2017.

2 Pankaj K. Agarwal. Simplex range searching and its variants: a review. In *A Journey Through Discrete Mathematics*, pages 1–30. Springer, 2017. doi:10.1007/978-3-319-44479-6_1.

3 Pankaj K. Agarwal and Jiří Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993. doi:10.1137/0222051.

4 Pankaj K. Agarwal and Micha Sharir. Applications of a new space-partitioning technique. *Discrete and Computational Geometry*, 9:11–38, 1993. doi:10.1007/BF02189304.

5 Pankaj K. Agarwal and Micha Sharir. Pseudoline arrangements: Duality, algorithms, and applications. *SIAM Journal on Computing*, 34:526–552, 2005. doi:10.1137/S0097539703433900.

6 Reuven Bar-Yehuda and Sergio Fogel. Variations on ray shootings. *Algorithmica*, 11:133–145, 1994. doi:10.1007/BF01182772.

7 Timothy M. Chan. Optimal partition trees. *Discrete and Computational Geometry*, 47:661–690, 2012. doi:10.1145/1810959.1810961.

8 Timothy M. Chan and Da Wei Zheng. Hopcroft's problem, log-star shaving, 2D fractional cascading, and decision trees. *ACM Transactions on Algorithms*, 2023. doi:10.1145/3591357.

9 Timothy M. Chan and Da Wei Zheng. Simplex range searching revisited: How to shave logs in multi-level data structures. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1493–1511, 2023. doi:10.1137/1.9781611977554.ch54.

10 Bernard Chazelle. Lower bounds on the complexity of polytope range searching. *Journal of the American Mathematical Society*, 2(4):637–666, 1989. doi:10.2307/1990891.

11 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete and Computational Geometry*, 9(2):145–158, 1993. doi:10.1007/BF02189314.

12 Siu Wing Cheng and Ravi Janardan. Algorithms for ray-shooting and intersection searching. *Journal of Algorithms*, 13:670–692, 1992. doi:10.1016/0196-6774(92)90062-H.

13 Herbert Edelsbrunner, J. O'Rourke, and Raimund Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15:341–363, 1986. doi:10.1137/0215024.

14 Herbert Edelsbrunner and Emo Welzl. Halfplanar range search in linear space and $O(n^{0.695})$ query time. *Information Processing Letters*, 23:289–293, 1986. doi:10.1016/0020-0190(86)90088-8.

15 Leonidas J. Guibas, Mark H. Overmars, and Micha Sharir. Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques. In *Proceedings of the 1st Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 64–73, 1988. doi:10.1007/3-540-19487-8_7.

16 David Haussler and Emo Welzl. ϵ-nets and simplex range queries. *Discrete and Computational Geometry*, 2:127–151, 1987. doi:10.1007/BF02187876.

17 Jiří Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8(3):315–334, 1992. doi:10.1007/BF02293051.

18 Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete and Computational Geometry*, 10(1):157–182, 1993. doi:10.1007/BF02573972.

19 Jiří Matoušek. Geometric range searching. *ACM Computing Survey*, 26:421–461, 1994. doi:10.1145/197405.197408.

20 Mark H. Overmars, Haijo Schipper, and Micha Sharir. Storing line segments in partition trees. *BIT Numerical Mathematics*, 30:385–403, 1990. doi:10.1007/BF01931656.

21 Haitao Wang. Unit-disk range searching and applications. *Journal of Computational Geometry*, 14:343–394, 2023. doi:10.20382/jocg.v14i1a13.

22 Haitao Wang. Algorithms for subpath convex hull queries and ray-shooting among segments. *SIAM Journal on Computing*, 53:1132–1161, 2024. doi:10.1137/21M145118X.

23 Dan E. Willard. Polygon retrieval. *SIAM Journal on Computing*, 11:149–165, 1982. doi:10.1137/0211012.

24 F. Frances Yao. A 3-space partition and its applications. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 258–263, 1983. doi:10.1145/800061.808755.

25 F. Frances Yao, David P. Dobkin, Herbert Edelsbrunner, and Mike Paterson. Partitioning space for range queries. *SIAM Journal on Computing*, 18:371–384, 1989. doi:10.1137/0218025.

Smoothed Analysis of Online Metric Problems

Christian Coester
Department of Computer Science, University of Oxford, UK

Jack Umenberger
Department of Engineering Science, University of Oxford, UK

Abstract

We study three classical online problems – k-server, k-taxi, and chasing size k sets – through a lens of smoothed analysis. Our setting allows request locations to be adversarial up to small perturbations, interpolating between worst-case and average-case models. Specifically, we show that if the metric space is contained in a ball in any normed space and requests are drawn from distributions whose density functions are upper bounded by $1/\sigma$ times the uniform density over the ball, then all three problems admit polylog(k/σ)-competitive algorithms. Our approach is simple: it reduces smoothed instances to fully adversarial instances on finite metrics and leverages existing algorithms in a black-box manner. We also provide a lower bound showing that no algorithm can achieve a competitive ratio sub-polylogarithmic in k/σ, matching our upper bounds up to the exponent of the polylogarithm. In contrast, the best known competitive ratios for these problems in the fully adversarial setting are $2k-1$, ∞ and $\Theta(k^2)$, respectively.

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

Keywords and phrases Online Algorithms, Competitive Analysis, Smoothed Analysis, k-server, k-taxi, Metrical Service Systems

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.115

Funding *Christian Coester*: Funded by the European Union (ERC grant CCOO 101165139). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.
Jack Umenberger: Supported by the University of Oxford's Strategic Research Fund through the ZERO Institute.

1 Introduction

In online algorithms, an algorithm has to make decisions online while the input is being revealed, without knowledge of the future input. Traditionally, the majority of research in the field has analyzed problems in the worst-case regime, in terms of the *competitive ratio*, i.e., the largest possible ratio between the algorithm's performance and the optimum in hindsight. This perspective enables clean mathematical models and robust guarantees for algorithms that are satisfied regardless of the input they face. However, worst-case instances are often degenerate and unlikely to occur in practice. Tuning algorithms against such pathological scenarios can lead to worst-case-like performance even when actual inputs are much more favorable.

Moreover, the guarantees that can be achieved even under the most pessimistic assumptions are often weak by necessity. While tight competitive ratio bounds can appeal theoretically, algorithms are unlikely to be adopted in practice if the best promise is that they perform at most some large (e.g., polynomial or exponential) factor worse than optimal. Worse yet, for some problems such as the k-taxi problem, no finite bound on the competitive ratio is currently known for general metric spaces of infinitely many points, even if their diameter is bounded.


© Christian Coester and Jack Umenberger;
licensed under Creative Commons License CC-BY 4.0



The research community generally understands the limitations of worst-case analysis, and there have been many demands for and an increased interest in beyond-worst-case analysis in recent years [46]. A major success story of beyond-worst-case analysis has been Spielman and Teng's smoothed analysis [48], spectacularly explaining why the simplex algorithm for linear problems performs well in practice despite its exponential worst-case running time. They show that perturbing any instance by small Gaussian noise yields an instance which the simplex algorithm solves in expected polynomial time. By varying the size of the random perturbations, smoothed analysis allows to interpolate between worst-case and average-case settings. Other examples of smoothed analysis of the running time of offline problems include k-means [3, 2], TSP [23, 41, 24], local max-cut [1] and more general local search problems [29]. We refer to [46] for more background.

In the context of online problems, prior smoothed analysis has focused mostly on regret minimization in online learning [45, 33, 20, 37, 44, 35, 36, 7, 34, 22] and online discrepancy minimization [36, 4, 5]. In contrast, understanding is much more limited about the smoothed complexity of more dynamic online problems which are typically analyzed through their competitive ratio (rather than regret). The only examples we are aware of are the work of [6], which conduct smoothed competitive analysis of the multi-level feedback algorithm for non-clairvoyant scheduling, and [47], who analyze the work function algorithm for metrical task systems when cost function values are randomly perturbed.

Smoothed Competitive Analysis in Metric Spaces. Here, we initiate smoothed competitive analysis of online problems in metric spaces with randomly perturbed request locations. To do so, we study three classical online problems: namely, the *k-server* problem [40], its generalization the *k-taxi* problem [27, 18], and *chasing small sets* [17] (also known as metrical service systems, and equivalent to layered graph traversal [43, 26]). In these problems, the goal is to minimize movement in a metric space while serving a sequence of requests. In the k-server problem, the algorithm controls k servers and each request is a location that one of the servers must move to. The k-taxi problem generalizes this to the case where each request is a pair of locations, modelling the pick-up and drop-off location of a ride request. In chasing small sets, there is a single server, and each request is a set of up to k points that the server must move to.

On general metric spaces, the best known competitive ratio is $\Theta(k)$ for the k-server problem [38] and $\Theta(k^2)$ for chasing small sets [11]. For the k-taxi problem, the situation is even worse and it is not even known whether any finite competitive ratio can be achieved, even for $k = 4$ servers on a line metric. Better bounds exist only in special cases such as *finite* metrics of n points or bounded aspect ratio Δ, where the k-server problem admits competitive ratios of $O(\log n \log^2 k)$ and $O(\log \Delta \log^3 k)$ [14], and the k-taxi problem an $O(\log^3 \Delta \log^2(nk\Delta))$-competitive algorithm [32]. However, since n and Δ can be infinite in general, these bounds provide no improvement even in seemingly simple metrics such as the line metric or $[0, 1]^2$ with some norm.

We argue that smoothed analysis is a realistic model for these problems. For example, a customer's pick-up location in taxi apps is often determined through somewhat inaccurate GPS or by placing a pin on a map by hand, and even if an address is specified this usually does not correspond to an exact point but rather a small area on a map. Even in cases where exact precision in response to arriving requests is important, the way that inputs are generated is rarely fully adversarial and may be subject to noise. If smoothed analysis can drastically improve the competitive ratio of these problems, we believe that algorithms achieving these improved guarantees have a much higher chance of being adopted in practice than pessimistic worst-case methods.

We consider the setting where the metric space M is a bounded diameter subset of some normed space (and therefore contained in some ball B_M). For each time step, an adversary can choose a distribution from which the next request is drawn, subject to the constraint that the density functions of points appearing in the requests are upper bounded by $1/\sigma$ times the density of the uniform distribution over B_M. Thus, as $\sigma \to 0$, this converges to the classical worst-case setting, whereas $\sigma = 1$ means that request locations are drawn uniformly from $M = B_M$. Small values of σ can model, for example, request locations that are slightly perturbed in a small ball around the worst-case locations.

For all three problems, we obtain $\mathrm{polylog}(k/\sigma)$-competitive algorithms in this setting via simple reductions to the adversarial setting in finite metrics. Thus, for σ polynomial in k we obtain an exponential improvement over the known worst-case guarantees for k-server and chasing small sets, and an infinite improvement for the k-taxi problem. Our algorithms require no knowledge about σ (except for the k-taxi problem, where a non-zero lower bound on σ is requied), and beyond these smoothed bounds they still also achieve the known worst-case guarantees (where they exist, i.e., for k-server and chasing small sets).

Further Related Work. The k-server problem and an easier version of the k-taxi problem was previously studied under a stochastic setting different from smoothed analysis by Dehgani et al. [21]. They consider the case where each request is drawn independently from some distribution, and these distributions are known to the algorithm. For this setting, they give an approximation algorithm for the line and finite metrics that is competitive against the best *online* algorithm for the instance. In contrast, we do not assume knowledge of the distributions, and compare against the best offline algorithm for the input.

Several online problems have been studied in a stochastic setting where requests are drawn i.i.d. from some distribution, see e.g. [28, 30, 31]. This differs from our setting, as we do not require distributions to be identical or independent.

Another beyond-worst-case model that has received significant interest in recent years is the learning-augmented paradigm [39, 42], where an algorithm's input is augmented with additional predictions about future input or best behavior.

The k-server problem, introduced in [40], is considered one of the most fundamental problems in online algorithms and is often called the "holy grail of competitive analysis". For deterministic algorithms the $\Theta(k)$ bound is known to be tight [40, 38] up to a constant factor, and the k-server conjecture states that it is exactly k. For randomized algorithms, the aforementioned improvements for the special cases of finite metrics and bounded aspect ratios are known [14]. The randomized k-server conjecture, positing an $O(\log k)$-competitive algorithm for general metrics, was recently refuted in [12]. The k-taxi problem is known to be at least exponentially harder than the k-server problem at least for deterministic algorithms, with an $\Omega(2^k)$ lower bound and no known upper bound for general metrics [18]. For chasing small sets, the deterministic competitive ratio is known to lie in $O(k2^k) \cap \Omega(2^k)$ [26, 16], and the randomized competitive ratio is $\Theta(k^2)$ [11, 12].

Chasing small sets is a special case of the metrical task systems (MTS) problem [9]. In MTS, each request consists of a cost function, and the algorithm which moves the single server incurs both the "service cost" as well as the movement cost. Chasing small sets corresponds to MTS with request cost functions restricted to set-membership indicator functions. The k-server problem is also a special case of MTS, though not in the same metric space but rather the metric space of configurations.

1.1 Preliminaries

All the problems we consider are defined over a metric space (M, d). The input revealed online is a sequence of requests $r_1, r_2, \ldots, r_T$, each of which must be served immediately as specified below.

k-Server. The algorithm controls the movement of k servers located at points in M. Each request is a point $r_t \in M$, and must be served by moving one of the servers to r_t. The cost of the algorithm is defined as the total distance travelled by all servers.

k-Taxi. The algorithm controls the movement of k taxis located at points in M. Each request is a pair of points $r_t = (a_t, b_t) \in M^2$, representing the start and destination of a ride request. To serve it, the algorithm must choose a taxi and move it first to a_t and then to b_t. The cost is defined as the total distance of "empty runs" (i.e., distance travelled except for the travels from a_t to b_t).

Chasing Small Sets. The algorithm controls the movement of a single server located in M. Each request is a subset $r_t \subset M$ of cardinality at most k. To serve it, the algorithm must move the server to one of the points in r_t. The cost incurred by the algorithm is the total distance travelled.

Smooth Instances. Throughout this paper, we assume that the metric space M is a subset of $\mathbb{R}^m$ and the metric d is induced by a norm, i.e., $d(x, y) = \|x - y\|$ for all $x, y \in M$, where $\|\cdot\|$ is an arbitrary norm. We assume that M has bounded diameter, and denote by B_M a (closed) ball of smallest radius containing M, and by R_M its radius.

We consider the variant of the problems described above where each request r_t is drawn from some probability distribution D_t. These distributions need not be independent, i.e., the distribution D_t (and whether it exists or the sequence stops earlier) can be chosen by the adversary after the realizations of $r_1, \ldots, r_{t-1}$ have been determined (however, without knowledge of any random choices made by the online algorithm). For the k-server problem, D_t is a distribution over M, for the k-taxi over M^2 and for chasing small sets over M^{k_t} for some $k_t \le k$. We denote by D_{ti} the corresponding marginal distributions, where $i = 1$ for k-server, $i \in \{1, 2\}$ for k-taxi and $i \in [k_t]$ for chasing small sets.

In the absence of any restrictions on the probability distribution, this problem setting is equivalent to the familiar worst-case setting, as the adversary can choose a (degenerate) distribution that places all probability mass on the worst-case request sequence. Thus, we impose the following smoothness assumption:

▶ **Definition 1.** *Let $\sigma \in (0, 1]$. An instance is σ-smooth if for each marginal distribution D_{ti}, the probability of any measurable set $S \subseteq M$ is at most a factor $1/\sigma$ larger than under the uniform distribution over B_M.*

Equivalently, this means that the density function of each marginal distribution is upper bounded by $1/(\sigma \cdot \mathrm{vol}(B_M))$. The case $\sigma = 1$ corresponds to uniformly distributed requests, whereas $\sigma \to 0$ captures the classical worst-case setting.

Competitive Ratio in the Smoothed Model. For a (possibly randomized) algorithm A, we denote its expected cost on an instance I by $\mathrm{cost}(A, I)$. Denote by opt the offline algorithm that achieves the lowest possible cost on any instance. In the classical (worst-case) setting, an online algorithm is said to be c-competitive if

$$\mathrm{cost}(A, I) \le c \cdot \mathrm{cost}(\mathrm{opt}, I) + \alpha, \tag{1}$$

for all possible instances I, where α is a constant independent of I. We say that A is c-competitive on σ-smooth instances if inequality (1) holds when I is sampled as a σ-smooth instance and both sides of the inequality are replaced by their expectation over the randomness in I. We allow opt to serve the actual realization of the instance optimally, i.e., knowing the outcome of the random sampling process of I in advance. Of course, any competitive algorithm in this setting is also competitive if opt has to make its choices before knowing the outcomes of the future sampling process of I.

We will usually drop I from the notation when it is understood from the context.

η-Nets. An *η-net* of a metric space M is a subset $N \subseteq M$ satisfying the following properties:

- N is η-dense in M: For all $x \in M$ there exists $y \in N$ such that $d(x, y) \leq \eta$.
- N is η-separated: For all $x, y \in N$ it holds that $d(x, y) > \eta$.

Aspect Ratio. The aspect ratio of a metric space is the ratio between the largest and smallest non-zero distance.

1.2 Our Results

We obtain polylog(k/σ)-competitive algorithms for all three problems we consider, as summarized in the following theorems. By known algorithm combination techniques, our algorithms for k-server and chasing small sets still also achieve the same robustness guarantees as classical worst-case algorithms.

▶ **Theorem 2** (k-Server). *There exists an $O(\min\{k, \log(k/\sigma)\cdot\log^2 k\})$-competitive randomized algorithm for k-server on σ-smooth instances, even if σ is unknown.*

▶ **Theorem 3** (k-Taxi). *There exists an $O(\log^5(k/\sigma))$-competitive randomized algorithm for k-taxi on σ-smooth instances, provided a non-zero lower bound on σ is known.*

▶ **Theorem 4** (Chasing Small Sets). *There exists an $O(\min\{k^2, \log^2(k/\sigma)\})$-competitive randomized algorithm for chasing small sets on σ-smooth instances, even if σ is unknown.*

We complement our upper bounds with the following lower bound, ruling out algorithms with a sub-polylogarithmic dependence on k/σ.

▶ **Theorem 5** (Lower Bound). *For any $c < 1/2$, there is no $O(\log^c(k/\sigma))$-competitive randomized algorithm for general σ-smooth instances of k-server, k-taxi, or chasing small sets when the metric space is a ball in $O(\log k)$-dimensional ℓ_∞-space, even if all distributions D_t are identical and independent.*

1.3 Organization

All our algorithms are based on the same framework, as explained in the next section. The upper bound proofs are completed by separate arguments for bounding the offline cost for the three problems in Section 3. The polylogarithmic lower bound (Theorem 5) is proved in Section 4.

2 Basic Construction

To describe our algorithm, let us assume initially that σ is known. Later we will show how to drop this assumption.

For some suitable $\eta > 0$, we choose a finite η-net N of the metric space M. For each $x \in M$, we pick some point $y \in N$ satisfying $d(x, y) \leq \eta$ and call it the *projection of x onto N*, denoted by $\pi(x) = y$ (making an arbitrary choice in case there are several options). Similarly, for a request r_t we denote by $\pi(r_t)$ the projection of r_t, where each point in r_t is replaced by its projection. As the request sequence $r_1, r_2, \ldots, r_T$ in M is revealed, we simulate some online algorithm A_N that operates on finite metrics to serve the request sequence $\pi(r_1), \pi(r_2), \ldots, \pi(r_T)$ in metric space (N, d). Our online algorithm A_M for the original problem on M follows the configurations of A_N at all times, except for an additional movement from $\pi(r_t)$ to r_t and back at time t, to make sure A_M serves the original request sequence. (For example, for a taxi request (a_t, b_t), algorithm A_M first moves the same taxi as A_N to the point $\pi(a_t)$, then pays at most η to move from $\pi(a_t)$ to a_t; from here, it serves the ride request, which changes the taxi's location from a_t to b_t (at no cost); finally, it moves from b_t to $\pi(b_t)$ for another cost of at most η, restoring the invariant that A_M is in the same configuration as A_N.) Thus, the cost of A_M exceeds the cost A_N by at most 2η per request, and the total cost of A_M is at most $2\eta \cdot T + \eta \cdot k$ larger for a sequence of T requests, with the additional cost ηk for projecting the initial configuration onto N. (In the case of chasing small sets, the $\eta \cdot k$ term can also be replaced by η.)

2.1 Reduction to Worst-Case Setting

Our analysis is based on the observation that the smoothing model forces the offline optimal algorithm to incur a sufficiently large movement cost. In particular, we have:

▶ **Proposition 6.** *For σ-smooth instances of k-server, k-taxi, and chasing small sets, the offline optimal cost is at least $\Omega\left(R_M \cdot \left(\frac{\sigma}{\mathrm{poly}(k)}\right)^{1/m}\right)$ amortized per request in expectation, where $\mathrm{poly}(k) = O(k)$ for k-server and k-taxi and $\mathrm{poly}(k) = O(k^2)$ for chasing small sets.*

More precise statements for the respective problems along with a proof of Proposition 6 are given in Section 3.

If η is within a constant factor of the bound in Proposition 6 and algorithm A_N is c_N-competitive for the worst-case (non-smoothed) problem in N, then we can show that algorithm A_M described above is $O(c_N)$-competitive for the smoothed problem in M:

▶ **Lemma 7.** *If A_N is c_N-competitive in metric space (N, d), then algorithm A_M is $O(c_N)$-competitive on instances in M whose optimal offline cost is at least $\Omega(\eta)$ amortized per request.*

Proof. Denote by opt_M and opt_N, respectively, an optimal offline algorithm for the instance in M and the projected instance in N. By assumption

$$\mathrm{cost}(A_N) \leq c_N \cdot \mathrm{cost}(\mathrm{opt}_N) + \alpha \tag{2}$$

for some constant α. Denote by T the number of requests and by $\mathbb{E}[T]$ its expectation[1]. As argued above,

$$\mathrm{cost}(A_M) \leq \mathrm{cost}(A_N) + 2\eta \cdot \mathbb{E}[T] + \eta \cdot k, \tag{3}$$

[1] T may be random as the adversary can choose the stopping point depending on the realizations of earlier requests.

with the additional cost due to movement between $\pi(r_t)$ and r_t at each time. By a similar argument,

$$\text{cost}(\text{opt}_N) \leq \text{cost}(\text{opt}_M) + 2\eta \cdot \mathbb{E}[T], \tag{4}$$

as a possible offline algorithm to service the requests $\pi(r_1), \pi(r_2), \ldots, \pi(r_T)$ in N is to always be in the configuration of opt_M projected onto N, incurring at most an additional cost 2η per request by the triangle inequality (noting that, without loss of generality, opt_M moves at most one server/taxi per time step). Combining inequalities (2), (3) and (4) gives

$$\text{cost}(A_M) \leq c_N \cdot \text{cost}(\text{opt}_M) + (c_N + 1) \cdot 2\eta \cdot \mathbb{E}[T] + \eta \cdot k + \alpha.$$

Next, we make use of the assumption that the optimal offline cost is at least $\Omega(\eta)$ amortized per request in expectation, i.e., $\text{cost}(\text{opt}_M) \geq \Omega(\eta) \cdot \mathbb{E}[T] - \beta$ for some constant β. Substituting this into the previous inequality gives

$$\text{cost}(A_M) \leq O(c_N) \cdot \text{cost}(\text{opt}_M) + \eta \cdot k + \alpha + O(\beta \cdot c_N),$$

and the lemma follows since the additive term $\eta \cdot k + \alpha + O(\beta \cdot c_N)$ is a constant independent of the request sequence. ◀

2.2 Size of the Net N

Lemma 7 allows us to reduce smoothed instances on M to non-smooth instances on N, provided the offline optimal cost is at least $\Omega(\eta)$ per request. For each of the problems we consider, by known results for finite metric spaces, the competitive ratio c_N can be expressed as a function of the number of points in N and possibly the aspect ratio Δ of N. Specifically, for k-server, the (best known) competitive ratio in n-point metric spaces is $O(\log^2 k \cdot \log n)$ [14, 15], for k-taxi it is $O(\log^3 \Delta \cdot \log^2(nk\Delta))$ [32], and for chasing small sets (which is a special case of metrical task systems) it is $O(\log^2 n)$ [13, 19].

Thus, it remains to choose η (i) large enough to obtain an appropriate value for the cardinality and aspect ratio of N and (ii) small enough to ensure the lower bound in Proposition 6 becomes $\Omega(\eta)$.

The following lemma provides the desired relationship between the parameter η and the number of points in N.

▶ **Lemma 8.** *For each $\eta \in (0, R_M]$, there exists an η-net N of M of size $|N| \leq \left(\frac{3R_M}{\eta}\right)^m$.*

Proof. The proof is similar to [25, Claim 2.6]. We initialize N as a singleton set containing an arbitrary point from M; then as long as there exists a point $x \in M$ such that $d(x, y) > \eta$ for all $y \in N$, we add x to N. By construction, N is an η-net. Moreover, the balls of radius $\eta/2$ centered at the points in N are disjoint and contained in a ball of radius $R_M + \eta/2$ (namely, the ball obtained by extending the radius of B_M by $\eta/2$). The ratio between the volume of a ball of radius $\eta/2$ and the ball of radius $R_M + \eta/2$ is $\left(\frac{\eta/2}{R_M+\eta/2}\right)^m$. Thus, there can be at most $\left(\frac{R_M+\eta/2}{\eta/2}\right)^m \leq \left(\frac{3R_M}{\eta}\right)^m$ disjoint balls. ◀

2.3 Putting it all Together

We now combine the above statements to prove the following statement.

▶ **Proposition 9.** *For known σ, there exists a randomized online algorithm whose competitive ratio on σ-smooth instances is $O(\log(k/\sigma) \cdot \log^2 k)$ for k-server, $O(\log^5(k/\sigma))$ for k-taxi and $O(\log^2(k/\sigma))$ for chasing small sets.*

Proof. Let $\eta = 3R_M \cdot \left(\frac{\sigma}{\text{poly}(k)}\right)^{1/m}$, where $\text{poly}(k)$ is the function from Proposition 6. Then A_M is $O(c_N)$-competitive by Proposition 6 and Lemma 7. If $\eta > R_M$, then N can be chosen as the singleton containing only the center of the ball B_M, which trivially yields $c_N = 1$ and the theorems. Otherwise, by Lemma 8 the size of the η-net N is

$$n := |N| \leq \frac{\text{poly}(k)}{\sigma},$$

and since the diameter of M is at most $2R_M$, the aspect ratio of N is

$$\Delta \leq \frac{2R_M}{\eta} = \frac{2}{3}\left(\frac{\text{poly}(k)}{\sigma}\right)^{1/m}.$$

Thus, since $c_N = O(\log^2 k \cdot \log n)$ for k-server [14, 15], $c_N = O(\log^3 \Delta \cdot \log^2(nk\Delta))$ for k-taxi [32], and $c_N = O(\log^2 n)$ for metrical task system [13, 19], and chasing small sets is a special case of metrical task systems (in the same metric space), the result follows. ◀

We note that the competitive ratio for k-taxi could actually be improved by a factor $1/\min\{m, \log(k/\sigma)\}^3$. (Note that the $\log \Delta$ term stands for a quantity that is at least 1, so the improvement is not greater.) This would typically only yield a constant factor improvement though, considering practical applications of the k-taxi problem usually having a small dimension of at most 3.

2.4 Unknown σ and Robustness

If σ is unknown, and to achieve the robustness bounds in our main theorems, the idea is to simulate in parallel several instances of the above algorithm for different values of σ as well as a worst-case algorithm, and combine them using the technique of Blum and Burch [8].

▶ **Theorem 10** (Blum and Burch [8])**.** *Given $\epsilon > 0$ and ℓ online algorithms $A_1, \ldots, A_\ell$ for a problem that can be formulated as a metrical task system of diameter* diam*, there exists an online algorithm whose expected cost is at most*

$$(1+\epsilon) \cdot \min_{i \in [\ell]} \text{cost}(A_i) + O\left(\frac{\text{diam} \cdot \log \ell}{\epsilon}\right).$$

Chasing Small Sets. Since chasing small sets in a metric space (M, d) is a metrical task system in the same metric space, we can invoke the above theorem with $\epsilon = 1$, $\ell = \lceil \log_2 k \rceil + 1$ and using as A_i for $i = 1, \ldots, \ell - 1$ our algorithm with smoothness parameter $\sigma_i = 2^{-2^i}$, and as A_ℓ the $\Theta(k^2)$-competitive algorithm from [11] for chasing small sets in the fully adversarial setting.

Since $\text{diam} \leq 2R_M$, the additive term in Theorem 10 is a constant independent of the request sequence. Thus, the resulting algorithm achieves, up to a factor $1 + \epsilon \leq 2$, the same competitive ratio as the best of the algorithms A_i. If the true smoothness parameter $\sigma < 2^{-(\ell-1)} \leq 2^{-k}$, then $O(\min\{k^2, \log^2(k/\sigma)\})$-competitiveness follows since the minimum is attained by k^2. Otherwise, there exists i such that $\sigma_i \leq \sigma \leq \sqrt{\sigma_i}$. Since any σ-smooth instance is also σ_i-smooth, we achieve competitive ratio $O(\log^2(k/\sigma_i)) = O(\log^2(k/\sigma))$. This yields Theorem 4.

***k*-Server.** For the k-server problem (Theorem 2), the same argument works since k-server can be modelled as a metrical task system in the space of server configurations (whose diameter is k times the diameter of the original metric space). The only difference is that the algorithm A_ℓ for general metrics in the fully adversarial setting has competitive ratio $\Theta(k)$ [38].

***k*-Taxi.** It is not known how to model the k-taxi problem as a metrical task system, due to configuration changes that incur no cost.[2] Nonetheless, the proof of Theorem 10 in [8] extends to this setting without any change, so we can still apply the same combination method also to the k-taxi problem. However, since no competitive algorithm for the k-taxi problem on general metrics in the fully adversarial setting is known, we require a lower bound on σ to ensure that the number of combined algorithms is finite.

3 Bounding Offline Cost

3.1 *k*-Server

▶ **Lemma 11.** *For σ-smooth instances of the k-server problem, the optimal offline algorithm pays an expected cost of at least*

$$\mathrm{cost}(\mathrm{opt}_M) \geq \frac{R_M}{8} \cdot \left(\frac{\sigma}{8k}\right)^{1/m} \cdot \mathbb{E}[T] - c,$$

where $c > 0$ is a constant independent of the request sequence.

Proof. Let $\delta > 0$. Consider a subsequence $r_{t+1}, r_{t+2}, \ldots, r_{t+4k}$ of $4k$ consecutive requests. For $i = 0, \ldots, 4k$, we denote by S_i a δ-separated subset of $\{r_{t+1}, r_{t+2}, \ldots, r_{t+i}\}$, constructed as follows: $S_0 = \emptyset$, and for $i \geq 1$,

$$S_i = \begin{cases} S_{i-1} \cup \{r_{t+i}\} & \text{if } S_{i-1} \cup \{r_{t+i}\} \text{ is } \delta\text{-separated} \\ S_{i-1} & \text{otherwise.} \end{cases}$$

Note that $S_i = S_{i-1}$ if and only if r_{t+i} belongs to a ball of radius δ centered at a point in S_{i-1}. The volume of each such ball is at most a $\left(\frac{\delta}{R_M}\right)^m$ fraction of the volume of B_M, and thus its probability under D_{t+i} is at most $\left(\frac{\delta}{R_M}\right)^m / \sigma$. Hence, since there are at most $4k$ such balls of radius δ, the probability that $S_i = S_{i-1}$ is bounded as

$$P\left(S_i = S_{i-1}\right) \leq \frac{4k}{\sigma} \left(\frac{\delta}{R_M}\right)^m.$$

We now choose $\delta = R_M \cdot \left(\frac{\sigma}{8k}\right)^{1/m}$, so that the right hand side becomes $1/2$. In other words, in each step, another point is added to S_i with probability at least $1/2$. Thus, after $4k$ steps we have $|S_{4k}| \geq 2k$ with probability at least $1/2$. If this is the case, then any offline algorithm must pay at least $k \cdot \delta$ to serve these requests, regardless of the configuration it starts at: Indeed, at most k of the requests in S_{4k} can be served by a server that previously served no other request from S_{4k}. For the remaining $|S_{4k}| - k \geq k$ requests, the serving server must have moved at least distance δ since the previous time it served a request from S_{4k}, since

[2] Though it belongs to the more general class of metrical task systems with transformation [10].

S_{4k} is δ-separated. Thus, any subsequence of $4k$ consecutive requests contributes $k \cdot \delta/2$ to the expected optimal cost. Over a sequence of T requests, this leads to an expected optimal cost of

$$\begin{aligned}\operatorname{cost}(\operatorname{opt}_M) &\geq \mathbb{E}\left[\left\lfloor \frac{T}{4k} \right\rfloor\right] \cdot \frac{k \cdot \delta}{2} \\ &\geq \frac{\mathbb{E}[T] \cdot \delta}{8} - \frac{k \cdot \delta}{2} \\ &= \frac{\mathbb{E}[T] \cdot R_M \cdot \left(\frac{\sigma}{8k}\right)^{1/m}}{8} - \frac{k \cdot R_M \cdot \left(\frac{\sigma}{8k}\right)^{1/m}}{2}\end{aligned}$$

◀

3.2 k-Taxi

▶ **Lemma 12.** *For σ-smooth instances of the k-taxi problem, the optimal offline algorithm pays an expected cost of at least*

$$\operatorname{cost}(\operatorname{opt}_M) \geq \frac{R_M}{8} \cdot \left(\frac{\sigma}{8k}\right)^{1/m} \cdot \mathbb{E}[T] - c,$$

where $c > 0$ is a constant independent of the request sequence.

Proof. We proceed similarly to the proof of Lemma 11, except in the construction of S_i we add a request $r_{t+i} = (a_{t+i}, b_{t+i})$ if its start location a_{t+i} is more than distance δ from the destinations of *all* the requests $r_{t+1}, \ldots, r_{t+i-1}$:

$$S_i = \begin{cases} S_{i-1} \cup \{r_{t+i}\} & \text{if } d(b_{t+j}, a_{t+i}) > \delta \text{ for all } j = 1, \ldots, i-1 \\ S_{i-1} & \text{otherwise.} \end{cases}$$

Again,

$$P\left(S_i = S_{i-1}\right) \leq \frac{4k}{\sigma}\left(\frac{\delta}{R_M}\right)^m$$

as $S_i = S_{i-1}$ requires a_{t+i} to fall in one of at most $4k$ balls of radius δ. As before, the same choice of δ yields $|S_{4k}| \geq 2k$ with probability at least $1/2$. If this is the case, then any offline algorithm must pay at least $k \cdot \delta$ to serve these requests: At least $|S_{4k}| - k \geq k$ of the requests in S_{4k} are served by a taxi that also served a previous request since time $t+1$, incurring cost at least δ by definition of S_{4k}. The lemma is then concluded in an identical fashion to the proof of Lemma 11. ◀

3.3 Chasing Small Sets

▶ **Lemma 13.** *In σ-smoothed instances of chasing small sets, the optimal offline algorithm pays an expected cost of at least*

$$\operatorname{cost}(\operatorname{opt}_M) \geq \frac{R_M}{2} \cdot \left(\frac{\sigma}{2k^2}\right)^{1/m} \cdot \mathbb{E}[T].$$

Proof. Let $\delta > 0$. Consider two consecutive request sets r_{t-1} and r_t. The probability that the ith point in r_{t-1} is within distance δ from the jth point in r_t is at most $\frac{1}{\sigma}\left(\frac{\delta}{R_M}\right)^m$. Thus, by a union bound over the at most k^2 pairs of points in $r_{t-1} \times r_t$, the probability that some point in r_{t-1} is within distance δ from some point in r_t is at most $\frac{k^2}{\sigma}\left(\frac{\delta}{R_M}\right)^m$. Otherwise, with probability at least $1 - \frac{k^2}{\sigma}\left(\frac{\delta}{R_M}\right)^m$, any offline algorithm must pay at least δ at time t to move from r_{t-1} to r_t. Choosing $\delta = R_M \cdot \left(\frac{\sigma}{2k^2}\right)^{1/m}$, we see that the probability is at least $1/2$. Thus, the expected offline cost is at least $\frac{\delta}{2} = \frac{R_M}{2} \cdot \left(\frac{\sigma}{2k^2}\right)^{1/m}$ per time step. ◀

4 Lower Bound

Complementing our upper bounds, we now prove a polylog(k/σ) lower bound for the three problems, i.e., that no algorithm can achieve a competitive ratio of $O(\log^c(k/\sigma))$ for any $c < 1/2$ (Theorem 5).

Our proof is based on the classical $\Omega(\log k)$ lower bound for k-server and MTS on $(k+1)$-point uniform metrics, in which each request location is drawn uniformly at random from the $k+1$ points [9]. As σ-smooth instances require distributions with infinite support, we intend to simulate this by substituting each of the $k+1$ points by a small neighborhood in normed space. We describe the construction for k-server first, and then explain how to adapt it to k-taxi and chasing small sets.

4.1 k-Server

We first recall the $\Omega(\log k)$ lower bound from [9]:[3] The underlying metric space is a uniform metric of $k+1$ points (i.e., all pairwise distances are 1) and each request is drawn uniformly at random from the $k+1$ points. Thus, an online algorithm must move at each time step with probability $1/(k+1)$, paying $T/(k+1)$ in expectation for a request sequence of length T. In contrast, an offline algorithm can, whenever a request arrives at the (unique) location not covered by a server, take the server from the location whose next request is furthest in the future. Thus, it need not move again until each of the k other locations is requested again at least once. By a standard coupon collector argument, it takes $\frac{k+1}{k} + \frac{k+1}{k-1} + \cdots + \frac{k+1}{1} = \Theta(k \log k)$ steps in expectation for each of the k other locations to be requested again, so the algorithm only needs to move roughly once every $\Theta(k \log k)$ steps. For a sequence of T requests, this leads to an optimal offline cost of $O\left(\frac{T}{k \log k} + 1\right)$ (see [9] for details). The ratio between the online cost $T/(k+1)$ and offline cost $O\left(\frac{T}{k \log k} + 1\right)$ yields the known $\Omega(\log k)$ lower bound for uniform metrics.

To simulate this in the smoothed setting, let $m = \lceil \log_2(k+1) \rceil$ and take as metric space the hypercube $[0,1]^m$ with ℓ_∞ distance, which is a ball in ℓ_∞^m. Let V be a fixed set of some $k+1$ vertices of this hypercube (i.e., with integer coordinates). Let $\epsilon = \frac{1}{2k \log_2 k} \leq \frac{1}{4}$. We say two points are *near* each other if their distance is at most ϵ. Otherwise, they are *far*. Let P be the set of points in $[0,1]^m$ that are near some vertex in V. Consider the instance where each request is drawn uniformly at random from P.

For the online algorithm, each request incurs cost at least $\Omega(1/k)$ in expectation, since with probability at least $\frac{1}{k+1}$ the request arrives near a vertex v whereas all online algorithm servers are near vertices in $V \setminus \{v\}$. For the offline algorithm, if each request arrived at the nearby vertex in V instead of its actual location in P, then the instance would be precisely the aforementioned instance on a $(k+1)$-point uniform metric, with an offline cost of $O\left(\frac{T}{k \log k} + 1\right)$ for T requests. The fact requests arrive *near* vertices instead of *at* vertices only increases the offline cost by at most ϵ per request, as the distance between any two points near the same vertex is at most ϵ. By choice of ϵ, this still leads to an offline cost of $O\left(\frac{T}{k \log k} + 1\right)$, and a competitive ratio lower bound of $\Omega(\log k)$.

It remains to express the lower bound in terms of k/σ. The overall hypercube $[0,1]^m$ has a volume of 1, whereas the set P has a volume of $(k+1)\epsilon^m$. Thus, $\sigma = \frac{1}{(k+1)\epsilon^m} = 2^{-\Theta(\log^2 k)}$, and $k/\sigma = 2^{\Theta(\log^2 k)}$. Hence, the lower bound can be written as $\Omega(\log^{1/2}(k/\sigma))$.

[3] The presentation in [9] is for MTS. We describe it here in the language of k-server.

4.2 k-Taxi

The construction is similar, but requests need to be pairs of two points (a, b). For each request, we sample a as above, and if a is near a vertex v, then we sample b uniformly at random from the points near v as well. This ensures that an offline algorithm can keep its taxis near k distinct vertices. The only difference in calculations is that σ is larger by a factor $k + 1$ as request destinations are sampled from distributions supported on points near a single vertex, but the asymptotic bound $\sigma = 2^{-\Theta(\log^2 k)}$ is unaffected by this.

4.3 Chasing Small Sets

We again use a similar construction. Each request set contains k points in P, sampled uniformly at random subject to the constraint that the points are near k distinct vertices of V. Thus, for each request set there is one vertex in V that is far from it. The online cost is still $\Omega(1/k)$ per request in expectation, as with probability at least $1/(k + 1)$ the request set contains no point near the algorithm's server. The offline cost is also bounded as before, as whenever it is far from all points in the request set, it can first move to the vertex for which it will happen furthest in the future that it is far from all points in the request set; additionally, it pays at most ϵ per request to move to a nearby point in the request set. The lower bound $\Omega(\log^{1/2}(k/\sigma))$ follows in the same way.

References

1. Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. Local max-cut in smoothed polynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, 2017.
2. David Arthur, Bodo Manthey, and Heiko Röglin. Smoothed analysis of the k-means method. *J. ACM*, 58(5), 2011. doi:10.1145/2027216.2027217.
3. David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In Nina Amenta and Otfried Cheong, editors, *Proceedings of the 22nd ACM Symposium on Computational Geometry*, 2006.
4. Nikhil Bansal, Haotian Jiang, Raghu Meka, Sahil Singla, and Makrand Sinha. Prefix discrepancy, smoothed analysis, and combinatorial vector balancing. In *13th Innovations in Theoretical Computer Science Conference, ITCS*, 2022. doi:10.4230/LIPIcs.ITCS.2022.13.
5. Nikhil Bansal, Haotian Jiang, Raghu Meka, Sahil Singla, and Makrand Sinha. Smoothed analysis of the komlós conjecture. In *49th International Colloquium on Automata, Languages, and Programming, ICALP*, 2022. doi:10.4230/LIPIcs.ICALP.2022.14.
6. Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average-case and smoothed competitive analysis of the multilevel feedback algorithm. *Mathematics of Operations Research*, 31(1):85–108, 2006. doi:10.1287/MOOR.1050.0170.
7. Adam Block, Yuval Dagan, Noah Golowich, and Alexander Rakhlin. Smoothed online learning is as easy as statistical learning. In *Conference on Learning Theory, COLT*, 2022.
8. Avrim Blum and Carl Burch. On-line learning and the metrical task system problem. *Mach. Learn.*, 39(1), 2000. doi:10.1023/A:1007621832648.
9. Allan Borodin, Nathan Linial, and Michael E Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM (JACM)*, 39(4):745–763, 1992. doi:10.1145/146585.146588.
10. Sébastien Bubeck, Niv Buchbinder, Christian Coester, and Mark Sellke. Metrical service systems with transformations. In *12th Innovations in Theoretical Computer Science Conference, ITCS*, 2021. doi:10.4230/LIPIcs.ITCS.2021.21.
11. Sébastien Bubeck, Christian Coester, and Yuval Rabani. Shortest paths without a map, but with an entropic regularizer. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, 2022.

12 Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k-server conjecture is false! In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 581–594, 2023. doi:10.1145/3564246.3585132.

13 Sébastien Bubeck, Michael B Cohen, James R Lee, and Yin Tat Lee. Metrical task systems on trees via mirror descent and unfair gluing. *SIAM Journal on Computing*, 50(3):909–923, 2021. doi:10.1137/19M1237879.

14 Sébastien Bubeck, Michael B Cohen, Yin Tat Lee, James R Lee, and Aleksander Madry. K-server via multiscale entropic regularization. In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*, pages 3–16, 2018. doi:10.1145/3188745.3188798.

15 Niv Buchbinder, Anupam Gupta, Marco Molinaro, and Joseph (Seffi) Naor. k-servers with a smile: Online algorithms via projections. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2019.

16 William R. Burley. Traversing layered graphs using the work function algorithm. *J. Algorithms*, 20(3), 1996. doi:10.1006/JAGM.1996.0024.

17 Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In *On-Line Algorithms, Proceedings of a DIMACS Workshop*. DIMACS/AMS, 1991. doi:10.1090/DIMACS/007/02.

18 Christian Coester and Elias Koutsoupias. The online k-taxi problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC*, 2019.

19 Christian Coester and James R. Lee. Pure entropic regularization for metrical task systems. *Theory Comput.*, 18:1–24, 2022. doi:10.4086/TOC.2022.V018A023.

20 Vincent Cohen-Addad and Varun Kanade. Online optimization of smoothed piecewise constant functions. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2017.

21 Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Stochastic k-server: How should uber work? In *44th International Colloquium on Automata, Languages, and Programming, ICALP*, 2017.

22 Naveen Durvasula, Nika Haghtalab, and Manolis Zampetakis. Smoothed analysis of online non-parametric auctions. In *Proceedings of the 24th ACM Conference on Economics and Computation, EC*, 2023.

23 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. *Algorithmica*, 68(1), 2014. doi:10.1007/S00453-013-9801-4.

24 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Smoothed analysis of the 2-opt algorithm for the general TSP. *ACM Trans. Algorithms*, 13(1), 2016. doi:10.1145/2972953.

25 Nova Fandina and Yair Bartal. Lecture notes in Metric embedding theory and its algorithmic applications, 2018. Accessed on June 29, 2024. URL: https://moodle2.cs.huji.ac.il/nu18/pluginfile.php/274639/mod_resource/content/1/METAP19_Lecture_2.pdf.

26 Amos Fiat, Dean P. Foster, Howard J. Karloff, Yuval Rabani, Yiftach Ravid, and Sundar Vishwanathan. Competitive algorithms for layered graph traversal. *SIAM J. Comput.*, 28(2), 1998. doi:10.1137/S0097539795279943.

27 Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, FOCS*, 1990.

28 Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2008.

29 Yiannis Giannakopoulos, Alexander Grosz, and Themistoklis Melissourgos. On the smoothed complexity of combinatorial local search. In *51st International Colloquium on Automata, Languages, and Programming, ICALP*, 2024. doi:10.4230/LIPIcs.ICALP.2024.72.

30 Fabrizio Grandoni, Anupam Gupta, Stefano Leonardi, Pauli Miettinen, Piotr Sankowski, and Mohit Singh. Set covering with our eyes closed. *SIAM J. Comput.*, 42(3):808–830, 2013. doi:10.1137/100802888.

31 Anupam Gupta, Guru Guruganesh, Binghui Peng, and David Wajc. Stochastic online metric matching. In *46th International Colloquium on Automata, Languages, and Programming, ICALP*, 2019. doi:10.4230/LIPIcs.ICALP.2019.67.

32 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Poly-logarithmic competitiveness for the k-taxi problem. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2024.

33 Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM J. Comput.*, 46(3), 2017. doi:10.1137/15M1050276.

34 Nika Haghtalab, Yanjun Han, Abhishek Shetty, and Kunhe Yang. Oracle-efficient online learning for smoothed adversaries. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS*, 2022.

35 Nika Haghtalab, Tim Roughgarden, and Abhishek Shetty. Smoothed analysis of online and differentially private learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS*, 2020.

36 Nika Haghtalab, Tim Roughgarden, and Abhishek Shetty. Smoothed analysis with adaptive adversaries. *J. ACM*, 71(3), 2024. doi:10.1145/3656638.

37 Sampath Kannan, Jamie Morgenstern, Aaron Roth, Bo Waggoner, and Zhiwei Steven Wu. A smoothed analysis of the greedy algorithm for the linear contextual bandit problem. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS*, 2018.

38 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5), 1995. doi:10.1145/210118.210128.

39 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4), 2021. doi:10.1145/3447579.

40 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2), 1990. doi:10.1016/0196-6774(90)90003-W.

41 Bodo Manthey and Rianne Veenstra. Smoothed analysis of the 2-opt heuristic for the TSP: polynomial bounds for gaussian noise. In *Algorithms and Computation - 24th International Symposium, ISAAC*, 2013.

42 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Commun. ACM*, 65(7), 2022. doi:10.1145/3528087.

43 Christos H Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991. doi:10.1016/0304-3975(91)90263-2.

44 Manish Raghavan, Aleksandrs Slivkins, Jennifer Wortman Vaughan, and Zhiwei Steven Wu. The externalities of exploration and how data diversity helps exploitation. In *Conference On Learning Theory, COLT*, 2018.

45 Alexander Rakhlin, Karthik Sridharan, and Ambuj Tewari. Online learning: Stochastic, constrained, and smoothed adversaries. *Advances in neural information processing systems*, 2011.

46 Tim Roughgarden, editor. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020.

47 Guido Schäfer and Naveen Sivadasan. Topology matters: Smoothed competitiveness of metrical task systems. *Theoretical Computer Science*, 341(1-3):216–246, 2005. doi:10.1016/J.TCS.2005.04.006.

48 Daniel A Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)*, 51(3):385–463, 2004. doi:10.1145/990308.990310.

Fast Gaussian Elimination for Low Treewidth Matrices

Martin Fürer
Department of Computer Science and Engineering,
Pennsylvania State University, University Park, PA, USA

Carlos Hoppen
Instituto de Matemática e Estatística, Universidade Federal do Rio Grande do Sul, Brazil

Vilmar Trevisan
Instituto de Matemática e Estatística, Universidade Federal do Rio Grande do Sul, Brazil

Abstract

Let $A = (a_{ij})$ be an $m \times n$ matrix whose elements lie in an arbitrary field $\mathbb{F}$, and let G be the bipartite graph with vertex set $\{v_1, \ldots, v_m\} \cup \{w_1, \ldots, w_n\}$ such that vertices v_i and w_j are adjacent if and only if $a_{ij} \neq 0$. We introduce an algorithm that finds an $m \times n$ matrix U in row echelon form and a permutation matrix Q of order n, such that AQ is row equivalent to U. If a tree decomposition $\mathcal{T}$ of G of width k and size $O(k(m+n))$ is part of the input, then Q and the columns of U that contain a pivot can be computed in time $O(k^2(m+n))$. Among other things, this allows us to compute the rank and the determinant of A in time $O(k^2(m+n))$. It also allows us to decide in time $O(k^2(m+n))$ whether the linear system $Ax = b$ has a solution and to compute a solution of the linear system in case it exists.

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

Keywords and phrases Gaussian elimination, FPT algorithms, treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.116

Funding *Carlos Hoppen*: C. Hoppen acknowledges the support of CNPq 315132/2021-3.
Vilmar Trevisan: V. Trevisan acknowledges partial support of CNPq grants 409746/2016-9 and 303334/2016-9, CAPES under project MATHAMSUD 18-MATH-01 and FAPERGS under Project PqG 17/2551-0001. CNPq is Conselho Nacional de Desenvolvimento Científico e Tecnológico, CAPES is Coordenação de Aperfeiçoamento de Pessoal de Nível Superior and FAPERGS is Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul.

1 Introduction

Reducing a matrix to a desired form or decomposing a matrix into a product of matrices with a given structure are basic operations in linear algebra. Problems involving triangular matrices and matrices in row echelon form are often easy to solve, and a large number of tasks may be accomplished by first reducing a general matrix into a matrix of this form. To be precise, a matrix $A = (a_{ij})$ is in *row echelon form* if all rows consisting entirely of zeros are at the bottom, and the following holds for the remaining rows. If i is a nonzero row, its *pivot* is the element $a_{ij} \neq 0$ with least j. If rows $i_1 < i_2$ have pivots in columns j_1 and j_2, respectively, then $j_1 < j_2$.

A basic result in linear algebra states that every matrix can be transformed into a matrix in row echelon form by a sequence of three types of row operations, known as *elementary row operations*. In this paper, we shall refer to two of these types: a type I elementary row operation on a matrix involves interchanging two rows. A type II elementary row operation involves adding a multiple of a row to another row. We say that A and B are *row equivalent* if A can be transformed into B by a sequence of elementary row operations. The process of


© Martin Fürer, Carlos Hoppen, and Vilmar Trevisan;
licensed under Creative Commons License CC-BY 4.0



reducing a matrix to row echelon form by elementary row operations is known as Gaussian elimination. The standard school algorithm reduces an arbitrary $m \times n$ input matrix A with entries in a field $\mathbb{F}$ to row echelon form in $O(mn \min\{m, n\})$ field operations. However, it can be faster if more is known about the structure of A. For instance, if A is sparse, its structure can be exploited if we find a *pivoting scheme* (also known as *elimination ordering*) that minimizes the *fill-in*, defined as the set of matrix positions that were initially 0, but became nonzero at some point during the computation. It is worth noting that sparse matrices often occur for some structural reason, such as small treewidth. Gaussian elimination cannot be used efficiently on random sparse matrices with any elimination order, as we would expect such matrices to quickly get dense as the algorithm progresses. We should mention that other strategies can be used to exploit the sparsity of the input matrix, see for instance the work of Peng and Vempala [12].

Much of the early focus has been on the question of characterizing the graphs corresponding to matrices with a perfect elimination order, meaning that there is a pivot strategy for Gaussian elimination without any fill-in. Parter [11] has assigned a graph to a matrix A with an edge $\{i, j\}$ if a_{ij} or a_{ji} are non-zero. He has then shown that there is a perfect elimination order if the graph is a tree. As became customary afterwards, he chooses all pivots in the diagonal and makes the strong assumption that all diagonal elements of the matrix are non-zero and remain so during the computation, usually referred to as not having any accidental cancellation.

Obviously, this assumption simplifies Gaussian elimination enormously. It is justified because it is true for symmetric positive definite matrices, which are the most important special class of matrices for systems of linear equations. However, one would like to see more widely applicable methods.

Rose [16] shows that Parter's result can be extended to all triangulated graphs (also known as chordal graphs). Rose, Tarjan, and Lueker [18] show that a graph has a perfect elimination order if and only if the graph is triangulated. In this case, they find a perfect elimination order in $O(|V| + |E|)$ time. Rose and Tarjan [17] efficiently find a perfect elimination order in a directed graph if one exists, and show that finding a minimal elimination ordering (minimizing the fill-in) is NP-hard. A directed graph describes the non-zero off-diagonal entries in an $n \times n$ matrix, where Gaussian elimination is done by choosing pivots in the diagonal under the assumption that there will never be a 0 in the diagonal.

The perfect elimination question is more complicated for the case of asymmetric $m \times n$ matrices A, where any nonzero pivot is allowed. Any $m \times n$ matrix $A = (a_{ij})$ may be naturally associated with a bipartite graph G with vertex set $\{v_1, \ldots, v_m\} \cup \{w_1, \ldots, w_n\}$ such that vertices v_i and w_j are adjacent if and only if $a_{ij} \neq 0$. We say that G is the *underlying bipartite graph* of A. This allows us to employ structural decompositions of graph theory to deal with the nonzero entries of A in an efficient way. Different from the symmetric case with pivots restricted to the diagonal, there is no nice characterization known for the bipartite graphs with perfect elimination order when arbitrary pivots are allowed. Nevertheless, a very large class is given by the *chordal bipartite* graphs defined by Golumbic and Goss [7]. These graphs are not chordal, as obviously only forests are bipartite and chordal. Still, chordal bipartite graphs are characterized similarly to chordal graphs. For chordal graphs, the defining property is that every cycle of length at least 4 has a chord, while chordal bipartite graphs are defined by the requirement that every cycle of length at least 6 has a chord. Golumbic and Goss [7] have shown that chordal bipartite graphs have a perfect elimination order.

They are not the only bipartite graphs with a perfect elimination order, but the only ones where one can always pick any simplicial edge (definition follows) as a pivot [7]. An edge $\{u, v\}$ is simplicial if the union of the neighborhoods of u and v induces a complete bipartite graph. It is easy to see that a bipartite graph G has a perfect elimination order if there exists an ordering of a matching $\{(u_1, v_1), (u_2, v_2), \ldots, (u_p, v_p)\}$ in G such that (u_i, v_i) is simplicial in the subgraph $G[V \setminus \{u_1, v_1, u_2, v_2, u_{i-1}, v_{i-1}\}]$ for $i \in \{1, \ldots, p\}$.

Radhakrishnan, Hunt, and Stearns [13] do Gaussian elimination for symmetric matrices with a given tree decomposition of width k in time $O(k^2 n)$. As has been common, they implicitly assume that the diagonal is non-zero and there is no accidental cancellation. Naturally, given today's familiarity with algorithms based on tree decompositions, it would now be an easy exercise to design such an algorithm when pivots can always be selected in the diagonal.

Only in recent years has there finally been some progress towards the difficult problem of efficient Gaussian elimination for matrices of small treewidth without the simplifying assumption that pivots can always be chosen in the diagonal. Fomin et al. [5] have shown a Gaussian elimination algorithm running in time $O(p^2 n)$ where p is the lesser known tree-partition width parameter or the pathwidth. The tree-partition width of a graph can be arbitrarily large already for graphs of constant treewidth, unless the maximum degree is bounded [19]. However, Fomin et al. [5] still obtained an $O(k^3 n)$ algorithm for treewidth k. We should also refer to the work of Dong, Lee, and Ye [4], whose main result implies an $O(k^2 n)$ algorithm for the solution of a linear system $Ax = b$ such that the matrix $A \in \mathbb{R}^{m \times n}$ is full-rank (actually, their result is about the solution of a linear program).

Naturally, the longstanding main goal is an $O(k^2 n)$ Gaussian elimination algorithm for treewidth k without restrictive assumptions. This would provide a smooth transition to the school algorithm when k approaches n. It would also achieve the same running time bound for the general case as for the easy case of non-zero diagonal and no accidental cancellation. It actually seems that the $O(k^2 n)$ goal has not been explicitly formulated until recently [5], maybe because it had not seemed achievable. However, when this goal has been obtained by simple algorithms under very strong assumptions, the question whether these assumptions (that convenient entries are nonzero and thus can be used as pivots) are necessary must have been obvious.

We make substantial progress in this direction by devising the algorithm `Fast Gaussian Elimination`. Given an $m \times n$ matrix A over a field $\mathbb{F}$ with underlying bipartite graph G, and given a tree decomposition $\mathcal{T}$ of G of width k and size $O(k(m + n))$, it produces a matrix U in row echelon form that is row equivalent to A. The entries of the columns of U that contain a pivot (and therefore generate a space of dimension $\operatorname{rank}(A)$) are fully computed in time $O(k^2 n)$. The entries of the remaining columns of U are described implicitly as linear combinations of at most k other columns, and may be computed explicitly in time $O(km(n - \operatorname{rank}(A)))$. In particular, this algorithm allows us to compute the rank and the determinant of A in time $O(k^2(m + n))$. It also allows us to decide in time $O(k^2(m + n))$ whether the linear system $Ax = b$ has a solution, and to compute a solution of the linear system in case it exists. Our main result is stated more formally below.

▶ **Theorem 1.** *Let A be an $m \times n$ matrix over a field $\mathbb{F}$, let G be its underlying bipartite graph, and let $\mathcal{T}$ be a tree decomposition of G of width k and size $O(k(m + n))$. Algorithm `Fast Gaussian Elimination` produces an $m \times n$ matrix U in row echelon form, and a permutation matrix Q of order n with the property that AQ is row equivalent to U. The permutation matrix Q, and the columns of U that contain a pivot may be computed with $O(k^2(m + n))$ field operations. The full matrix U may be computed with $O(km(n - \operatorname{rank}(A)))$ additional field operations.*

We actually also compute a permutation matrix P of order m such that U is the row echelon form that would be obtained from PAQ by the most standard algorithm (adding multiples of earlier rows to later rows).

Our algorithm is not based on the $O(k^3n)$ algorithm of Fomin et al. [5], as it uses standard tree decompositions in a natural way, and does not rely on the somewhat obscure notion of tree-partition width. Even though our algorithm removes the decades old severe restriction to pivoting in the main diagonal, the main idea of our Gaussian elimination algorithm is quite simple. The tree decomposition provides an elimination order for rows (equations) and columns (variables). This order sequentially hands out permissions to rows and columns to be eliminated. However, when a row i receives a green light to be processed, it is only processed immediately if it contains a variable that has already received a green light. Otherwise, this row is put into a buffer to be delayed until one of its variables x_j also has a green light. Then a_{ij} is chosen as a pivot. Likewise, when a variable receives a green light then, if needed, its column is put into a buffer until some row containing this variable has a green light.

As was the case in Fomin et al. [5] we use a *nice tree decomposition* of the underlying bipartite graph G as defined by Kloks [9]. However, to deal with 0's at intended pivot locations, we create temporary buffers for rows and columns that allow us to delay the selection of pivots as discussed above. As is typical for algorithms based on tree decompositions, row operations are actually performed on small submatrices of the input matrices. For simplicity, the algorithm does not manipulate the large given matrix. Instead, these small submatrices, called boxes, are maintained separately. The small submatrices in this paper include additional information to keep track of the rows and columns that lie in the temporary buffer, which allows us to make headway towards the output matrix in row echelon form even when accidental cancellations occur. It is crucial to perform so-called bookkeeping operations that ensure that the temporary buffers remain small during the entire application of the buffer.

The paper is organized as follows. The algorithm is described in Section 2, where we also discuss its correctness. The running time is analyzed in Section 3. We should mention that, to achieve this running time, we need to keep track of the field operations in a global way, rather than adding the contributions of the worst-case scenarios at each step. Together, these two sections establish Theorem 1. We provide several consequences of our result in Section 4.

2 The Algorithm

We now describe the algorithm `Fast Gaussian Elimination`. We start with the definition of a tree decomposition, which we now state. Let $G = (V, E)$ be a ν-vertex graph, with the standard assumption that $V = [\nu] = \{1, \ldots, \nu\}$. A *tree decomposition* of a graph G is a tree $\mathcal{T}$ with *nodes* $\{1, \ldots, r\}$, where each node t is associated with a *bag* $B_t \subseteq V$, satisfying the following properties: (1) $\bigcup_{t=1}^{r} B_t = V$; (2) For every edge $\{v, w\} \in E$, there exists B_t containing v and w; (3) For any $v \in V$, the subgraph of $\mathcal{T}$ induced by the nodes whose bags contain v is connected. The *width* of the tree decomposition $\mathcal{T}$ is defined as $\max_t (|B_t| - 1)$ and the *treewidth* $\mathrm{tw}(G)$ of graph G is the smallest k such that G has a tree decomposition of width k. Tree decompositions were popularized by the seminal work of Robertson and Seymour [14, 15], but other definitions that are similar or even equivalent have appeared in earlier work, see [1, 8]. As mentioned in the introduction, we use the concept of *nice tree decomposition* introduced by Kloks [9], which is a rooted tree decomposition $\mathcal{T}$ of a graph G such that all nodes are of one of the following types:

(a) **(Leaf)** The node t is a leaf of $\mathcal{T}$;
(b) **(Introduce)** The node t introduces vertex v, that is, it has a single child s, $v \notin B_s$ and $B_t = B_s \cup \{v\}$.
(c) **(Forget)** The node t forgets vertex v, that is, t has a single child s, $v \notin B_t$ and $B_s = B_t \cup \{v\}$;
(d) **(Join)** The node t is a join, that is, it has two children s and s', and $B_t = B_s = B_{s'}$.

We further assume that the bag associated with the root is empty. This is easy to accomplish, as any tree decomposition $\mathcal{T}$ that satisfies (a)-(d), but does not satisfy this additional requirement, may be turned into a nice tree decomposition with empty root bag by appending a path of length at most $k+1$ to the root of $\mathcal{T}$, where all nodes have type forget. Having a root with an empty bag ensures that each vertex of $G = (V, E)$ is associated with exactly one forget node that forgets it. Moreover, it ensures that, for any $v \in V$, the node of $\mathcal{T}$ that is closest to the root among all t such that $v \in B_t$ is the child of the node that forgets v. Despite the additional structure, a nice tree decomposition may be efficiently derived from an arbitrary tree decomposition. More precisely, Kloks [9, Lemma 13.1.2] has shown that, if G is a graph of order ν and we are given an arbitrary tree decomposition of G with width k and r nodes, it is possible to turn it into a nice tree decomposition of G with at most 4ν nodes and width at most k in time $O(k(\nu + r))$. Regarding the computation of a tree decomposition, Bodlaender's theorem [2] provides an optimal tree decomposition in linear time when the treewidth k is a constant. Bodlaender et al. [3] and Korhonen [10] compute faster constant factor approximations in linear time. Thus, for bounded k, there is no need to assume a tree decomposition is given, since both the tree decomposition and the Gaussian elimination are then computed in time $O(n)$. However, the constants hidden in the O-notation grow quickly with k.

We are now ready to describe the algorithm. Let A be an $m \times n$ matrix with entries in a field $\mathbb{F}$ and let $G = (V, E)$ be the underlying bipartite graph with vertex set $V = \{v_1, \ldots, v_m\} \cup \{w_1, \ldots, w_n\}$ associated with it. We wish to find an $m \times m$ permutation matrix P, an $n \times n$ permutation matrix Q and an $m \times n$ matrix U in row echelon form such that PAQ is row equivalent to U in a direct way. We shall operate on a nice tree decomposition $\mathcal{T}$ of G with node set $[r]$ and width k rather than on the matrix A itself. The algorithm `Fast Gaussian Elimination` works bottom-up on the rooted tree $\mathcal{T}$, that is, it only processes a node t after its children have been processed. Each node t except the root produces a data structure known as a *box* and transmits it to its parent. A box is a matrix A_t with $O(k)$ rows and columns. It may be recorded as a triple of matrices (M_t, N_t, J_t) whose rows and columns are labeled by distinct elements of $\{v_1, \ldots, v_m\}$ and $\{w_1, \ldots, w_n\}$, respectively, associating rows and columns of the boxes with rows and columns of the original matrix. We may view it as

$$A_t = \begin{array}{|c|c|} \hline \mathbf{0}_{m_t \times n_t} & M_t \\ \hline N_t & J_t \\ \hline \end{array}, \tag{1}$$

where J_t is an $r_t \times c_t$ matrix whose rows and columns are labeled by the elements in B_t, the bag associated with node t. Here r_t is the number of row vertices (i.e., vertices v_i) in B_t and c_t is the number of column vertices (i.e., vertices w_j) in B_t. The matrices M_t and N_t have dimensions $m_t \times c_t$ and $r_t \times n_t$, respectively, where $m_t < 2c_t$ and $n_t < 2r_t$. Observe

that some of these matrices can be degenerated, in the sense that $m_t = 0$, $n_t = 0$, $c_t = 0$ or $r_t = 0$. In a case where $m_t = 0$, $n_t = 2$, $c_t = 2$ and $r_t = 3$, for instance, the box A_t is 3×4 matrix, M_t is a matrix with no rows and two empty columns, and both N_t and J_t are 3×2 matrices. We should note that a similar data structure, also called a box, has been exploited in the context of computing a diagonal matrix that is congruent to a given symmetric matrix using a tree decomposition of its underlying graph [6].

It will also be useful to assign a *canonical ordering* to the rows and columns of the input matrix. We assume that they are ranked according to the order in which their Forget nodes appear in some topological ordering (e.g., obtained by a post-order traversal of $\mathcal{T}$), that is, v_1 is the first vertex that is forgotten among all vertices that represent rows, v_2 is the second, and so on. The same applies to $w_1, \ldots, w_n$.

We start with an intuition about the meaning of the boxes and about how the algorithm works. Recall that the nodes of the nice tree decomposition $\mathcal{T}$ are ordered bottom-up as $1 \ldots, r$, where r is the root and $B_r = \emptyset$. At each node $t \in V(\mathcal{T})$, the algorithm initializes box A_t (if t is a leaf) or produces a new box A_t based on its bag B_t and on the boxes transmitted by its children. While producing the box A_t, the algorithm may access entries a_{ij} of the input matrix such that both i and j are associated with vertices in B_t. The box A_t is then transmitted to its parent, except if t is the root of $\mathcal{T}$. We call this *processing* node t.

It is also useful to keep in mind that the algorithm performs two types of elementary row operations, which we call *authentic operations* and *bookkeeping operations*. The former are operations that can be performed in the full matrix A in order to reach row echelon form. The latter are operations that are only performed on the boxes as a way to limit their sizes, but which do not have full matrix counterparts. The effect of processing nodes may be viewed as a sequence of $m \times n$ matrices

$$\tilde{A}_0 = A, \tilde{A}_1, \ldots, \tilde{A}_r = U, \tag{2}$$

where U is in row echelon form and $\tilde{A}_t$ is obtained from $\tilde{A}_{t-1}$ by performing the authentic elementary row operations that have been performed while node t is processed. We should mention that this sequence of row equivalent matrices is a simplification of what actually happens, but that it could be achieved if an oracle gave us the proper ordering of rows and columns of A. In reality, the proper orderings are also computed by the algorithm while the nodes are processed, leading to the permutation matrices P and Q mentioned in the statement of Theorem 1.

To achieve the complexity stated in Theorem 1, row operations cannot be actually performed on $m \times n$ matrices, but are instead performed only in the (smaller) boxes. It is useful to think that any row or column of the input matrix A, say row i, is initially *untouched* and changes to a *regular* row when the vertex v_i associated with it appears in the bag of a node of the branch[1] of the tree decomposition rooted at t. When v_i is removed from the bag (at the node that forgets it in the nice tree decomposition), either row i is assigned a pivot in some column j, in which case row i is classified as *processed* (column j is also classified as processed), or row i becomes a *buffer* row, meaning that it has been given the green light for a pivot, but that the decision of assigning a pivot has been deferred to a later step because no column has received a green light. In the process of keeping the size of the box bounded, the algorithm may decide that it is impossible or unnecessary to assign a pivot to a buffer row or to a buffer column, in which case the status of the row or column also

[1] We use "branch" to refer to the subtree induced by all descendants of a node.

changes to *processed*. We observe that the classification of each row and column is defined independently for disjoint branches of the tree decomposition, and an important feature of the algorithm is that these independent classifications can be made compatible when two branches are merged (which happens at join nodes).

This finally gets us to the meaning of a box $A_t = (M_t, N_t, J_t)$ at node t. The rows and columns of the matrix J_t are precisely the regular rows and columns of the branch $\mathcal{T}_t$ of $\mathcal{T}$ rooted at node t. In other words, these are the rows and columns corresponding to the vertices in the bag B_t. Moreover, each entry ij of J_t records the *net change* to the original entry a_{ij} due to row operations performed while processing nodes in $\mathcal{T}_t$. The matrix M_t records the entries ij of the matrix $\tilde{A}_t$ defined in (2) in the case where i is a buffer row (with respect to $\mathcal{T}_t$) and j is a regular column. Analogously, N_t records the entries ij of $\tilde{A}_t$ in the case where i is a regular row and j is a buffer column. This reveals another important feature of the algorithm: The entries of the full $m \times n$ matrix associated with regular rows and regular columns can easily be modified in parallel by computations in disjoint branches, as the algorithm does not capture their exact values while processing a single branch. The boxes just store the net changes to the values due to operations in this branch. However, this does not happen for buffer rows and buffer columns, for which exact values are recorded in the box. The top left corner of A_t in (1) shows that, if i is a buffer row and j is a buffer column, the entry ij in $\tilde{A}_t$ is equal to zero. This is consistent with the idea that the algorithm selects a new pivot whenever there are a buffer row i and a buffer column j for which the element ij is nonzero, as both the row and the column had already been given the green light as possible rows and columns for a pivot. So, if row i and column j are in the buffer at the end of step t, then the entry ij must be 0.

While producing the boxes, the algorithm may select pivots for the output matrix U, it may identify zero rows that cannot contain a pivot and it may identify columns that are linear combinations[2] of other columns, and therefore cannot contain pivots together with the other columns. The status of these rows and columns changes to processed, which means that information about them is recorded by global variables, while they are removed from the box.

After this description, we come back to a matrix $\tilde{A}_t$ in (2), depicted in Figure 1[3]. The top left entries lie in processed rows and processed columns that contain pivots. These entries are not modified later in the algorithm. The bottom rows are (processed) zero rows that have been produced while limiting the size of the set of buffer rows (it is indeed impossible to find a pivot in such a row), while the rightmost columns are (processed) columns that have been identified as linear combinations of other columns (except for possible entries in rows whose pivot had already been computed when this column was processed). As a consequence, it is unnecessary to assign a pivot to them. Moreover, while the actual values in the rightmost columns may be modified in later steps of the algorithm, we need not keep track of them, as each such column may be computed as a linear combination of (a known set of) $O(k)$ columns with smaller index at the end of the algorithm. The central submatrix represents entries in the buffer, and in regular or untouched rows and columns. Some of them lie in the box where computations are actually performed in a given step.

Before giving a formal description of every step of the algorithm, we briefly describe what is done at a node of each type. A node of type leaf merely initializes a box with regular rows or columns, while a node of type introduce simply adds a new regular row or column to the

[2] Actually, they are not necessarily linear combinations of the other columns with respect to the entire matrix, but they are linear combinations with respect to a submatrix as depicted in Figure 1.

[3] The authors thank Elizandro Max Borba for the figure.

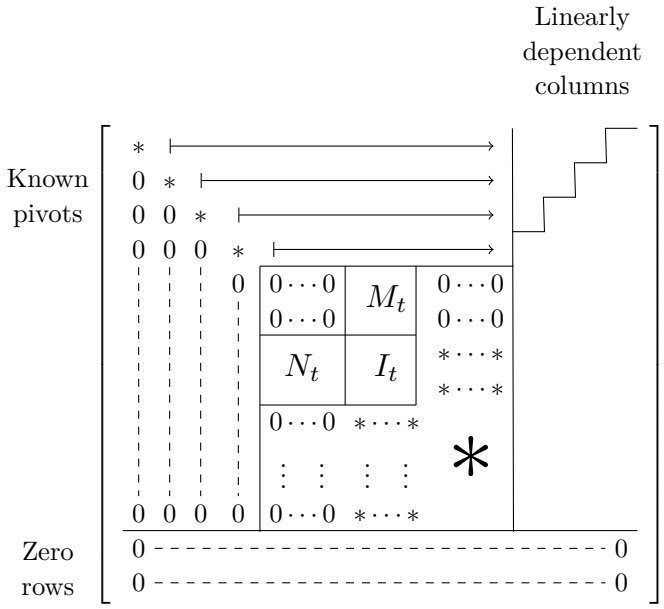


Figure 1 A description of an intermediate matrix in the Gaussian elimination process.

box transmitted by its child. Nodes of type join combine the information that is passed on by two different branches of the decomposition, which have been processed independently of each other. Let A_j and A_ℓ denote the boxes produces by the children of t. A new matrix M_t is produced by stacking the matrix M_j on top of M_ℓ, and a new matrix N_t is produced by juxtaposing N_j to the left of N_ℓ. Such matrices M_t and N_t may become too large, so that row operations are performed to remove some rows and/or columns from the buffer. Producing J_t also requires merging J_j and J_ℓ , as the regular rows and columns are the same. Being able to easily merge the matrices transmitted by the children is the reason why the rows and columns are considered with the canonical ordering mentioned above.

Nodes of type forget concentrate most of the action. Recall that each row i of A is associated with a vertex v_i of G, which in turn is associated with the unique Forget node of $\mathcal{T}$ that forgets v_i. Analogously, each column j of A is associated with a vertex w_j of G and the unique Forget node of $\mathcal{T}$ that forgets w_j. Assume that the algorithm is processing a Forget node t, which means that it received a box A_s of its child s. Further assume that node t forgets a vertex v_i corresponding to row i. This means that the algorithm singles out this row as a candidate for having a pivot. In the box A_s, it checks whether there is a column j that has already been singled out for a possible pivot, but which has no pivot so far. These are precisely the columns of N_s. If such a column exists, it checks whether the entry ij in N_s nonzero. If this is also satisfied, the algorithm performs operations to create a pivot in row i and column j, and they are both removed from the box. If one of the conditions fails, row i is removed from J_s and added to M_t, which contains rows singled out for a pivot, but which have no pivot so far. Furthermore, if the condition on the maximum number of rows of M_t fails after the addition of this new row, the algorithm acts to remove some rows from the buffer.

Analogously, if node t forgets vertex w_j corresponding to column j, it first checks whether there are rows that have been singled out for pivots, but have not been used, namely the rows of M_s. It then looks for one such row i for which the entry ij is nonzero. If such a row

exists, the algorithm performs operations to create a pivot in row i and column j, and they are both removed from the box. If one of the conditions fails, column j is removed from J_s and added to N_t, which contains columns singled out for a pivot, but which have no pivot so far. If the condition on the number of columns of N_t fails after the addition of this new column, the algorithm acts to remove some columns from the buffer.

A high level description of the algorithm may be found in Figure 2. Note that, in this descriptiom, we have five global variables: the number ρ of pivots already computed, two vectors $\mathbf{u}$ and $\mathbf{p}$ of length m and two vectors $\mathbf{w}$ and $\mathbf{q}$ of length n. Initially, these are zero vectors. At the end of the algorithm, vector $\mathbf{u}$ allows us reconstruct the rows of the output matrix U, while vectors $\mathbf{p}$ and $\mathbf{q}$ encode the permutations that generate P and Q, respectively. As we discussed, the algorithm may also find columns of A that are linear combinations of other columns and remove them from boxes. When this happens, the linear combination that produces it is recorded in the vector $\mathbf{w}$, so that the values of entries of U in these columns can also be recovered at the end of the algorithm.

```
Fast Gaussian Elimination(A)
input: a nice tree decomposition 𝒯 of width k of the underlying
graph G associated with the m × n matrix A, the nonzero entries of A
output: matrices U, P and Q
Initialize zero vectors p and u of length m
Initialize zero vectors q and w of length n
Initialize the counter ρ = 0 for the number of pivots
Order the nodes of 𝒯 as 1, 2, ..., r in post-order
for t from 1 to |𝒯| do
    if is-Leaf(t) then (M_t, N_t, J_t)=LeafBox(B_t)
    if is-Introduce(t) then (M_t, N_t, J_t)=IntroBox(B_t, v, A_s)
    if is-Join(t) then (M_t, N_t, J_t)=JoinBox(B_t, A_s, A_s')
    if is-Forget(t) then (M_t, N_t, J_t)=ForgetBox(B_t, v, A_s, A[B_t])
```

Figure 2 High level description of the algorithm `Fast Gaussian Elimination`. The entries of matrix U and the permutations leading to P and Q are constructed stepwise during the `ForgetBox` and `JoinBox` operations.

In the remainder of this section, we shall describe each operation in detail. To simplify our discussion, for a node t, we say that buffer rows and buffer columns have type 1, while regular rows and regular columns have type 2.

Leaf Box

When a node is a leaf corresponding to a bag B_t, then we apply procedure `Leaf Box`. This procedure simply initializes a box A_t where $m_t = 0$, $n_t = 0$, and J_t is a zero matrix of dimension $r_t \times c_t$ whose rows and columns are labeled by the elements of B_t, which contains r_t row vertices and c_t column vertices. No elementary row operations are performed.

Introduce Box

When a node t introduces a vertex u, its bag satisfies $B_t = B_s \cup \{u\}$, where B_s (which does not contain u) is the bag of its child s. The input of `Introduce Box` is the vertex u that has been introduced and the box A_s transmitted by its child. The box A_t is obtained by the

insertion of a new type 2 row or column labeled u whose elements are all zero[4], taking care to insert it in the correct order (with respect to the canonical order of rows and columns of J_t. More precisely, if u is a row vertex v_i, this means that N_t and J_t are obtained from N_s and J_s, respectively, by the addition of a zero row, while $M_t = M_s$. It is clear that $n_t = n_s < 2r_s < 2(r_s + 1) = 2r_t$. Similarly, if v is a column vertex w_j, the addition of this column means that M_t and J_t are obtained from M_s and J_s, respectively, by the addition of a zero column, while $N_t = N_s$. It is clear that $m_t = m_s < 2c_s < 2(c_s + 1) = 2c_t$. No elementary row operations are performed.

Join Box

Let t be a node of type Join and let A_s and $A_{s'}$ be the boxes transmitted by its children, where $s < s' < t$. By the definition of box, A_s and $A_{s'}$ have the same rows and columns of type 2, that is, J_s and $J_{s'}$ have the same dimension and have rows and columns labeled by the same elements (in the same order). Moreover, because vertices of type 1 have been forgotten in their respective branches, the labels of the rows of M_s and $M_{s'}$ are all different, as are the labels of the columns of N_s and $N_{s'}$. The `JoinBox` operation first creates a matrix A_t^* whose rows and columns are labeled by the union of the labels of A_s and $A_{s'}$ with the structure below.

$$A_t^* = \begin{array}{|c|c|c|} \hline \mathbf{0}_{m_s \times n_s} & \mathbf{0}_{m_s \times n_{s'}} & M_s \\ \hline \mathbf{0}_{m_{s'} \times n_s} & \mathbf{0}_{m_{s'} \times n_{s'}} & M_{s'} \\ \hline N_s & N_{s'} & J_t^* \\ \hline \end{array}. \tag{3}$$

Here $J_t^* = J_s + J_{s'}$ has order $r_t \times c_t$ and records the net change of the entries of the original matrix because of row operations in the two branches that are being merged at this step. Let M_t^* be the $(m_s + m_{s'}) \times c_t$ matrix obtained by stacking M_s on top of $M_{s'}$[5]. Let N_t^* be the $r_t \times (n_s + n_{s'})$ matrix obtained by placing N_s to the left of $N_{s'}$.

If $m_s + m_{s'} < 2c_t$, define $M_t = M_t^*$. Otherwise $m_s + m_{s'} \geq 2c_t$. Apply Gaussian elimination to M_t^* until we get a matrix $\tilde{M}_t$ in row echelon form. Here, we can use a standard algorithm for dense matrices that takes $O(c_t^3)$ operations, as M_t^* has c_t columns and at most $4c_t - 2$ rows. Since $m_s + m_{s'} \geq 2c_t$, this process ends with at least c_t zero rows. The matrix M_t is obtained from $\tilde{M}_t$ by removing all the zero rows, and by reordering the rows so that they remain in the original ordering of M_t^*. Assume that the zero rows are labeled by the vertices $v_{i_1}, \ldots, v_{i_\ell}$. The algorithm updates the permutation vector $\mathbf{p}$ by assigning the values $i_1, \ldots, i_\ell$ to the last ℓ components of $\mathbf{p}$ with value 0, which means that the ℓ zero rows of M_t^* become the top zero rows in the description of Figure 1. Given that the new box is defined based on the matrix that has been modified, the type II row operations used for Gaussian elimination are authentic row operations. Given that the order of the rows has been preserved, type I row operations are bookkeeping operations (we observe that the labels of the rows used for authentic row operations will always refer to the original input matrix to account for the fact that row interchanging is not actually performed)[6].

Similarly, if $n_s + n_{s'} < 2r_t$, define $N_t = N_t^*$. Otherwise, make an auxiliary copy of N_t^* and apply Gaussian elimination on it until we get a matrix $\tilde{N}_t$ in row echelon form. As above, we can use a standard algorithm for dense matrices that takes $O(r_t^3)$ operations. There is

[4] The added row or column may actually be empty if a new row vertex is introduced, but $c_s = 0$ and $n_s = 0$, or a new column vertex is introduced, but $r_s = 0$ and $m_s = 0$.

[5] Alternatively, to preserve the canonical ordering of rows, we could merge the two matrices M_s and M'_s in a way that preserves the canonical ordering. However, this does not affect the final result.

[6] The algorithm could be easily adapted to a version where row interchanges are also authentic row operations, but this makes the analysis slightly more involved.

a set S_t of at most r_t columns that form a basis of the column space of N_t^* (precisely the columns of N_t^* with pivots in $\tilde{N}_t$). The matrix N_t is obtained from N_t^* by removing the columns that are not in S_t, form which there are at least $n_s + n_{s'} - r_t \geq r_t$.

▶ Remark 2. We emphasize that N_t is defined in terms of N_t^* rather than $\tilde{N}_t$ in this case, which means that the row operations performed to obtain $\tilde{N}_t$ are *not* performed in the full matrix while producing the sequence (2), but are instead auxiliary operations to find a basis of the column space of N_t^* and to write the remaining columns as linear combinations of the columns in this basis. This is crucial. If we consider the effect of the row operations in the full matrix, while the nonzero entries of the rows corresponding to M_t in the full matrix lie entirely within M_t (except perhaps for entries in columns that are linear combinations of other columns), there may be nonzero entries in the rows corresponding to N_t that lie outside the box A_t. Indeed, the rows of N_t are regular rows, so that they may even be processed in parallel by different branches of the tree decomposition. Adding multiples of them to other rows might affect entries that are not being monitored at this step. This is why the operations performed to turn a copy of N_t^* to row echelon form must be bookkeeping operations.

Coming back to the description of the algorithm, assume that $w_{j_1}, \ldots, w_{j_\ell}$ are the labels of the columns of N_t^* with no pivot in $\tilde{N}_t$, so that they are linear combinations of the remaining columns of $\tilde{N}_t$. The algorithm updates the permutation vector $\mathbf{q}$ by assigning the values $j_1, \ldots, j_\ell$ to the last ℓ components of $\mathbf{q}$ with value 0, which may be seen as moving these columns to the right (see Figure 1). Moreover, the corresponding entries of the vector $\mathbf{w}$ record information about the linear combinations. For each column w_{j_i}, this information consists of the indices of the columns that produce the linear combination, the coefficients of this linear combination, and the number of pivots ρ at this step of the algorithm (as the linear combinations are with respect to the part of the column that does not include rows for which pivots have been assigned.)

Forget Box

Assume that t forgets vertex v and let s be its child, so that $B_t = B_s \setminus \{v\}$. This procedure uses A_s to produce a new box A_t where the row or column associated with v gets a pivot (and is therefore removed from the box) or changes to type 1.

Assume first that $v = v_i$ for a row i. Consider the matrix A_t^* given by[7]

$$A_t^* = \begin{array}{|c|c|} \hline \mathbf{0}_{m_s \times n_s} & M_t^* \\ \hline N_t^* & J_t^* \\ \hline \mathbf{x}_i & \mathbf{y}_i \\ \hline \end{array}, \tag{4}$$

where $M_t^* = M_s$, $\mathbf{x}_i$ is row v_i of N_s, N_t^* is obtained from N_s by removing row $\mathbf{x}_i$, J_t^* is obtained from J_s by removing row v_i, and $\mathbf{y}_i$ is obtained from row v_i of J_s by adding the entry a_{ij} of the input matrix to the entry corresponding to each column w_j.

[7] Representing the row associated with v_i as the last row is helpful for visualizing the operations, but this need not be part of an implementation.

If $\mathbf{x}_i$ is empty or zero, the algorithm sets $J_t = J_t^*$, and adds $\mathbf{y}_i$ to M_t^* as the new bottom row to produce M_t^{**}. If M_t^{**} has at most $2c_t - 1$ rows, the algorithm calls this matrix M_t and concludes the step. Intuitively, this means that row v_i is a new buffer row. Otherwise M_t^{**} has $2c_t$ rows. The algorithm turns this matrix into a matrix $\tilde{M}_t$ in row echelon form and removes $\ell \geq c_t$ zero rows of $\tilde{M}_t$ to produce M_t, as was done in `Join Box`. As a byproduct of these authentic row operations, the algorithm updates $\mathbf{p}$. Similarly, if $n_s < 2r_t$, we set $N_t = N_t^*$. However, because $r_t = r_s - 1$ and the number of columns of N_t^* is n_s, it may be that $n_s \geq 2r_t$. If this happens, as was the case for `Join Box`, the algorithm performs bookkeeping operations to define N_t, and update $\mathbf{q}$ and $\mathbf{w}$.

If $\mathbf{x}_i$ is nonempty and nonzero, the algorithm selects the first nonzero entry of $\mathbf{x}_i$, suppose that it is in column w_j. It then performs row operations in A_t^* in order to eliminate all the other nonzero elements in column w_j (note that these elements are necessarily in rows of type 2, so that only rows in N_t^* and J_t^* are modified). The algorithm defines $\mathbf{p}(\rho) = i$ and $\mathbf{q}(\rho) = j$, while $\mathbf{u}(\rho)$ records information about the nonzero entries of the vector $[\mathbf{x}_i\ \mathbf{y}_i]$, namely the labels j of the nonzero columns of the vector $[\mathbf{x}_i\ \mathbf{y}_i]$ and the actual values of the nonzero entries. The algorithm then adds one to the value of ρ. Matrix J_t is assigned this modified J_t^*, matrix M_t is defined as M_t^*. Moreover, column j is removed from N_t^* to produce N_t^{**}. The row operations performed up to this point are authentic. If the number of columns in N_t^{**} is less than $2r_t$ (note that $r_t = r_s - 1$ and that N_t^{**} has $n_s - 1$ columns), N_t is set to be N_t^{**}. Otherwise, $n_t = 2r_t$ and the algorithm performs bookkeeping operations as in `Join Box` to define N_t, and update $\mathbf{q}$ and $\mathbf{w}$.

Next assume that $v = w_j$ for a column j. Consider the matrix A_t^* given by

$$A_t^* = \left[\begin{array}{ccc} \mathbf{0}_{m_s \times n_s} & M_t^* & \mathbf{c}_j \\ \hline N_t^* & J_t^* & \mathbf{d}_j \end{array}\right], \tag{5}$$

where $N_t^* = N_s$, M_t^* is obtained from M_s by removing column w_j, $\mathbf{c}_j$ is column w_j of M_s, J_t^* is obtained from J_s by removing column w_j and $\mathbf{d}_j$ is obtained from column w_j of J_s by adding the entry a_{ij} (of the input matrix) to the entry corresponding to each row v_i.

If $\mathbf{c}_j$ is empty or zero, the algorithm sets $J_t = J_t^*$ and adds $\mathbf{d}_j$ as a new column of to N_t^* to produce N_t^{**}. If N_t^{**} has fewer than $2r_t$ columns, the algorithm calls this matrix N_t and concludes the step. Otherwise N_t^{**} has $2r_t$ columns and the algorithm defines N_t with bookkeeping operations as was done in `Join Box` (this leads to changes in $\mathbf{q}$ and $\mathbf{w}$). Similarly, if $m_s < 2c_t$, we set $M_t = M_t^*$. However, because $c_t = c_s - 1$ and the number of rows of M_t^* is m_s, it may be that $m_s \geq 2c_t$. If this happens, the algorithm turns M_t^* into a matrix $\tilde{M}_t$ in row echelon form and removes the $\ell \geq c_t$ zero rows of $\tilde{M}_t$ to produce M_t, as was done above (this leads to changes in $\mathbf{p}$). These operations are authentic.

If $\mathbf{c}_j$ is nonempty and nonzero, the algorithm selects the first nonzero entry of $\mathbf{c}_j$, suppose that it is in row i (of type 1). Let c_{ij} be this nonzero entry and let $\mathbf{x}_i$ be row i in M_t^*. The algorithm then performs authentic row operations in A_t^* using row i to eliminate all the other nonzero elements in column j (note that only rows in M_t^* and J_t^* are modified). It then removes row i from M_t^* to produce M_t^{**}. As in the case of rows, the algorithm defines $\mathbf{p}(\rho) = i$ and $\mathbf{q}(\rho) = j$, while $\mathbf{u}(\rho)$ records the nonzero entries of $[\mathbf{x}_i, \mathbf{y}_i]$. It then updates ρ to $\rho + 1$. If the number of columns rows of M_t^{**} is less than $2c_t = 2c_s - 2$, M_t is set to be M_t^{**}. Otherwise, the algorithm turns M_t^{**} into a matrix $\tilde{M}_t$ in row echelon form with authentic operations and removes $\ell \geq c_t$ zero rows of $\tilde{M}_t$ to produce M_t, as was done in `Join Box` (this leads to changes in $\mathbf{p}$)). Matrix N_t is defined as N_t^*.

3 Running time of the algorithm

In this section, we find an upper bound on the number of operation required to produce matrices P, Q and U as in the statement of Theorem 1 using algorithm `Fast Gaussian Elimination`. If the algorithm is given a tree decomposition $\mathcal{T}$ with r nodes and width k of the underlying graph G of matrix M, it first computes a nice tree decomposition $\mathcal{T}'$ of the same width k with fewer than $4(m+n)$ nodes in time $O(k(r+m+n))$ as discussed above. As it turns out, the number of nodes of each type in $\mathcal{T}'$ is at most $m+n$.

We consider the number of operations performed at each type of node. `Leaf Box` initializes a matrix in $r_t \times c_t = O(k^2)$ trivial steps. `Introduce Box` uses $O(k)$ steps to create a row or column filled with zeros.

For `Join Box` and `Forget Box`, the main cost comes from row operations. As each row has size at most $n_t + c_t \leq 2r_t + c_t \leq 2(r_t + c_t) \leq 2k + 2$, each such row operation requires $O(k)$ additions and multiplications. Regarding `Forget Box`, when vertex v_i associated with a row i or vertex w_j associated with a column j is forgotten, either a new type 1 row or column is created, or a new pivot is found. If the latter happens, the algorithm performs at most $m_t + r_t - 1 = O(k)$ row operations, where each row has size at most $O(k)$, leading to $O(k^2)$ field operations. Additionally, the algorithm may perform row operations to keep M_t or N_t of the right size, just as in `Join Box` (see below). If a new type 1 row or column is created, the algorithm just inserts the new row in M_t or the new column in N_t, possibly performing row operations to keep M_t or N_t of the right size, just as in `Join Box`.

It remains to account for the row operations performed to keep matrices M_t and N_t of the right size. Recall that, every time row operations of this type are necessary to produce M_t, we start with a matrix of dimension $m^* \times c_t$, where $m^* \geq 2c_t$ and we create at least $m^* - c_t \geq c_t$ zero rows. These rows become processed. For N_t, we start with a matrix of dimension $r_t \times n^*$, where $n^* \geq 2r_t$ and we identify at least $n^* - r_t \geq r_t$ columns that are linear combinations of other columns. Again, the columns become processed. We shall keep track of the overall number of field operations performed at these steps by "charging" the rows and columns processed at this step, that is, if f field operations are performed and ℓ rows are processed, each of the ℓ rows is charged with f/ℓ operations. Because of the dimensions of the matrices, getting them to row echelon form takes $O(m^{*2}c_t)$ and $O(n^* r_t^2)$ field operations, respectively. In particular, each row can be charged at most $O(m^* c_t) = O(k^2)$ and each column can be charged at most $O(r_t^2) = O(k^2)$. Since any row or column may be processed at most once in this way, the overall number of field operations that the algorithm performs to keep matrices M_t or N_t of the right size is $O((m+n)k^2))$.

Overall, the number of operations taken by the algorithm to produce the vectors $\mathbf{p}$ and $\mathbf{q}$, in addition to vectors $\mathbf{u}(i)$ and $\mathbf{w}(j)$ for every $i \leq \rho = \operatorname{rank}(A)$ and every $j > \rho$ is $O(k(|\mathcal{T}| + m + n) + k^2(m+n))$.

We now consider the decomposition $U_t = P_t \tilde{A}_t Q_t$. The matrices P and Q are defined using $\mathbf{p}$ and $\mathbf{q}$ with no additional field operations. Indeed, $P = (p_{ij})$ is the permutation matrix such that $p_{\mathbf{p}(i)i} = 1$ for $1 \leq i \leq m$, while $Q = (q_{ij})$ is the permutation matrix such that $q_{\mathbf{q}(i)i} = 1$ for $1 \leq i \leq n$.

For U, the entries may be computed in two ways. They are either obtained from a vector $\mathbf{u}(i)$ with no additional computation, or they are obtained as a linear combination of $O(k)$ earlier values using coefficients recorded in $\mathbf{w}(j)$, which requires $O(k)$ sums and products. The number of operations performed to obtain entries of the second type is at most

$$\sum_{\ell=\operatorname{rank}(A)+1}^{n} (m - \gamma_\ell)O(k) = O((n - \operatorname{rank}(A))km). \tag{6}$$

This concludes the analysis.

4 Consequences of Theorem 1

We conclude the paper with the statement of results that are immediate consequences of Theorem 1.

▶ **Corollary 3.** *If A is an $m \times n$ matrix whose bipartite graph is given with a tree decomposition of width k with $O(m+n)$ nodes and b is a column vector of length n, then using $O(k^2(m+n))$ time and arithmetic operations, it can be decided whether the system of linear equations $Ax = b$ has a solution, and if so, then such a solution can be produced.*

Proof. To get one solution of the system of linear equations $Ax = b$, the algorithm to transform A into row echelon form is augmented by doing all row operations on the right hand side as well. If one ever gets a zero row on the left hand side together with a non-zero value on the right hand side, then the system has no solutions. Otherwise, with the left hand side in row echelon form, a solution is obtained by back-substitution. Hereby, all variables that correspond to columns that have been identified as linearly dependent during the algorithm, or just have no pivot, are set to 0. ◀

▶ **Corollary 4.** *The following hold if A is an $m \times n$ matrix whose bipartite graph is given with a tree decomposition of width k with $O(n)$ nodes:*

(a) *The rank of A can be computed using $O(k^2(m+n))$ time and arithmetic operations.*

(b) *For $m = n$, the determinant of A can be computed using $O(k^2 n)$ time and arithmetic operations.*

Proof. The rank is just the number of non-zero rows of the output matrix U. If one of the columns is a linear combination of previous columns, the determinant is zero. Otherwise, the determinant is just the product of the signs of the permutation matrices P and Q with the product of the diagonal elements of the equivalent matrix in row echelon form. ◀

▶ **Corollary 5.** *For a graph given with a tree decomposition of width k with $O(n)$ nodes, the size of a maximum matching can be computed using $O(k^2 n)$ time and arithmetic operations by a randomized algorithm with with one-sided error that is correct with probability at least $1 - 1/n^c$ for an arbitrary constant. In case of an error, the algorithm reports a suboptimal value.*

Proof. This follows from Theorem 1.4 of Fomin et al. [5] by replacing their $O(k^3 n)$ rank algorithm by our $O(k^2 n)$ rank algorithm. ◀

References

1 U. Bertelè and F. Brioschi. *Nonserial Dynamic Programming.* Elsevier, 1972.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

3 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016. doi:10.1137/130947374.

4 Sally Dong, Yin Tat Lee, and Guanghao Ye. A nearly-linear time algorithm for linear programs with small treewidth: a multiscale representation of robust central path. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1784–1797. ACM, 2021. doi:10.1145/3406325.3451056.

5 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.

6 Martin Fürer, Carlos Hoppen, and Vilmar Trevisan. Efficient diagonalization of symmetric matrices associated with graphs of small treewidth. *Theoretical Computer Science*, 1040:115187, 2025. doi:10.1016/j.tcs.2025.115187.

7 Martin C. Golumbic and Clinton F. Goss. Perfect elimination and chordal bipartite graphs. *Journal of Graph Theory*, 2(2):155–163, 1978. doi:10.1002/jgt.3190020209.

8 Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1):171–186, 1976. doi:10.1007/BF01917434.

9 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. doi:10.1007/BFB0045375.

10 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. *SIAM Journal on Computing*, pages FOCS21–174, November 2023. doi:10.1137/22M147551X.

11 Seymour V. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3(2):119–130, 1961. doi:10.1137/1003021.

12 Richard Peng and Santosh S. Vempala. Solving sparse linear systems faster than matrix multiplication. *Commun. ACM*, 67(7):79–86, 2024. doi:10.1145/3615679.

13 Venkatesh Radhakrishnan, Harry B. Hunt III, and Richard E. Stearns. Efficient algorithms for solving systems of linear equations and path problems. In Alain Finkel and Matthias Jantzen, editors, *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, volume 577 of *Lecture Notes in Computer Science*, pages 109–119. Springer, 1992. doi:10.1007/3-540-55210-3_177.

14 Neil Robertson and Paul D. Seymour. Graph minors. I. excluding a forest. *J. Comb. Theory B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

15 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

16 Donald J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(3):597–609, 1970. doi:10.1016/0022-247X(70)90282-9.

17 Donald J. Rose and Robert E. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM Journal on Applied Mathematics*, 34(1):176–197, 1978.

18 Donald J. Rose, Robert E. Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976. doi:10.1137/0205021.

19 David R. Wood. On tree-partition-width. *European Journal of Combinatorics*, 30(5):1245–1253, 2009. Part Special Issue on Metric Graph Theory. doi:10.1016/j.ejc.2008.11.010.