Proceedings of the 37th Canadian Conference on Computational Geometry (CCCG 2025)

York University, Toronto, Canada

August 13 - 15, 2025

Welcome Note from the Conference Chair

This volume contains the papers presented at the 37th Canadian Conference on Computational Geometry (CCCG 2025), held on August 13–15, 2025, at York University (Keele Campus) in Toronto, Canada. The conference was co-located with the Workshop on Algorithms and Data Structures (WADS 2025), which took place on August 11–13, continuing the successful tradition of co-locating CCCG and WADS that began in 2019. These proceedings will be made available electronically after the conclusion of the conference on the CCCG website: http://www.cccg.ca/.

This year, CCCG received 69 submissions. After the review process, 44 papers were accepted. Each submission received at least three reviews. We thank the Program Committee for their diligent and thoughtful reviewing and discussions, which ensured a high-quality technical program.

We especially thank the invited speakers for their outstanding contributions to the conference program: Dr. Piotr Indyk (MIT), who delivered the *Paul Erdős Memorial Talk*; Dr. David Mount (University of Maryland), who delivered the *Godfried Toussaint Memorial Talk*; and Dr. Birgit Vogtenhuber (Graz University of Technology), who delivered the *Ferran Hurtado Memorial Talk*. Their talks added depth and perspective to CCCG 2025, and we are grateful to them for sharing their insights with the computational geometry community.

We also continued the Best PhD Dissertation Award, which was administered through a dedicated submission track with a review process similar to that of regular papers. This year, four doctoral theses were submitted, each receiving three reviews by members of the CCCG Steering Committee. The award was presented to:

> Da Wei (David) Zheng (University of Illinois Urbana-Champaign), supervised by Timothy Chan
> *"From Geometry to Graphs and Back: Geometric Range Searching and Algorithms in Structured Graphs"*

In addition, the Best Paper Award was selected through a multi-stage process. During the review period, discussions were initiated for papers that received perfect or near-perfect scores. Following additional evaluations and discussions among Program Committee members and the Steering Committee, the award was presented to:

> Elizabeth Munch, Elena Xinyi Wang, and Carola Wenk
> *"The Kinetic Hourglass Data Structure for Computing the Bottleneck Distance of Dynamic Data"*

We gratefully acknowledge the generous financial support of our sponsors: the Centre for Innovation in Computing at Lassonde (IC@L), Research and Innovation at York University, the Lassonde School of Engineering at York University, and the Department of Electrical Engineering and Computer Science (EECS) at York University.

We thank all authors for submitting their work, as well as the presenters for sharing their research at the conference. Our sincere appreciation goes to the members of the Program Committee and the Steering Committee for their time, care, and professionalism throughout the reviewing and decision-making processes. We also thank the student volunteers whose dedication made this conference possible.

Shahin Kamali
Chair of the Program Committee, CCCG 2025

Sponsors

- The Centre for Innovation in Computing at Lassonde (IC@L), York University
- Research and Innovation, York University
- Lassonde School of Engineering, York University
- Department of Electrical Engineering and Computer Science (EECS), York University

Invited Speakers

- Piotr Indyk, MIT (Paul Erdős Memorial talk)
- David Mount, University of Maryland (Godfried Toussaint Memorial talk)
- Birgit Vogtenhuber, Graz University of Technology (Ferran Hurtado Memorial talk)

Program Committee

- Peyman Afshani, Aarhus University, Denmark
- Hee-Kap Ahn, Pohang University of Science and Technology, South Korea
- Yeganeh Bahoo, Toronto Metropolitan University, Canada
- Gill Barequet, Technion, Israel
- Therese Biedl, University of Waterloo, Canada
- Ahmad Biniaz, University of Windsor, Canada
- Sujoy Bhore, IIT Bombay, India
- Jean-Lou De Carufel, University of Ottawa, Canada
- Stephane Durocher, University of Manitoba, Canada
- Bin Han, University of Alberta, Canada
- Hiro Ito, The University of Electro-Communications, Japan
- Shahin Kamali (chair), York University, Canada
- Myroslav Kryven, University of Manitoba, Canada
- Maarten Loffler, Utrecht University, Netherlands
- Amir Nayyeri, Oregon State University, USA
- Rahnuma Islam Nishat, Brock University, Canada
- Andre Nusser, CNRS, Inria Center at Université Côte d'Azur, France
- Denis Pankratov, Concordia University, Canada
- Aditya Potukuchi, York University, Canada
- Christiane Schmidt, Linköping University, Sweden
- Isabelle Sivignon, CNRS, France
- Michiel Smid, Carleton University, Canada
- Frank Staals, Utrecht University, Netherlands
- Tamon Stephen, Simon Fraser University, Canada
- Tim Wylie, University of Texas Rio Grande Valley, USA
- Hamid Zarrabi-Zadeh, Sharif University of Technology, Iran

Additional Reviewers

- Reymond Akpanya, RWTH Aachen University
- Anna Brötzner, Malmö University
- Chirag Kaudan, Oregon State University
- Kian Mirjalali, Sharif University of Technology
- Joseph S.B. Mitchell, Stony Brook University
- Tanmay Inamdar, IIT Jodhpur

Organizing Committee

- Shahin Kamali (York University)
- Eric Ruppert (York University)

Volunteers

- Shalom Asbell
- Shayan Ghalehdar
- Samiha Ibnat
- Katherine Ling
- Natan Melzer
- Parsa Merat
- Melina Najimi
- Elmira Onagh
- Elham Razi
- Seyed Mohammad Seyed Javadi
- Miodrag Tasic
- Saba Yazdani
- Taisia Zhizhina
- Wenhao Zhu

Conference Program

Day 1

Paul Erdős Memorial Lecture

- *Piotr Indyk*
  Towards overcoming the reranking bottleneck 1

Session 1: Graph Embedding & Layout Constraints

- *Rustem Kakimov, Xing Tan*
  On Upward Book Embeddability of DAGs 2
- *Therese Biedl, Erin Chambers, Irina Kostitsyna, Günter Rote*
  Sweeping x-monotone pseudolines 12
- *Hong Duc Bui*
  Straight-line Orthogonal Drawing of Complete Ternary Tree Requires $O(n^{1.032})$ Area 22
- *Stephane Durocher, Myroslav Kryven, Tamara Mchedlidze*
  On Upward Planar Embeddings of Paths with Partially Fixed Vertices 29

Session 2: Noise, Entropy, and Dynamics

- *Michael T. Goodrich, Vinesh Sridhar*
  Optimal Parallel Algorithms for Convex Hulls in 2D and 3D under Noisy Primitive Operations 36
- *David Eppstein, Michael T. Goodrich, Abraham M. Illickan, Claire A. To*
  Entropy-Bounded Computational Geometry Made Easier and Sensitive to Sortedness 53
- *Tanmay Inamdar, Sounak Modak, Kushal Singanporia*
  PTAS for Stabbing Unit Squares and Variants 62

Session 3: Polygons, Polyhedra, and Polyominos

- *Reymond Akpanya, Adi Rivkin, Frederick Stock*
  Inside-Out Dissections of Polygons and Polyhedra 71
- *MIT CompGeom Group, Hugo A. Akitaya, Erik D. Demaine, Adam Hesterberg, Thomas C. Hull, Anna Lubiw, Jayson Lynch, Klara Mundilova, Chie Nara, Joseph O'Rourke, Frederick Stock, Josef Tkadlec, Ryuhei Uehara*
  Quasigeodesics on the Cube 78
- *David Eppstein*
  Decremental Greedy Polygons and Polyhedra Without Sharp Angles 85
- *Gill Barequet, Neal Madras, Johann Peters*
  On t-fold Totally-Concave Polyominoes 92

Day 2

Godfried Toussaint Memorial Lecture

- *David Mount*
  Computational Geometry from a Non-Euclidean Perspective 99

Session 4A: Visibility and Guarding

- *Arash Ahadi, Ahmad Biniaz, Mohammad Hashemi, Ali Nakhaeisharif*
  Guarding Polygons With Mutually Visible π-Guards 100
- *Anna Brötzner, Bengt J. Nilsson, Christiane Schmidt*
  Multiple Watchman Routes in Staircase Polygons 107
- *Gabriel Aldous, Seth Barber, Alper Üngör*
  On Super-Guarding Convex and Star-Shaped Polygons 114
- *Matt Gibson-Lopez, Erik Krohn, Zhongxiu Yang*
  The VC-Dimension of Limited Visibility on the Boundary of a Simple Polygon 122

Session 4B: Transformations and Metric Extensions

- *Takumi Shiota, Yudai Enomoto, Masashi Gorobe, Takashi Horiyama, Tonan Kamata, Toshiki Saitoh, Ryuhei Uehara*
  The Number of Non-overlapping Unfoldings in Convex Polyhedra 133
- *Shoon Mineyoshi, Kazuhisa Seto, Takashi Horiyama*
  On the Diameters of Reconfiguration Graphs and ZDD-Based BFS-Algorithm for Optimal Reconfiguration Problems of Optimal Ladder Lotteries 140
- *Donald R. Sheehy*
  Fast Approximate Lipschitz Extensions in Doubling Metrics 147
- *Jonathan James Perry, Benjamin Raichel*
  Hausdorff Edit Distance 154

Session 5A: Tree and Center Problems

- *Stephane Durocher, Arman Heydari, J. Mark Keil, Debajyoti Mondal*
  Minimum Anchored Steiner Trees with Removable Obstacles 164
- *Majid Mirzanezhad, Arash Rafiey*
  Approximate and Exact Geometric Generalized Minimum Spanning Trees 173
- *Stephane Durocher, Sahar Lamey, Pak Ching Li*
  Fault-Tolerant Euclidean k-Centres 183
- *Taehoon Ahn, Sang Won Bae, Sang Duk Yoon*
  The Orthogonal Two-Line Center Problem 190

Session 5B: Puzzles & Complexity

- *Timothy Gomez, Robert Schweller, Tim Wylie*
  Covert Computation in the Prebuilt aTAM 197
- *Steven van den Broek, Marc van Kreveld, Soeren Terziadis*
  Partitioning Colored Points into Monochromatic Islands is NP-Complete 204
- *Takeshi Yamada, Tom van der Zanden, Ryuhei Uehara*
  Puzzles are hard enough just by rotations 211
- *Hugo A. Akitaya, Matias Korman, Frederick Stock*
  Input-Sensitive Reconfiguration of Sliding Cubes 216
- *Jonathan Gabor, Aaron Williams*
  Reverse Rush Hour is NP-Complete 223

Day 3

Session 6A: Probing Geometry & Query Optimization

- *Shuhao Tan, David M. Mount*
  Approximating Metric Depth Queries 231
- *Chirag Kaudan, Amir Nayyeri*
  Reconstructing Bounded Treelength Graphs with Linearithmic Shortest Path Distance Queries 243
- *Ofek Gila, Michael T. Goodrich, Zahra Hadizadeh, Daniel S. Hirschberg, Shayan Taherijam*
  The Marco Polo Problem: A Combinatorial Approach to Geometric Localization 249
- *Ofek Gila, Michael T. Goodrich, Zahra Hadizadeh, Daniel S. Hirschberg, Shayan Taherijam*
  The Rectilinear Marco Polo Problem 262

Session 6B: Graphs and Approximation

- *A. Karim Abu-Affash, Paz Carmi, Iliya Lisin*
  Approximating 2-Clique in Unit Disk Graphs 275
- *Anastasiia Tkachenko, Haitao Wang*
  Computing Maximum Cliques in Unit Disk Graphs 283
- *Thomas Schibler, Subhash Suri, Jie Xue*
  Approximation and Hardness of Polychromatic TSP 292
- *Bubai Manna*
  Minimum Selective Subset on Some Graph Classes 299

Ferran Hurtado Memorial Lecture

- *Birgit Vogtenhuber*
  Combinatorial Reconfiguration: Flipping Non-Crossing Spanning Trees 306

Session 7A: Motion Planning and Robotics

- *Eduard Eiben, Iyad Kanj, Salman Parsa*
  Motion Planning of Disk and Rectangular Robots 307
- *Jared Coleman, Evangelos Kranakis, Danny Krizanc, Oscar Morales-Ponce*
  Optimal Delivery with a Faulty Drone 315
- *Aaron T. Becker, Sándor P. Fekete, Jonas Friemel, Ramin Kosfeld, Peter Kramer, Harm Kube, Christian Rieck, Christian Scheffer, Arne Schmidt*
  Efficient Reconfiguration of Tile Arrangements by a Single Active Robot 322
- *Sharareh Alipour, Arash Ahadi, Kajal Baghestani*
  Improved Wake-Up Time For Euclidean Freeze-Tag Problem 331

Session 7B: Geometric Covering and Packing

- *Gang Liu, Haitao Wang*
  Minimum-Weight Half-plane Hitting Set 340
- *David Alcántara, Francisco Criado, Francisco Santos*
  Covering radii of 3-zonotopes and the shifted lonely runner conjecture 347
- *Hong Duc Bui*
  Square Packing with Asymptotically Smallest Waste Only Needs Good Squares 355

Best Paper Award

- *Elizabeth Munch, Elena Xinyi Wang, Carola Wenk*
  The Kinetic Hourglass Data Structure for Computing the Bottleneck Distance of Dynamic Data 362

Open Problems

- *Hugo Akitaya, Shahin Kamali, Seyed-Mohammad Seyed-Javadi, Frederick Stock*
  Open Problems from CCCG 2025 371

Towards overcoming the reranking bottleneck

Paul Erdős Memorial Lecture

Piotr Indyk[*]

Reranking is a popular approach to information retrieval. It proceeds in two stages. In the first stage, a "quick-and-dirty" data structure retrieves a shortlist of r points closest to the query, where the length of the shortlist r is larger than the desired output k. In the second stage, the shortlist is post-processed to identify $k \ll r$ points that satisfy the desired objective. For example, the postprocessing could identify the k most "diverse" points in the shortlist or use a "slower-but-accurate" distance metric to identify the best answers. Despite its popularity, it has various drawbacks; notably the quality of the output is limited by the accuracy of the first stage.

In this talk, I will discuss an alternative to reranking, which fuses the two stages into a single search procedure. The new approach crucially uses recent developments in graph-based algorithms for high-dimensional similarity search, as well the tools developed to analyze such algorithms.

[*]CSAIL, MIT, indyk@mit.edu

On Upward Book Embeddability of DAGs

Rustem Kakimov* Xing Tan*†

Abstract

The k-page upward book embedding problem ($kUBE$) for directed acyclic graphs (DAGs) is of significant relevance to network visualization and circuit design. It is known to be NP-complete for $k \geq 2$. This study investigates $kUBE$ embeddability and its relationship to vertex count, edge density, and page count. We begin by exhaustively enumerating all small DAGs to identify embeddability patterns, and then extend the analysis to larger instances via representative sampling. Our results confirm that embeddability decreases with increasing graph density and improves with the number of pages. By encoding $kUBE$ as a Boolean satisfiability problem, we demonstrate the effectiveness of SAT-based methods in addressing this computationally challenging class of problems.

1 Introduction

The book embedding problem (BE) involves arranging a graph's vertices along a spine and distributing edges across k pages without crossings on the same page [3]. The k-page upward book embedding problem ($kUBE$) for directed acyclic graphs (DAGs) requires edges to be oriented upward, relevant for applications like network visualization and circuit routing [7,8]. While *1UBE* is solvable in linear time [13], $kUBE$ is NP-complete for $k \geq 2$ [1,5].

We investigate $kUBE$ embeddability of DAGs, focusing on the effects of vertex count (n), edge count (m), and page count (k). Our research addresses:

- **Impact of Graph Density** As edge density (m/n) increases for fixed n, does the percentage of k-page embeddable DAGs decrease, and can we observe a phase transition?
- **Effect of Pages** How does embeddability change as k increases beyond 2?

We start with small DAGs (n small), enumerating all DAGs on n vertices, to identify any embeddability patterns. We then consider larger DAGs (obtained via sampling). Our analysis reveals density-driven phase transitions and page-dependent patterns. Using a SAT-based approach with the SAT-1 encoding [15] and the SAT solver Kissat [4], we efficiently verify embeddability, providing insights into the complexity of $kUBE$.

*Department of Computer Science, Lakehead University, Canada. {rkakimov, xing.tan}@lakeheadu.ca

†Corresponding author.

2 Preliminaries

This section outlines the core concepts and techniques used in our study. We introduce book embedding and its upward variant for DAGs, describe methods for enumerating small DAGs and sampling random DAGs, and explain the SAT-based approach for verifying embeddability, which underpins our experimental investigations.

2.1 Book Embedding

Definition 1 ***k-Page Book Embedding for Undirected Graphs*** *A book embedding of a graph* $G = (V, E)$ *consists of*

1. *a linear ordering* $\pi : V \rightarrow \{1, 2, \ldots, |V|\}$ *of vertices along the line called the spine of a book, and*
2. *an assignment* $\sigma : E \rightarrow \{1, \ldots, k\}$ *of individual edges to one of the k pages such that no edges assigned to the same page geometrically cross. More precisely, for any two edges* $\{u_1, v_1\}$ *and* $\{u_2, v_2\}$ *on the same page, such that* $\pi(u_1) < \pi(v_1)$ *and* $\pi(u_2) < \pi(v_2)$*, the following two conditions are not allowed:* $\pi(u_1) < \pi(u_2) < \pi(v_1) < \pi(v_2)$ *and* $\pi(u_2) < \pi(u_1) < \pi(v_2) < \pi(v_1)$.

Definition 2 ***k-Page Upward Book Embedding for Directed Graph*** *An upward book embedding of a directed graph* $G = (V, E)$ *also consists of a linear ordering* $\pi : V \rightarrow \{1, 2, \ldots, |V|\}$ *of vertices along the spine of the book, and an assignment* $\sigma : E \rightarrow \{1, \ldots, k\}$ *of edges to k pages such that no two edges geometrically cross. Additionally, all the edges should be oriented in the same direction. That is, for any edge* (u, v), $\pi(u) < \pi(v)$ *must hold.*

Definition 3 ***kBE Problem*** *A k-Page Book Embedding Problem (kBE) is defined as follows: Given an undirected graph G, does there exist a k-page book embedding for G?*

Definition 4 ***kUBE Problem*** *A k-Page Upward Book Embedding Problem (kUBE) is defined as follows: Given a directed graph G, does there exist a k-page upward book* *embedding for G? Note that, an upward book embedding can possibly exist in a directed graph G only if the graph is acyclic. In other words, the graph G must be a DAG.*

Figure 1 presents an example of a *kUBE* problem instance with three pages (i.e., $k = 3$). The input is a DAG with 6 nodes and 15 edges, shown in Figure 1a. The goal is to embed it into three pages. The resulting solution is shown in Figure 1b, where the nodes are placed in a topological order along a horizontal spine, and edges—colored to indicate page assignment—are drawn without crossings within the same page.

Both the kBE and kUBE problems are computationally challenging, as indicated in the following theorems.

Theorem 1 *For any $k \geq 2$, kBE Problem is NP-complete. Meanwhile, 1BE can be solved in $O(n)$ time. This result is due to Wigderson [25].*

Theorem 2 *For any $k \geq 2$, kUBE is NP-complete. Meanwhile, 1UBE can be solved in $O(n)$ time. These results are due to Bekos et al. [1] and Heath and Pemmaraju [13], respectively.*

2.2 Enumerating All Small DAGs

We use an algorithm that generates all DAGs for n nodes, see Algorithm 1. It begins by initializing the node set $V = \{0, 1, \ldots, n-1\}$ and the set E_{all} of all possible directed edges between distinct nodes. The algorithm then iterates over all subsets of E_{all}, constructs the corresponding graph, and checks for acyclicity via topological sort. Subsets that form valid DAGs are collected.

Given the exponential number of edge subsets—reflecting the inherent complexity of enumerating all DAGs, the algorithm adopts a brute-force approach. Each DAG verification requires $O(n^2)$ time, resulting in exponential time complexity overall. Additionally, storing all DAGs demands exponential space, limiting practical use to $n \leq 10$. Nonetheless, our implementation emphasizes simplicity and completeness. By exhaustively exploring edge combinations, it ensures every DAG is generated, making it well-suited for small n or theoretical analysis where clarity and correctness outweigh efficiency considerations.

Algorithm 1 Generate all DAGs with n nodes

```
1: Initialization : dags ← [ ]; $V \leftarrow \{0,1,\ldots,n-1\}$; $E_{\text{all}} \leftarrow \{(u,v) \mid u \neq v, u,v \in V\}$
2: for all edge subsets $E'$ of $E_{\text{all}}$ do
3:     $G \leftarrow$ CreateGraph($V, E'$)
4:     If (IsDAG($G$))  dags ← dags ∪ {$E'$}
5: end for
6: return dags
```

2.3 Generating Random DAGs via Sampling

When n is large, generating and analyzing all possible DAGs becomes computationally infeasible due to the exponential growth of the DAG instance space. Instead, we sample a subset of DAGs to study their properties, making the approach computationally manageable. It is crucial, however, to ensure that the sampled subset is representative, as an inappropriate sampling method can skew results and misrepresent the underlying properties of the DAG population. For example, a biased sampling method might over-represent certain structures—such as those with more valid topological orderings—leading to inaccurate conclusions about phenomena like satisfiability transitions.

To balance computational efficiency and statistical precision, we adopt a two-tiered strategy that integrates existing methods with our own adaptations into a cohesive framework. First, we propose Algorithm 2, an efficient topological-order-based heuristic algorithm that generates a broad set of DAGs across all edge counts m (ranging from 0 to $n(n-1)/2$) for each n. We then complement this with targeted uniform sampling in the critical phase-transition region using the Kuipers–Moffa method [19]. The combination of random sampling for broad coverage and uniform sampling for key areas ensures both efficiency and accuracy in our experimental analysis.

Algorithm 2 Generate a random DAG of n nodes and m edges

```
Require: $n$, $m$
Ensure: A list of $m$ edges forming a random DAG
1: $order \leftarrow$ Shuffle($[0,1,\ldots,n-1]$)    ▷ Random permutation of nodes
2: $all\_edges \leftarrow \{(order[i], order[j]) \mid 0 \leq i < j < n\}$    ▷ All forward edges
3: $chosen\_edges \leftarrow$ UniformSampling($all\_edges, m$)    ▷ Select $m$ edges uniformly
4: return Sort($chosen\_edges$)    ▷ Return sorted edge list
```

The topological-order method, detailed in Algorithm 2, generates DAGs by shuffling node orders and uniformly selecting m edges from all possible forward edges, producing 100 DAGs per m (or 30 for high n and k). Meanwhile, the Kuipers–Moffa method, implemented via the `unifDAG` R package [20], ensures uniform sampling by recursively constructing DAGs through outpoint removal and reverse connection sampling. We specifically use the approximate method for broad exploration across all m, reducing to 30 samples for computationally intensive cases ($n = 20, k \geq 6$), and validate its empirical adequacy against uniform sampling for selected n. We switch to Kuipers–Moffa sampling in the phase-transition region, where satisfiability is about

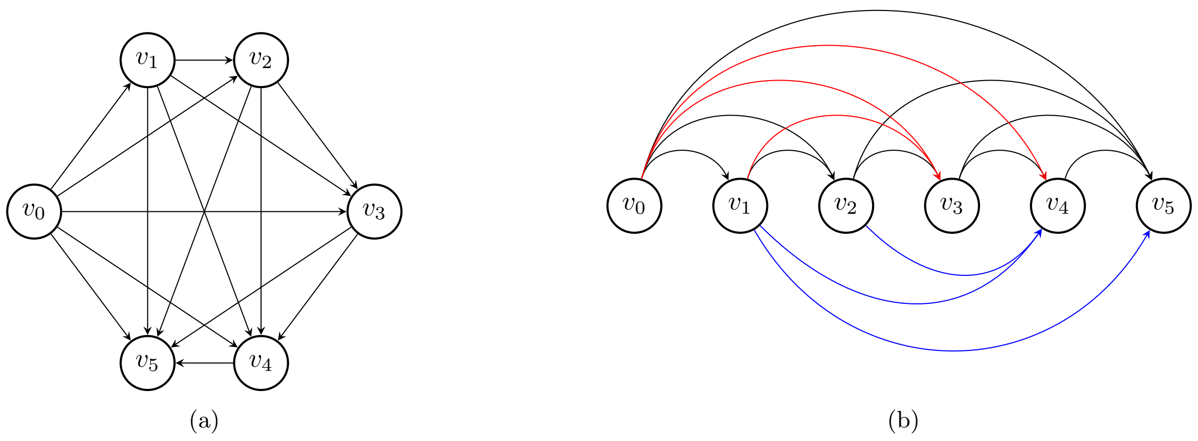


Figure 1: (a) A DAG with 6 nodes and 15 edges (maximal). (b) Its 3-page Upward Book Embedding (*3UBE*), where edges are partitioned into three pages highlighted in black, blue, and red.

50%. In this study, we used this sampling approach to generate DAGs for $n \in \{7, 8, \ldots, 20\}$, storing them as edge lists for analysis.

2.4 SAT for Embeddability Verification

To investigate the embeddability of DAGs in the *kUBE* problem, we utilize SAT-1, a specific SAT encoding for *kUBE* [15]. SAT-1 has its roots in the SAT encoding originally proposed by Bekos et al. [1, 2] for undirected book embedding problems. Several revisions have been made to address the unique requirements of linear ordering of vertices and directed edges in kUBE problems. That is, it requires a linear ordering $\pi : V \to \{1, \ldots, |V|\}$ of vertices along a spine and an assignment $\sigma : E \to \{1, \ldots, k\}$ of edges to k pages, such that edges are upward (i.e., $\pi(u) < \pi(v)$ for all $(u, v) \in E$) and no same-page edges cross geometrically.

SAT-1 transforms these constraints into a Boolean satisfiability formula called $\mathcal{F}_1(G, k)$ in conjunctive normal form (CNF), enabling efficient verification with modern SAT solvers. For a DAG $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, SAT-1 defines three key variable sets:

1. $L(v_i, v_j)$ for vertex ordering, which is true if $\pi(v_i) < \pi(v_j)$;
2. $EP(e_i, p)$ for edge-to-page assignments, which is true if edge e_i is on page $p \in \{0, \ldots, k-1\}$; and
3. $X(e_i, e_j)$ for same-page edge pairs, which is true if e_i and e_j share a page.

The formula is satisfiable if and only if G admits a valid k-page upward embedding, with a polynomial size of $O(n^2 + m^2 + mk)$ variables and $O(n^3 + m^2)$ clauses.

SAT-1's strength lies in its completeness and tailored adaptation to *kUBE*'s directed nature. The encoding for *kUBE* enforces topological ordering via clauses $L(u, v)$ for each $(u, v) \in E$, ensuring upward directionality. Compared with the original encodings [1, 2], edge-crossing constraints are simplified by considering only direction-consistent permutations (e.g., (a, c, b, d) or (c, a, d, b) for edges (a, b) and (c, d)), reducing clause overhead while maintaining correctness [1]. Transitivity of the vertex order is enforced through CNF clauses like $[\neg L(v_i, v_j), \neg L(v_j, v_k), L(v_i, v_k)]$, ensuring a consistent linear arrangement. This adaptability, combined with the efficiency of solvers, makes SAT-1 a practical tool for embeddability testing.

We have conducted empirical evaluations of SAT-1, providing strong evidence of its efficiency based on benchmarks such as the North dataset (1277 DAGs, [12]) and large grid graphs. In our current research, we apply SAT-1 to systematically assess *kUBE* embeddability across DAGs of varying sizes, densities, and structural properties. This enables us to probe phenomena such as phase transitions, the effect of increasing k and potentially other structural factors influencing *kUBE* embeddability.

2.5 Kissat: An Efficient SAT Solver

In this study, we use Kissat [4, 10, 11], a modern SAT solver, to handle SAT-1. Kissat follows the conflict-driven clause learning (CDCL) framework and incorporates optimizations such as inprocessing and phase saving to improve efficiency. Its ability to handle large constraint systems with structured variable dependencies makes it particularly suitable for satisfiability problems arising in graph theory, where constraints often reflect structural properties of graphs. In our case, Kissat is used to determine the embeddability of DAGs under *kUBE* constraints by efficiently exploring large search spaces and resolving complex logical dependencies.

3 Upward Book Embeddability of DAGs

This section examines the upward book embeddability of DAGs, analyzing small enumerable instances, theoretical page bounds, larger random samples, and phase transitions, to characterize how graph size, edge density, and page numbers influence embeddability.

3.1 Embeddability of Small DAGs

We begin by investigating the upward book embeddability of DAGs with at most $n = 6$ vertices. In these small cases, all possible DAGs can be exhaustively enumerated. This examination provides significant insight and valuable perspectives on the broader context of larger problems.

We assess the embeddability of all enumerated DAGs for $n = 4, 5, 6$ using the SAT approach described in Section 2.4. For each n, we generate all possible DAGs with Algorithm 1, categorize them by edge count m (as shown in Tables 2, 3, and 4), and test each instance for k-page upward book embeddability with $k = 1$ and $k = 2$. The SAT-1 encoding translates a *kUBE* problem into a Boolean satisfiability problem, which is then solved using a SAT solver. For each combination of n, m, and k, we compute the percentage of satisfiable DAGs by dividing the number of SAT outcomes (indicating embeddability) by the total number of DAGs at that m, providing the data points to plot the curves in Figure 2.

Figure 2 illustrates the percentage of DAGs that can be embedded using k pages as a function of the total number of edges m, for varying numbers of nodes n. The x-axis represents m, ranging from 0 to 15, while the y-axis shows the percentage of satisfiable DAGs, from 0% to 100%. Note that when $n = 6$, the maximal possible m value is 15; when $n = 5$, the maximal m value is 10; and when $n = 4$, the maximal m is 6. The data is categorized by n and k: filled circles denote $n = 6$, crosses denote $n = 5$, and triangles denote $n = 4$, with blue representing $k = 1$ and orange representing $k = 2$.

For all configurations, we have the following observations: 1) The percentage starts at 100% when $m = 0$, as a DAG with no edges is trivially embeddable. As m increases, the percentage decreases steadily and smoothly, reflecting the growing complexity of the DAG. For any specific curve with a fixed n, this indicates that embeddability decreases as the graph density (defined as m/n) increases. 2) For a fixed n, the percentage of satisfiable DAGs is consistently higher for $k = 2$ (orange) than for $k = 1$ (blue). For instance, the solid orange line ($n = 6, k = 2$) remains above the solid blue line ($n = 6, k = 1$) across all m, indicating that using two pages provides greater flexibility to satisfy the DAG, and 3) When n is fixed and k is repeatedly increased (i.e., the total number of pages allowed for the embedding), there exists a threshold k value at which all DAGs of size n are embeddable. For small DAGs (where $n = 4, 5, 6$), these threshold k values are obtained using the SAT-1 encoding and a SAT solver, and are presented in Table 1.

Table 1: Minimal number of pages k required for all size-n DAGs to be embeddable ($n = 4, 5, 6$).

DAG size (n)	4	5	6
Minimal # of pages required (k)	2	3	3

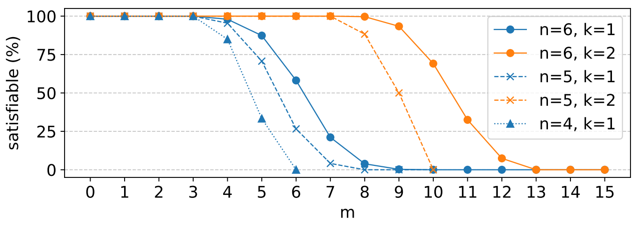


Figure 2: Percentage of satisfiable DAGs as a function of the number of edges m, for different numbers of nodes n and pages k. Filled circles represent $n = 6$, crosses represent $n = 5$, and triangles represent $n = 4$. Blue denotes $k = 1$, and orange denotes $k = 2$.

The table explains why there are only five curves in Figure 2. In addition, we also manually proved that when $k \geq 3$, all $n = 6$ DAGs are embeddable in the following theorem.

Proposition 3 *All $n = 6$ DAGs can be upward book-embedded using 3 pages.*

Proof. We know that for complete cases (where each node has an edge to every node to its right), we are able to find a 3-upward book embedding, as shown in the right subfigure of Figure 1. This implies that any $n = 6$ DAG contains fewer edges than the "maximal DAG", can be embedded using 3 pages. We conclude that all $n = 6$ DAGs can be upward book-embedded using three pages. □

3.2 Embeddability of Random DAGs

When n increases, enumerating all DAGs quickly becomes impractical. For instance, when $n = 10$, the total number of DAGs is approximately 4.18×10^{18} ([24, A003024]). To explore the embeddability properties of larger DAGs, we rely on sampling techniques, which are introduced in Section 2.3, to generate random DAG instances. We used this method to obtain 4,600 DAGs with $n = 10$, 10,600 DAGs with $n = 15$, and 19,100 DAGs with $n = 20$, the latter of which was downsampled to 5,730 instances for $k \geq 6$.

We then benchmarked SAT solver runtimes across increasing values of k. The experimental setup covered the following configurations, with the number of edges m ranging from 0 to its maximum in all cases: (a) $n = 10$, with $k = 2, 3, 4$; (b) $n = 15$, with $k = 2$ to 7; and (c) $n = 20$, with $k = 2$ to 7.

Due to the prohibitively high computational cost at these values, results for $k = 8$ and up are not included in the current paper. For each value of n, we analyzed how increasing the edge count m, and thus the graph density m/n, impacts both the embeddability of DAGs in $kUBE$ and the runtime of the SAT solver. Results are presented in Figure 3, where each subfigure corresponds to a fixed n and consists of two aligned plots: the top shows the satisfiability rate as a function of edge density m/n, and the bottom shows the mean run time of the SAT solver. In both plots, the x-axis is divided into bins of width 0.2, and the mean is calculated per bin.

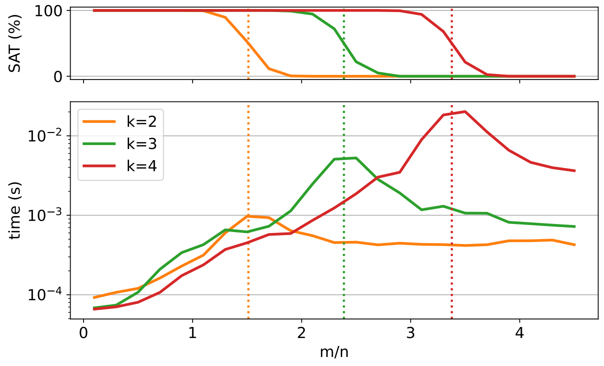
Recall that (Section 3.1) when DAGs are small and full enumeration was feasible, we identified two key patterns: 1) embeddability starts at 100% with $m = 0$, dropping smoothly as m increases due to the increasing density of the graph (m/n), and 2) higher k values, such as $k = 2$ versus $k = 1$, consistently improve embeddability by offering greater flexibility. Interestingly, these patterns hold in our experiments with sampled larger DAGs at $n = 10$, 15, and 20, where we observe a comparable gradual decline in embeddability as edge density grows, alongside a significant improvement in embedding success with larger k values, closely mirroring the behavior of their smaller, fully enumerated counterparts.

The bottom plots of the three subfigures, which depict mean computation time, exhibit a consistent pattern across all configurations of n and k:

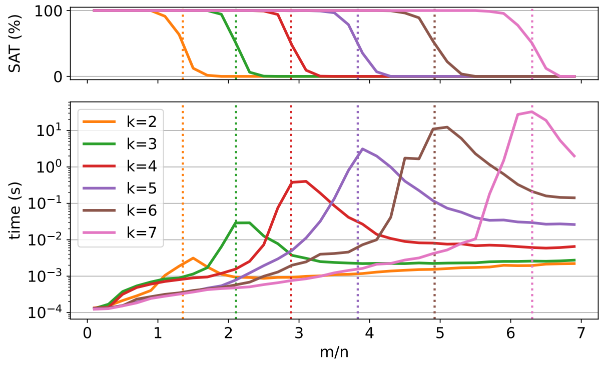
1. For each (n, k) combination, the peak runtime is aligned with the 50% embeddability threshold (indicated by the vertical dotted line). This suggests that computational complexity is maximized when the proportion of SAT to UNSAT instances is approximately balanced.

2. For larger values of k, the peak runtime increases substantially compared to smaller k. As the y-axis uses a logarithmic scale, we can observe that the peak runtime for k is typically one order of magnitude higher than that for $k - 1$.

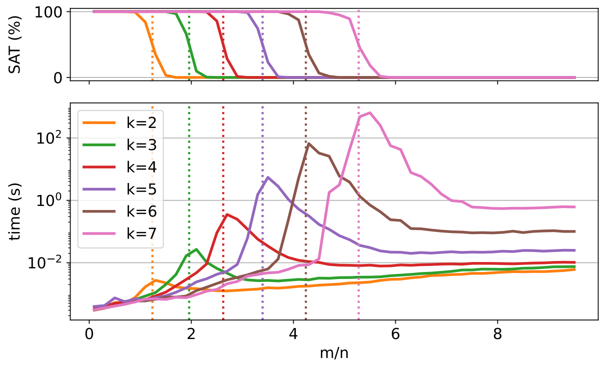
3. Prior to the peak, the runtime differences across k values are relatively small. Notably, there is consistently a range of m/n values where higher k in-



(a) $n = 10$



(b) $n = 15$



(c) $n = 20$

Figure 3: Embeddability and computational cost for sampled DAGs with $n = 10$, 15, and 20. Each subfigure consists of: (Top) Fraction of embeddable DAGs versus edge count m, plotted for varying k values, showing a consistent decline with increasing m. (Bottom) Mean computation time aligned with m, displaying a peak near the 50% embeddability threshold (vertical dashed line), followed by a linear rise, with higher k values consistently increasing overall times for all n.

stances are solved faster than those with lower k. This range tends to occur near the peak runtime for the lower k, likely because at those m/n values, the higher k instances are nearly always embeddable—possibly even trivially so.

4. After the peak, as m/n continues to increase, instances become predominantly non-embeddable. In this regime, higher k instances consistently take longer to solve. This is likely due to the increased number of SAT variables and constraints introduced at higher k, which makes proving unsatisfiability more computationally demanding.

4 Discussion and Conclusion

To recap, this study investigates the *kUBE* problems for DAGs (NP-complete for $k \geq 2$), by analyzing embeddability across vertex count (n), edge count (m), and page count (k). Employing exhaustive enumeration of small DAGs, uniform sampling of larger instances, and SAT-based verification, we derive useful insights into the structural and computational properties of *kUBE*. We observe a consistent decline in embeddability as graph density (m/n) increases. For small DAGs ($4 \leq n \leq 6$), exhaustive enumeration reveals a smooth transition from fully embeddable at low density to entirely non-embeddable at high density. This pattern persists in larger DAGs ($n \leq 20$), where uniform sampling confirms that sparse graphs are universally embeddable, while dense graphs are predominantly not, highlighting the difficulty of embedding dense structures within limited page constraints (Figures 2, 3). Meanwhile, increasing the number of pages ($1 \leq k \leq 6$) significantly improves embeddability across all values of n. Higher values of k consistently shift embeddability curves upward, reflecting greater flexibility in edge assignments and enabling previously non-embeddable DAGs to admit valid embeddings (Figures 2, 3).

The SAT-1 encoding and the Kissat solver efficiently verified embeddability across diverse datasets, including 1,277 North Graph DAGs and 45,000 sampled instances, managing large graphs ($n = 20$) despite peak computational complexities. The scalability of SAT solvers suggests their potential for addressing other NP-complete graph-theoretic problems. It is also noted that the case $k = 1$ does not exhibit a computational runtime peak, in contrast to the pronounced peaks observed for $k \geq 2$. Since the $k = 1$ case is solvable in poly-time, this contrast suggests a possible connection between runtime behavior and underlying computational complexity. That is, examining whether the presence or absence of runtime peaks in other parameterized problems could serve as an empirical indicator of complexity class between poly-time solvable and NP-hard cases.

While our findings may not reveal fundamentally new structural properties, they provide empirical confirmation of a known pattern of *kUBE* embeddability across varying graph densities and page counts. This reinforces the utility of SAT-based formulations (e.g., [14]) for systematically exploring the parameter space of hard combinatorial problems. Although upward book embeddings are primarily of theoretical interest, our SAT-based approach demonstrates practical scalability in evaluating embeddability over large datasets. This may support future optimization variants relevant to multi-layer circuit design [7, 17] or constrained network visualization [21, 23], where related abstractions are sometimes employed. Our work also aligns with broader studies of layout models such as queue and stack layouts [8], which share similar structural constraints.

A closer examination of the embeddability phase transitions reveals a sharp decline in satisfiability from 100% to 0% at specific m/n thresholds—ranging from 0.865 for $k = 1$ to 5.098 for $k = 6$—which quantifies how increasing page count permits denser DAGs to remain embeddable (Figure 4). Notably, these thresholds closely align with observed runtime peaks for $k \geq 2$, indicating a tight coupling between structural transition points and computational difficulty. In contrast, the absence of a peak for $k = 1$ reflects its polynomial-time solvability and qualitatively different runtime behavior. These patterns suggest a potential diagnostic role for runtime peaks in identifying complexity class transitions. A full characterization of these phase boundaries, along with their implications for solver performance and DAG structure, is presented in the appendix.

Acknowledgements

We sincerely thank all the reviewers for their thoughtful and constructive feedback, which greatly helped improve the quality and presentation of this work. We also gratefully acknowledge the support of the Discovery Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC), as well as the research grant provided by the Faculty of Science and Environmental Studies at Lakehead University.

References

[1] M. A. Bekos, G. Da Lozzo, F. Frati, M. Gronemann, T. Mchedlidze, and C. N. Raftopoulou. Recognizing DAGs with page-number 2 is NP-complete. *Theoretical Computer Science*, 946:113689, 2023.

[2] M. A. Bekos, M. Kaufmann, and C. Zielke. The book embedding problem from a SAT-solving perspective. In *Graph Drawing and Network Visualization: 23rd International Symposium, GD 2015, Los Angeles, CA, USA, September 24-26, 2015, Revised Selected Papers 23*, pages 125–138. Springer, 2015.

[3] F. Bernhart and P. C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331, 1979.

[4] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt. CaDiCal, Gimsatul, IsaSAT and Kissat entering the SAT competition 2024. *SAT Competition 2024*, page 8, 2024.

[5] C. Binucci, G. Da Lozzo, E. Di Giacomo, W. Didimo, T. Mchedlidze, and M. Patrignani. Upward book embeddability of st-graphs: Complexity and algorithms. *Algorithmica*, 85(12):3521–3571, 2023.

[6] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the 12th IJCAI*, pages 331–337, 1991.

[7] F. R. Chung, F. T. Leighton, and A. L. Rosenberg. Embedding graphs in books: a layout problem with applications to VLSI design. *SIAM Journal on Algebraic Discrete Methods*, 8(1):33–58, 1987.

[8] V. Dujmović and D. R. Wood. Stacks, queues and tracks: Layouts of graph subdivisions. *Discrete Mathematics & Theoretical Computer Science*, 7, 2005.

[9] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

[10] J. K. Fichte, M. Hecher, and S. Szeider. A time leap challenge for SAT-solving. In *International Conference on Principles and Practice of Constraint Programming*, pages 267–285. Springer, 2020.

[11] N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda. SAT competition 2020. *Artificial Intelligence*, 301:103572, 2021.

[12] Graph Drawing Community. The International Symposium on Graph Drawing and Network Visualization. Accessed: 2025-02-20.

[13] L. S. Heath and S. V. Pemmaraju. Stack and queue layouts of directed acyclic graphs: Part ii. *SIAM Journal on Computing*, 28(5):1588–1626, 1999.

[14] M. J. H. Heule and S. Szeider. A SAT approach to clique-width. *ACM Transactions on Computational Logic*, 16(3):24:1–24:27, 2015.

[15] R. Kakimov and X. Tan. SAT for upward book embedding: An empirical study. In *Proceedings of the 38th Canadian Conference on Artificial Intelligence*. Canadian Artificial Intelligence Association, 2025.

[16] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[17] A. Knippel and F. Lardeux. The multi-layered network design problem. *European Journal of Operational Research*, 245(1):1–12, 2015.

[18] M. R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 13(1–2):15–20, 1967.

[19] J. Kuipers and G. Moffa. Uniform random generation of large acyclic digraphs. *Statistics and Computing*, 25:227–242, 2015.

[20] J. Kuipers and G. Moffa. *unifDAG: Uniform sampling of directed acyclic graphs*, 2022. R package version 1.0.4.

[21] F. McGee, M. Ghoniem, G. Melançon, B. Otjacques, and B. Pinaud. The state of the art in multilayer network visualization. *Computer Graphics Forum*, 38(6):125–149, 2019.

[22] B. Selman and H. J. Levesque. The hard and easy distribution of SAT problems. In *Proceedings of the 10th AAAI*, pages 440–446, 1992.

[23] R. Tamassia. *Handbook of graph drawing and visualization*. CRC press, 2013.

[24] The On-Line Encyclopedia of Integer Sequences. Sequence A003024: Number of directed acyclic graphs (DAGs) with n labeled nodes, 2024. Accessed: 2024-03-20.

[25] A. Wigderson. The complexity of the Hamiltonian circuit problem for maximal planar graphs. *EECS Department Report*, 298, 1982.

Appendix

The appendix provides supplementary details and analyses that support and extend the main findings of our study on the k-page upward book embedding problem (*kUBE*) for directed DAGs. It includes in Section A a detailed breakdown of the number of DAGs for small vertex counts ($n \leq 6$) across varying edge counts, as enumerated using Algorithm 1, and in Section B an in-depth exploration of phase transitions in embeddability. These transitions reveal critical thresholds in graph density (m/n) where embeddability shifts sharply, offering insights into the structural and computational complexities of *kUBE*.

A Numbers of DAGs with $n <= 6$

According to the On-Line Encyclopedia of Integer Sequences ([24, A003024]), the total number of DAGs is 25 for $n = 3$, 543 for $n = 4$, 29,281 for $n = 5$, and 3,781,503 for $n = 6$. In fact, we employ Algorithm 1 to determine their detailed breakdown across different edge counts (m), with the results presented in Tables 2, 3 and 4.

Table 2: Number of DAGs with (a) $n = 3$, and (b) $n = 4$ nodes, for different edge counts

(a) $n = 3$

Edges	0	1	2	3
# of DAGs	**1**	**6**	**12**	**6**

(b) $n = 4$

Edges	0	1	2	3	4	5	6
# of DAGs	**1**	**12**	**60**	**152**	**186**	**108**	**24**

Table 3: Number of DAGs with $n = 5$ nodes for different edge counts

Edges	0	1	2	3	4	5	6	7	8	9	10
# of DAGs	**1**	**20**	**180**	**940**	**3050**	**6180**	**7960**	**6540**	**3330**	**960**	**120**

Table 4: Number of DAGs with $n = 6$ nodes for different edge counts

Edges	0	1	2	3	4	5	6	7
# of DAGs	**1**	**30**	**420**	**3600**	**20790**	**83952**	**240480**	**496680**
Edges	8	9	10	11	12	13	14	15
# of DAGs	**750810**	**838130**	**691020**	**416160**	**178230**	**51480**	**9000**	**720**

B Phase Transition

From our study of DAGs—whether enumerating smaller instances or analyzing larger sampled ones with $n = 10$, 15, and 20—a clear phenomenon emerges: the presence of a phase transition in embeddability. In simple terms, a phase transition refers to a sudden shift in a system's behavior as a key parameter changes, much like water turning to ice as temperature drops; here, it manifests as a rapid drop from nearly all DAGs being embeddable to nearly none as the edge count m crosses a critical threshold.

To investigate this, one examines how a property (like embeddability) changes with a control parameter (such as m/n, or graph density), looking for a sharp transition rather than a gradual one, often marked by a specific point where the outcome flips dramatically, for example, where embeddability falls from 100% to near 0%. Observing this behavior in our previous results, we now delve into a detailed study of the phase transition, specifically using graph density (m/n) as the control variable, to characterize how embeddability drops sharply as m/n increases, indicating a rapid shift similar to a phase transition, while noting that the mean runtime peaks around this steep decline, across varying n and k.

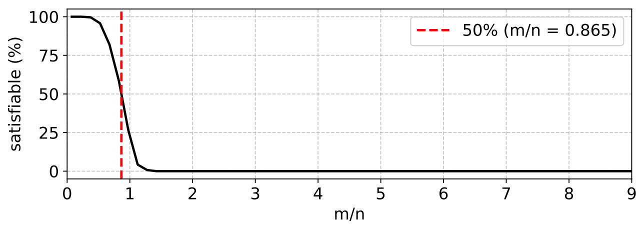
We used the 1,277 DAGs from the North Graph dataset [12] along with 45,000 DAGs generated through the sampling methods described in previous sections, analyzing their distribution in increasing graph density from $k = 1$ to 6 in terms of embeddability (Figures 4 a–f) and runtime (Figures 5 a–f).

Figure 4 in particular shows the percentage of embeddable DAGs decreasing rapidly at critical m/n values: 0.865 for $k = 1$, 1.467 for $k = 2$, 2.304 for $k = 3$, 3.075 for $k = 4$, 3.930 for $k = 5$, and 5.098 for $k = 6$, as marked by the 50% embeddability thresholds. These thresholds reveal a clear relationship: the critical m/n at which the phase transition occurs increases near-linearly with k, suggesting that each additional page allows the DAG to sustain a higher density before embeddability collapses, a trend we explore further to quantify its implications across varying n.

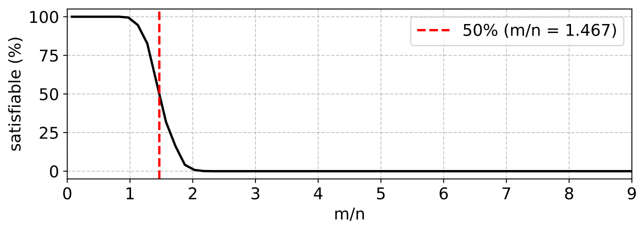
Figure 5 illustrates runtime behavior across $k = 1$ to 6 in subfigures (a–f), where each subfigure employs a scatter plot to depict runtime (in seconds, on a logarithmic scale) versus m/n for DAGs with $n = 7$ to 20, using distinct colors to differentiate node sizes and highlight trends across graph scales. For $k = 2$ to 6, the scatter plots reveal a pronounced peak in runtime that aligns closely with the critical m/n values from Figure 4—1.467 for $k = 2$, 2.304 for $k = 3$, 3.075 for $k = 4$, 3.930 for $k = 5$, and 5.098 for $k = 6$—where the phase transition occurs, underscoring the solver's peak complexity during the embeddability shift. Notably, the right side of each peak exhibits higher runtimes, as the increasing m/n corresponds to a larger number of edges in the graph, thereby requiring more time to verify embeddability.

The runtime behavior for $k = 1$ in Figure 5 differs markedly from cases where $k \geq 2$, due to differences in computational complexity. For $k \geq 2$, the NP-completeness of kUBE leads to runtime peaks at the phase transition (e.g., $m/n = 1.467$ for $k = 2$, increasing to 5.098 for $k = 6$), as shown in subfigures (b–f). In contrast, for $k = 1$, its polynomial-time solvability results in a gradual runtime increase without a peak at $m/n = 0.865$ in subfigure (a). This contrast underscores that when only one page is involved, a simpler decision process suffices, avoiding the exponential complexity spike that arises with multiple pages.

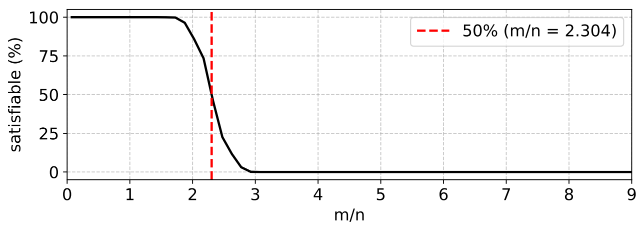
This behavior mirrors the distinction between 2SAT and 3SAT. While 3SAT is NP-complete [16] and exhibits a pronounced computational peak near its phase transition [6,22], 2SAT is solvable in linear time [9, 18] (e.g., via the implication graph and strongly connected components). As the clause-to-variable ratio in 2SAT grows, contradictions emerge more quickly, allowing for early solver termination. The lack of a runtime peak for the $k = 1$ case similarly reflects its polynomial solvability: contradictions or valid embeddings become apparent and are resolved efficiently in denser instances.



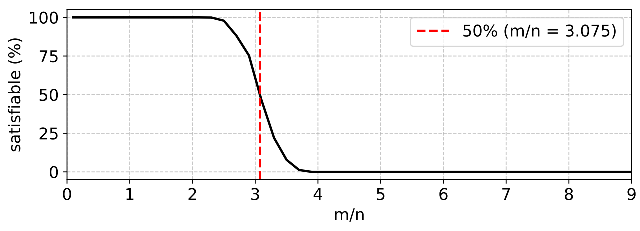
(a) $k = 1$



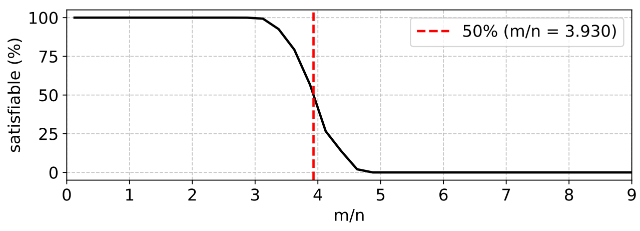
(b) $k = 2$



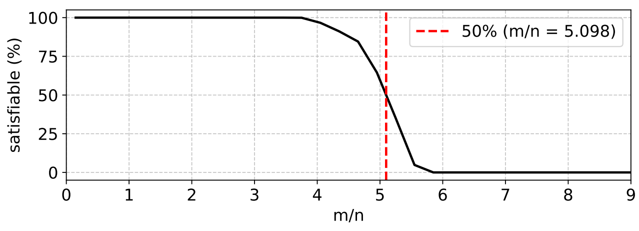
(c) $k = 3$



(d) $k = 4$

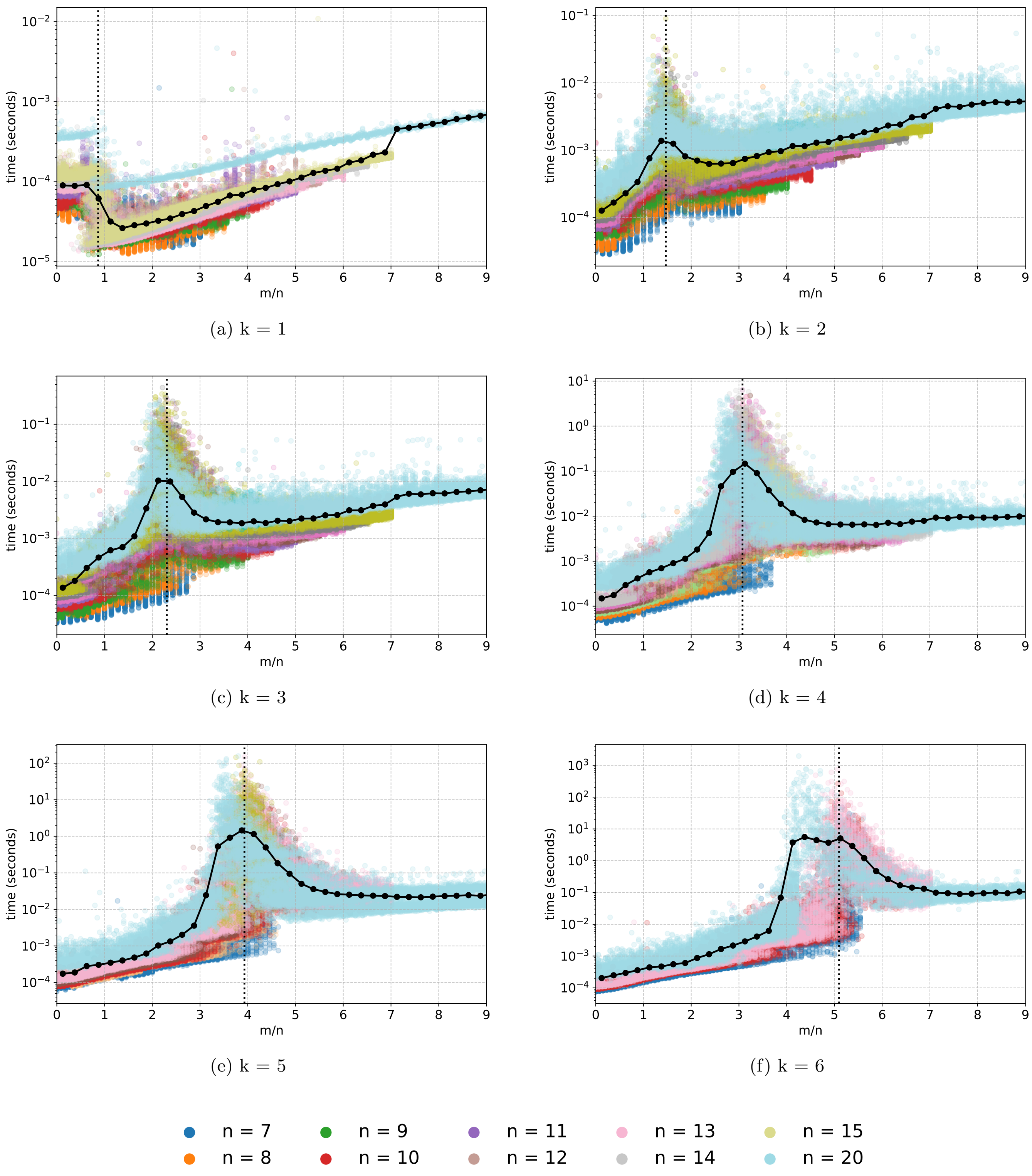


(e) $k = 5$



(f) $k = 6$

Figure 4: Percentage of DAGs embeddable in a k-page upward book embedding as a function of graph density (m/n). Vertical, red and dashed lines mark the 50% embeddability thresholds.



(a) k = 1

(b) k = 2

(c) k = 3

(d) k = 4

(e) k = 5

(f) k = 6

Figure 5: Run-time versus m/n for $k \in \{1, 2, \ldots, 6\}$. Each subplot presents a scatter plot of all combined instances for a fixed k, with points colored according to the value of n. The black line denotes the mean run-time per bin of m/n, and the vertical dotted line indicates the 50% embedding threshold.

Sweeping x-monotone pseudolines[*]

Therese Biedl[†] Erin Chambers[‡] Irina Kostitsyna[§] Günter Rote[¶]

1 Introduction

Consider an arrangement $\mathcal{A}$ of n x-monotone infinite curves where each pair of curves crosses exactly once. These define a directed acyclic planar graph $G_{\mathcal{A}}$, by replacing each crossing with a new vertex, adding two vertices s, t at negative and positive infinity, and directing edges left-to-right. This paper concerns the problem of sweeping the arrangement with a rope of short length, or equivalently, sweeping $G_{\mathcal{A}}$ with a sequence of short st-paths. Formally, we start with a rope at the lower hull of the arrangement. At each step, whenever the rope contains the bottom chain of an inner face F, we may *flip* across F by replacing the bottom chain by the top chain of F. We stop when the rope is the upper hull. The *rope-length* of such a sweep is the maximum length of the rope, measured as the number of edges in the graph. See Figure 1.

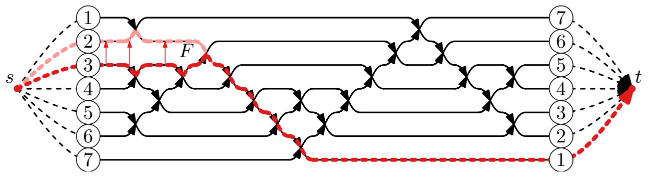


Figure 1: A pseudoline arrangement $\mathcal{A}$ with x-monotone curves and the corresponding graph $G_{\mathcal{A}}$. Rope π (red dashed) has length 8 and can be flipped across face F.

One can easily construct an arrangement $\mathcal{A}$ where the lower hull has length n, so we cannot in general hope to find a sweep of rope-length less than n. But can we always achieve rope-length $n+O(1)$ with a suitable sweep? We show that this is false: for some arrangements we need rope-length at least $\frac{7}{4}n - \frac{5}{4}$. We also provide an asymptotically matching upper bound: For any such arrangement $\mathcal{A}$, we can find a sweep with rope-length at most $2n - 2$. Furthermore, the sweep has special properties: we simultaneously sweep the dual graph $G^*_{\mathcal{A}}$ of $G_{\mathcal{A}}$, and the two ropes of the two sweeps "hug" in some sense.

Finally, we study hardness results. A rope in $G_{\mathcal{A}}$ corresponds to an edge-cut in $G^*_{\mathcal{A}}$, and sweeping with a rope hence corresponds to finding a vertex order that has small cuts. This is the *cutwidth* problem, and since we impose special conditions on the graph and the sweep, our problem is equivalent to solving Directed Cutwidth in $G^*_{\mathcal{A}}$ (definitions and details are in Section 5). Surprisingly enough, we have not been able to find NP-hardness results for this problem, especially not in planar graphs. We therefore show that Directed Cutwidth is NP-hard even in planar graphs with maximum degree 6. Unfortunately the graphs constructed in the reduction are not duals of pseudoline arrangements, so the complexity of minimizing the rope-length in our sweeping problem remains open.

Related results: The problem of minimizing the rope-length of a sweep is motivated by the problem of enumerating all arrangements of n pseudolines [14]. An easy upper bound on the rope-length in a sweep is the maximum length of an x-monotone st-path. However, this does not lead to a good upper bound: x-monotone paths can have close to n^2 edges [2, 11], see [10] for related results. This shows that it is necessary to choose a sweep carefully.

The idea of "sweeping a plane graph" is closely related to the so-called *homotopy height*, see [3, 7, 13] for an overview. Here we are given an undirected planar graph G with a fixed planar embedding and two vertices s, t on the outer-face. We are asked to find a sequence of st-paths that begin and end with the two st-paths that run along the outer-face. Consecutive st-paths in the sequence must be related via a limited set of *permitted operations*, which include flipping across a face and introducing or eliminating a spike along an edge. The goal is to minimize the maximum path-length in the sequence. Our problem is hence the same as computing the homotopy height, except that we restrict the set of permitted operations since the path must follow the edge directions.

Computing the homotopy height of a graph is in NP

[*]Research initiated during the 2020 Workshop on Graphs and Geometry at the Bellairs Research Institute. The authors would like to thank all participants of the workshop, but especially Hugo Akitaya and Stefan Felsner, for helpful input.

[†]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada. Supported by NSERC.

[‡]Department of Computer Science and Engineering, University of Notre Dame, Indiana, USA. Research supported in part by NSF awards 1907612 and 2444309.

[§]KBR at NASA Ames Research Center, Moffett Field, CA, USA.

[¶]Freie Universität Berlin, Institut für Informatik, Germany.

[7], but it remains open whether this problem is NP-hard. There is also a relationship between the homotopy height and the height of a planar straight-line grid-drawing [3]; in particular every N-vertex planar graph G has homotopy height at most $\frac{2}{3}N + O(1)$ since G has a planar straight-line grid-drawing where the smaller dimension is $\frac{2}{3}N + O(1)$ [8]. Unfortunately, this does not help to solve our problem, for two reasons. First, in our sweeps we impose stronger restrictions on when we are allowed to flip across a face. Second, we are sweeping an arrangement of n curves, hence the corresponding planar graph has $N \in \Theta(n^2)$ vertices and the above bounds are meaninglessly big.

As mentioned earlier, sweeping a pseudoline arrangement $\mathcal{A}$ with a short rope corresponds to solving Directed Cutwidth in the dual graph $G^*_{\mathcal{A}}$. The (undirected) version Cutwidth of this problem is very well-established in the literature and is known to be NP-hard even in planar graphs with maximum degree 3 [12]. Cutwidth is also SSE-hard to approximate within any constant factor [15]. SSE stands for the *Small Set Expansion conjecture*; we refer to this paper for the definition of "SSE-hard" and other results concerning cutwidth.

2 Definitions

Throughout the paper, $\mathcal{A}$ denotes a set of n x-monotone infinite curves that form a *pseudoline arrangement*, i.e., each pair of curves has exactly one point in common where the curves properly cross. The curves in $\mathcal{A}$ are called *pseudolines*. Arrangement $\mathcal{A}$ naturally defines a planar graph $G_{\mathcal{A}}$, by replacing every crossing between pseudolines by a vertex, adding an edge whenever two crossings are consecutive on a pseudo-line, adding two vertices s and t that represent the points at negative and positive infinity, and connecting s to the first crossing and t to the last crossing of each pseudo-line. We direct all edges of $G_{\mathcal{A}}$ from left to right, making it a directed acyclic planar graph with exactly one source s and one sink t that are both on the outer-face. Such a graph is known as a *bipolar orientation*, and many properties are known, see for example [9]. In particular, for any inner face F, the boundary consists of two directed paths; in our situation where edges are drawn left-to-right these paths naturally are called the *top chain* and *bottom chain* of F. Their common start-vertex is the *source* $s(F)$ of F, and their common end-vertex is the *sink* $t(F)$ of F. At any vertex $v \neq s, t$, the incoming edges are consecutive in the clockwise order around v, as are the outgoing edges. In our situation with edges drawn left-to-right, we can naturally speak of the topmost/bottommost incoming/outgoing edge of a vertex.

A *rope* of $\mathcal{A}$ is a directed st-path π in $G_{\mathcal{A}}$; alternatively we can view π as an x-monotone infinite curve along pseudo-lines. For any two points p, p' on π, we use $\pi(p, p')$ to denote the sub-curve between the two points (including p, p'). If π contains the entire bottom chain of some inner face F, then *flipping rope* π *across* F means to create a new rope that is π except that the bottom chain $\pi(s(F), t(F))$ of F gets replaced by the top chain of F. A *sweep* of $\mathcal{A}$ consists of a sequence $\pi_1, \dots, \pi_k$ of ropes where π_1 is the lower hull of $\mathcal{A}$, π_k is the upper hull of $\mathcal{A}$, and consecutive ropes are obtained by flipping across an inner face. The *rope-length* of such a sweep is the maximum length (measured by the number of edges) among the used ropes, and the problem studied in this paper is to find a sweep that has small rope-length.

Graph $G_{\mathcal{A}}$ (and generally any bipolar orientation) naturally gives rise to a dual graph $G^*_{\mathcal{A}}$ that is also a bipolar orientation as follows. Temporarily add an edge (s, t) to $G_{\mathcal{A}}$, and let s^*, t^* be the two faces incident to it, with s^* incident to the upper hull of $\mathcal{A}$. The vertices of $G^*_{\mathcal{A}}$ are now s^*, t^*, and one vertex F for each inner face of $G_{\mathcal{A}}$. For every edge $e = u \to v$ of $G_{\mathcal{A}}$, let F_ℓ and F_r be the faces that lie to the left and right when walking from u to v. (Since our edges are directed left-to-right, these faces are really above and below e, but "left"/"right" is the established term in the literature.) We add to $G^*_{\mathcal{A}}$ the *dual edge* e^* *of* e, which is $F_\ell \to F_r$. Note that e lies on the top chain of F_r and the bottom chain of F_ℓ, so in any sweep we must have swept F_r *before* we can sweep F_ℓ. We think of dual graph $G^*_{\mathcal{A}}$ as drawn such that each vertex F is placed in the corresponding face of $G_{\mathcal{A}}$, and each edge e^* crosses the edge e that it is dual to. By definition, e^* crosses e from left to right.

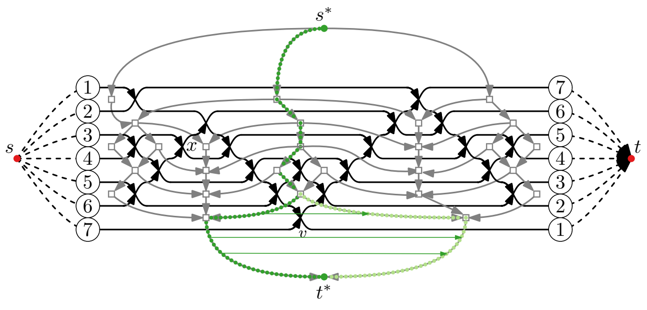


Figure 2: The dual graph $G^*_{\mathcal{A}}$ with a dual rope π^* (green dotted) that can be flipped across vertex v.

Since $G^*_{\mathcal{A}}$ is also a bipolar orientation, concepts such as "rope" and "flipping across a face" can also be applied to $G^*_{\mathcal{A}}$. For ease of distinction, we use the term *dual rope* for a rope in $G^*_{\mathcal{A}}$, and *flipping across a vertex (of* $G_{\mathcal{A}}$*)* for the operation of flipping across a face of $G^*_{\mathcal{A}}$. Note that any dual rope π^* defines an st-cut by virtue of taking the edges of $G_{\mathcal{A}}$ that it *crossed* (i.e., whose duals it contained), and symmetrically every rope

π defines an s^*t^*-cut. Both these cuts are *directed*, i.e., contain only edges directed from the source-side to the sink-side.

3 A lower bound

Theorem 1 *For $n = 3 \bmod 4$, there exists a pseudoline arrangement $\mathcal{A}$ of n x-monotone curves such that any sweep requires rope-length at least $\frac{7}{4}n - \frac{5}{4}$.*

Proof. The construction is symmetric, and we describe it from left to right, see Figure 3 for the construction for $n = 7$ and Figure 7 (in the appendix) for $n = 15$. Start with two curves c, c' (black solid) that are at the top and bottom at the far left and intersect in some point x. All other curves will pass above x. Set $K = \frac{n-3}{4}$. Between c and c' at the far left are $2K + 1$ "top" curves (red, dashed) at even positions, and $2K$ "bottom" curves (blue, dotted) at odd positions.

In the beginning, the red curves move up and the blue curves move down until they are separated, forming a $2K \times 2K$ half-grid (shown shaded in Figure 3). So far there are no intersections between curves of the same color. In the area below all red curves and above all blue curves, there are three faces F_ℓ, F_c, and F_r, separated from each other by c and c'.

Before the $2K+1$ red curves cross c, we let the lower $K+1$ of them cross each other in such a way that they all become incident to the top chain of F_ℓ. These curves, together with c, hence create a $(K+1) \times (K+1)$ half-grid. (In terms of sorting networks, this half-grid is the *bubble-sort* network.)

In the middle, in the area above x, we do two things: a) We cross the blue curves in such a way that they all become incident to the bottom chain of F_c, forming a $(2K+1) \times (2K+1)$ half-grid together with c and c'. b) We cross the upper K with the lower K red curves (the middle red curve remains uncrossed, as it meets all other red curves in the half-grids above F_ℓ and F_r).

The right part of the construction is symmetric. As shown in Figure 8 in the appendix, this arrangement can even be drawn with straight lines. Observe the following properties of x-monotone paths in the construction:

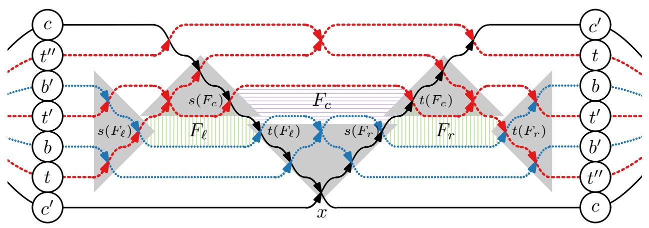


Figure 3: Construction for the lower bound for $n = 7$ (so $K = 1$); we need rope-length 11.

- Any x-monotone path from s to the source $s(F_\ell)$ of F_ℓ has length at least $2K$. This holds because such a path must traverse the $2K \times 2K$ half-grid, plus the edge from s to reach the half-grid.
- Any x-monotone path π from $t(F_c)$ to t has length at least $2K + 1$. This is obvious if π walks along c' until the intersection with the last red curve (and from there to t). So assume that it walks along c' for $i < 2K$ edges and then turns onto a red curve that brings us (perhaps after some more edges) to the half-grid right of $t(F_r)$. It then traverses a $(2K-i) \times (2K-i)$ half-grid, which takes $2K-i$ edges, plus one more edge to t. Hence the path has length at least $2K+1$.
- Any x-monotone path from $t(F_\ell)$ to $s(F_r)$ has length at least $2K$, because it must go across the $(2K+1) \times (2K+1)$ half-grid below F_c and can (at best) use shortcuts along the bottom chain of F_c.

Now we come to the actual proof. Consider any sweep of $\mathcal{A}$. Since the dual graph has edges $F_c \to F_\ell$ and $F_c \to F_r$, we must flip across both F_ℓ and F_r before flipping across F_c. By symmetry we may assume that we flip across F_r first, and consider the rope π immediately after we flipped across F_ℓ. Then π goes from s to $s(F_\ell)$, from there along the top chain of F_ℓ to $t(F_\ell)$, from there to $s(F_r)$ and $t(F_c)$ (since we have flipped across F_r but not F_c yet), and from there to t. So

$$\begin{aligned} |\pi| &= |\pi(s, s(F_\ell))| + \text{length of top chain of } F_\ell \\ &\quad + |\pi(t(F_\ell), s(F_r))| + 1 + |\pi(t(F_c), t)| \\ &\geq 2K + K{+}2 + 2K + 1 + 2K{+}1 = 7K + 4 \end{aligned}$$

which is at least $7\frac{n-3}{4} + 4 = \frac{7}{4}n - \frac{5}{4}$. □

4 An upper bound: The primal-dual sweep

We now show an upper bound on the required rope-length by defining a sequence of ropes in $G_{\mathcal{A}}$ and simultaneously a sequence of dual ropes that "hug" the ropes. To define this, we first need a few other definitions and observations about a rope π and a dual rope π^* (see also Figure 4).

Rope π connects s to t, hence must go across the directed st-cut defined by π^*, and can do so only once since π is directed. It follows that exactly one edge e of π is crossed by π^*; we call e the *active edge* and let x be the point where it is crossed by π^*. This *crossing-point* x splits the rope into two parts $\pi(s, x)$ and $\pi(x, t)$, and likewise splits the dual rope into $\pi^*(s^*, x)$ and $\pi^*(x, t^*)$, and the properties that we require will depend on which part we are in.

Definition 1 *We say that a rope π and dual rope π^** hug *each other if the following four (symmetric) conditions hold: (1) for every edge e in $\pi(s, x)$, the face*

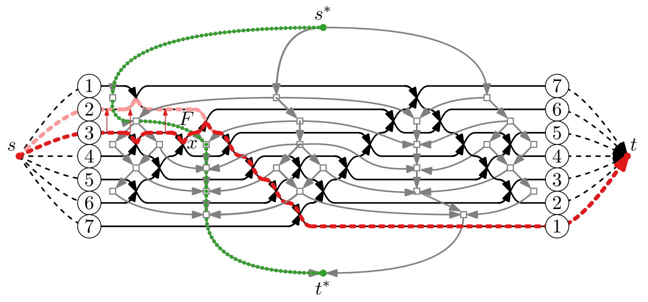


Figure 4: A rope and a dual rope that hug each other. We can flip across face F, which is to the left of the active edge.

to the left of e belongs to π^*; (2) for every edge e in $\pi(x,t)$, the face to the right of e belongs to π^*; (3) for every edge e^* in $\pi^*(s^*,x)$, the face of $G^*_{\mathcal{A}}$ (hence vertex of $G_{\mathcal{A}}$) to the right of e^* belongs to π; (4) for every edge e^* in $\pi^*(x,s^*)$, the vertex of $G_{\mathcal{A}}$ to the left of e^* belongs to π.

We will now define a sequence of *rope pairs* (i.e., pairs of a rope π and a dual rope π^*) such that the ropes sweep $G_{\mathcal{A}}$, the dual ropes sweep $G^*_{\mathcal{A}}$, and at all times π and π^* hug each other. Then we argue that this implies rope-length at most $2n-2$ at all times. We initialize rope π as the lower hull of $\mathcal{A}$, so all edges of π have t^* to their right. We initialize the dual rope π^* to contain all faces incident to s, in order from top to bottom, so all edges of π^* have s to their right. The active edge is the bottommost outgoing edge of s, and one easily verifies all conditions. (The appendix shows an example of a sweep from the beginning.) To explain how to update the rope pair, we need some observations.

Claim 1 *(1) At any vertex $v \neq s$ of $\pi(s,x)$, rope π uses the top incoming edge. (2) At any vertex $v \neq t$ of $\pi(x,t)$, rope π uses the bottom outgoing edge. (3) At any face $F \neq s^*$ of $\pi^*(s^*,x)$, dual rope π^* crosses the first edge of the top chain of F. (4) At any face $F \neq t^*$ of $\pi^*(x,t^*)$, dual rope π^* crosses the last edge of the bottom chain of F.*

Proof. We only prove the first claim, the other three are symmetric. Let e be the incoming edge of v on π, and assume for contradiction that e is not top incoming. Then the face F to the left of e is incident to two incoming edges of v, hence $v = t(F)$ and e is the last edge of the bottom chain of F. By the hugging-condition F belongs to π^*; the next edge on π^* hence crosses the bottom chain of F. But then $v = t(F)$ is on the t-side of the st-cut defined by the dual rope π^*, contradicting that $v \in \pi(s,x)$. □

Claim 2 *Let e be the active edge and let v be its head and F be the face to its left. If $F \neq s^*$ or $v \neq t$, then we can flip π across F or flip π^* across v, and the new pair of rope and dual rope hug each other.*

Proof. The claim is illustrated in Figure 5. Assume first that e is not top incoming, which implies that it is the last edge of the bottom chain of F. We know that $F \neq s^*$ since all edges incident to s^* are top incoming. Since $\pi(s,x)$ only uses top incoming edges, π must have traversed the entire bottom chain of F and by $F \neq s^*$ we can hence flip across F to get the new rope π'. The new active edge is the first edge of the top chain of F by Claim 1(3). The hugging-conditions could be violated only at face F (everywhere else the rope and dual rope are unchanged), and one easily verifies that they hold here because all new edges of π' have F to their right.

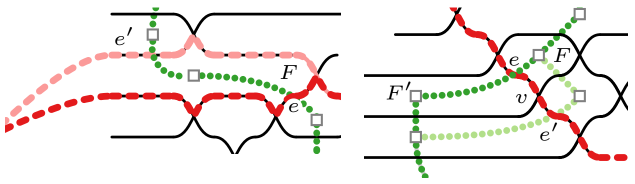


Figure 5: Closeup of flipping across a face and a vertex. Dual graph not shown.

Now assume that e is top incoming, which implies that $v \neq t$ since otherwise $F = s^*$ and not both are allowed. Let F' be the face to the right of e; this is in $\pi^*(x,t)$ since e (as active edge) is crossed by π^*. All other incoming edges of v are the last edge of the bottom chain of the faces to their left. Applying Claim 1(4) repeatedly, starting with $F' \in \pi^*(x,t^*)$, therefore dual rope π^* must cross all incoming edges of v. So by $v \neq t$ we can flip the dual rope across v. By Claim 1(2) rope π continues from v along the bottom outgoing edge, which hence becomes the new active edge. Again one easily verifies the hugging condition, since all new edges of the new dual rope have v to their right. □

We hence update π and π^* as follows. Let e be the active edge, and let v be its head and F be the face to its left. If $F = s^*$ and $v = t$ then e is the last edge of the upper hull. By Claim 1(1) hence π is the upper hull and the sweep is finished. By Claim 1(4) π^* crosses all incoming edges of t, and so the sweep of the dual is also finished. Otherwise (either $F \neq s^*$ or $v \neq t$) we perform one of the flips that exists by Claim 2 and repeat.

4.1 Analysis

The sweeping algorithm as described would actually work for any bipolar orientation. We now show that if the bipolar orientation comes from a pseudoline arrangement $\mathcal{A}$ of x-monotone curves, then the rope-length is at most $2n-2$ at all times. Enumerate the pseudolines from top to bottom in the order of incidence with s as $c_1, \ldots, c_n$. The *index* of an edge e is the index of the pseudoline that supports e, i.e., along which e runs. The following observation is trivial (it holds since pseudolines intersect only once, so one can go above the other only once), but will be crucial for counting vertices later.

Observation 1 *At any vertex $v \neq s, t$, the indices of incoming edges increase from top to bottom, while the indices of outgoing edges decrease from top to bottom.*

An *encounter* of rope π with pseudoline c_i is a maximal sub-curve $\pi(v, v')$ that belongs to c_i. Note that v, v' are necessarily vertices, and possibly $v = v'$.

Corollary 1 *While walking along $\pi(s, x)$, the index i of the current edge of π can only increase, and any pseudoline c_j encountered at the next vertex v satisfies $j \geq i$.*

Proof. Rope π enters along the top incoming edge of v, hence i is the smallest index of a pseudoline incident to v. So all pseudolines encountered at v (including the one along which π leaves) cannot have smaller index. □

Claim 3 *While walking along $\pi(s, x)$, we encounter every pseudo-line at most once.*

Proof. Assume for contradiction that we encounter pseudoline c_i at least twice. At the end of the first encounter we hence have a vertex v with $v \in c_i \cap \pi(s, x)$, but π continues beyond v along some pseudoline c_j with $j \neq i$. If $j > i$, then the index throughout $\pi(v, x)$ is at least $j > i$, and so we cannot encounter c_i again. So we must have $j < i$, which means that the outgoing edge of π at v is *below* the outgoing edge along c_i by Observation 1. Therefore c_i has entered the s^*-side of the s^*t^*-cut defined by π. Since $\pi(s, x)$ always uses top incoming edges, there are no edges from the s^*-side to $\pi(s, x)$, and so c_i cannot encounter $\pi(s, x)$ again. □

Claim 4 *At any time during the sweep, rope π has length at most $2n - 2$.*

Proof. Assign to s the pseudoline along which π leaves, and assign to every vertex $v \neq s$ on $\pi(s, x)$ the pseudoline c that supports the bottom incoming edge e at v. This assigns every pseudoline at most once, for e was *not* in $\pi(s, x)$ by Claim 1, and so v is the beginning of the unique encounter of c with $\pi(s, x)$. (This also shows that c was not assigned to s). So $\pi(s, x)$ has at most n vertices, and symmetrically $\pi(s, t)$ has at most n vertices and the rope-length is at most $2n - 1$.

We claim that this is not tight. Assume for contradiction that at some point rope π has length exactly $2n-1$, so $\pi(s, x)$ has n vertices and *all* pseudolines have been assigned to some vertex of $\pi(s, x)$. Observe that c_1 must have been assigned to s, for otherwise the index of $\pi(s, x)$ would be greater than 1 throughout, so $\pi(s, x)$ could not encounter c_1, so c_1 would not be assigned to a vertex. Also observe that c_2 must have been assigned to a vertex v that lies on c_1, because it is not assigned to s, and we assign (by Observation 1 and Claim 1(1)) a pseudoline c_j to a vertex $v \neq s$ only if $\pi(s, x)$ has index less than j when it reaches v. In particular therefore c_1 and c_2 intersect at a point on $\pi(s, x)$. By a completely symmetric argument, c_1 and c_2 intersect again at a point on $\pi(x, t)$. This is not possible in a pseudoline arrangement. □

Theorem 2 *For every pseudoline arrangement of n x-monotone curves, there exists a sweep with rope-length at most $2n - 2$.*

A few comments are in order. First, as the example shown in the appendix illustrates, the bound is tight: for some arrangements this particular method of computing a sweep requires rope-length $2n - 2$.

Also, our coordinated primal-dual sweep can be interpreted as a *left-first greedy* sweep: At each stage, the rope π selects the leftmost possible position where it can flip over a face. The dual rope π^* can be interpreted as guiding the search for the sweep position: As long as a flip is not possible at the current position of the active edge, the active edge advances to the right, and this corresponds to a dual flip. Such a left-first greedy method was used in an algorithm by Alvarez and Seidel as a tool to count the number of triangulations [1].

5 NP-hardness

In this section, we reduce our sweep-problem to solving Directed Cutwidth in $G^*_{\mathcal{A}}$. Then we show that Directed Cutwidth is NP-hard even in planar graphs with maximum degree 6. Unfortunately this does not prove the sweep-problem NP-hard since the graph that we construct cannot be the dual graph of a pseudoline arrangement (it has vertices of degree 2 and many sources and sinks).

We need a few definitions. Fix a vertex order $\sigma = \langle v_1, \ldots, v_n \rangle$ of G. For $1 \leq i \leq n$, the *ith cut* (or *cut after* v_i) is the set of edges (v_h, v_j) with $h \leq i < j$. The maximum cardinality of these cuts is the *width* of the vertex order, and the *cutwidth* of graph G is the minimum width over all vertex orders.

The cutwidth is defined for undirected graphs, but for directed acyclic graphs there exists a natural restriction, apparently first studied in [4]: The *directed cutwidth* of a directed acyclic graph G is the minimum width of a vertex order of G that is a *topological order*, i.e., where every edge is directed from a lower-indexed to a larger-indexed vertex.

Lemma 3 *Let $\mathcal{A}$ be a pseudo-line arrangement with x-monotone curves. Then $\mathcal{A}$ has a sweep with rope-length at most w if and only if $G^*_{\mathcal{A}}$ has directed cutwidth at most w.*

Proof. We only show one direction, the other is similar. Fix a sweep with rope-length w. This defines a sequence $\sigma = \langle F_1, \dots, F_k \rangle$ of the inner faces of $G_{\mathcal{A}}$ via the order in which the sweep flips the rope across faces. We append $s^* =: F_{k+1}$ and pre-pend $t^* := F_0$ to this sequence since the rope begins incident to t^* and ends incident to s^*. Sequence σ hence gives a vertex order $F_0, F_1, \dots, F_{k+1}$ of $G^*_{\mathcal{A}}$. Any directed edge $F_\ell \to F_r$ of $G^*_{\mathcal{A}}$ is dual to an edge e of $G_{\mathcal{A}}$ that is on the upper chain of F_r and the lower chain of F_ℓ. So the sweep must flip across F_r *before* flipping across F_ℓ, i.e., $r < \ell$. So in our face order all edges of $G^*_{\mathcal{A}}$ are directed right-to-left, and reversing it (which does not affect the width) gives a topological order. Finally the edges of the ith cut are dual to the edges of the rope after flipping across F_i, and vice versa. Therefore the width of the topological order is the same as the rope-length. □

So we are interested in the complexity of problem Directed Cutwidth, the decision version of the problem: Given a directed acyclic graph G and an integer w, is there a topological order of width at most w? Surprisingly, the complexity of this problem does not appear to have been studied much in the literature. Wu et al. [15] showed that Directed Cutwidth (not specifically named there, but appearing in row 6 of their Table 1) is SSE-hard to approximate (the constructed graphs are non-planar). There are also some positive results; in particular Directed Cutwidth has a linear-time algorithm if w is a constant [4], and for series-parallel graphs it can be computed in quadratic time [5]. But we have the following new result:

Theorem 4 Directed Cutwidth *is NP-hard, even in planar graphs with maximum degree 6.*

Proof. The reduction is from Cutwidth, which is known to be NP-hard, even for a planar graph with maximum degree 3 [12]. So assume that we are given a planar graph G with maximum degree 3 and an integer w and we want to test whether its cutwidth is at most w. We may assume that G has no isolated vertices or isolated edges: They do not affect the cutwidth, except in the trivial case that G consists exclusively of isolated edges and vertices. We create a directed graph H as follows (see Figure 6). We retain all vertices of G, and replace every edge $e = (v, w)$ by a source s_e and a sink t_e that are both incident to both v, w. (A similar transformation, using only a sink, was used in [15].)

We claim that G has a vertex order σ_G of width at most w if and only if H has a topological order σ_H of width at most $2w + 2$. We sketch here a proof; details

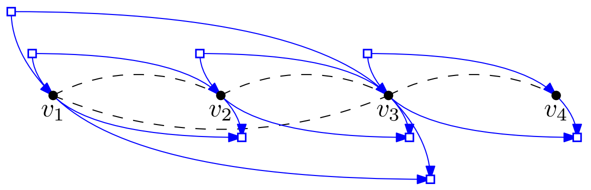


Figure 6: From a vertex order of G (black dashed) to a topological order of H (blue solid). For ease of reading we offset sources to be above and sinks to be below vertices of G.

are in the appendix. To convert σ_G to σ_H, simply add (for each edge e of G) source s_e just before the first endpoint of e in σ_G, and sink t_e just after the second endpoint of e in σ_G. Elementary arguments (using that G has maximum degree 3) show that σ_H then has width at most $2w + 2$. To convert σ_H to σ_G, initially simply take the induced vertex order, which is easily seen to have width at most $w + 1$. This can be tight (say at the ith cut) only if v_i has no neighbours on the left while v_{i+1} has no neighbours on the right. Call such a pair (v_i, v_{i+1}) *improvable*: exchanging the two vertices in the order improves the size of the ith cut and leaves all other cuts after vertices unchanged. Exchanging all improvable pairs hence gives the desired σ_G. □

6 Summary and outlook

In this paper, we studied the problem of sweeping a pseudoline arrangement of n x-monotone curves using a rope between the points of infinity. The only permitted move is to flip parts of the rope from the bottom chain to the top chain of a face, and the goal is to keep the number of edges on the rope small. We argue that the worst-case rope-length is in $\Theta(n)$, and specifically, at most $2n - 2$ (for all arrangements) and at least $\frac{7}{4}n - \frac{5}{4}$ (for some arrangements).

The most tantalizing open problem is the complexity of finding the shortest rope, possibly for an arbitrary bipolar orientation instead of a pseudoline arrangement. We proved NP-hardness of Directed-Cutwidth, which is closely related to our problem via duality. But the graph that we construct for the NP-hardness ihas many sources and sinks, and so is not the dual graph of a pseudoline arrangement, and proving NP-hardness of the original problem or finding a polynomial-time algorithm for it remains open.

Our sweep by definition is monotone in the sense that every inner face is swept exactly once. Could a shorter rope-length ever be achieved if we are permitted to reverse some flips? We suspect that (as for the homotopy height under some restrictions on the input [6]) repeatedly sweeping a face cannot shorten the rope-length, but this remains open.

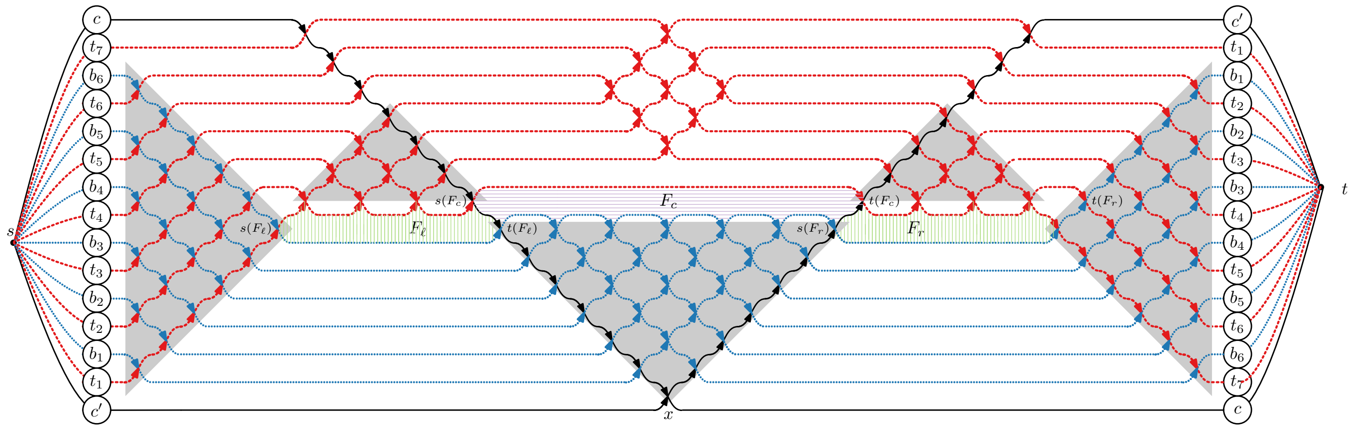


Figure 7: The lower-bound construction for $n = 15$ pseudolines ($K = 3$); we need rope-length 25.

References

[1] Victor Alvarez and Raimund Seidel. A simple aggregative algorithm for counting triangulations of planar point sets and related problems. In Guilherme Dias da Fonseca, Thomas Lewiner, Luis Mariano Peñaranda, Timothy M. Chan, and Rolf Klein, editors, *Symposium on Computational Geometry 2013, SoCG '13, Rio de Janeiro, Brazil, June 17–20, 2013*, pages 1–8. ACM, 2013.

[2] József Balogh, Oded Regev, Clifford D. Smyth, William L. Steiger, and Mario Szegedy. Long monotone paths in line arrangements. *Discret. Comput. Geom.*, 32(2):167–176, 2004.

[3] T. Biedl, E. Chambers, D. Eppstein, A. de Mesmay, and T. Ophelders. Homotopy height, grid-major height and graph-drawing height. In Daniel Archambault and Csaba D. Tóth, editors, *Graph Drawing and Network Visualization (GD 2019)*, volume 11904 of *Lecture Notes in Computer Science*, pages 468–481. Springer, 2019.

[4] Hans L. Bodlaender, Michael R. Fellows, and Dimitrios M. Thilikos. Derivation of algorithms for cutwidth and related graph layout parameters. *J. Comput. Syst. Sci.*, 75(4):231–244, 2009.

[5] Hans L. Bodlaender, Lars Jaffke, and Jan Arne Telle. Typical sequences revisited - computing width parameters of graphs. *Theory Comput. Syst.*, 67(1):52–88, 2023.

[6] Erin Wolf Chambers, Gregory R. Chambers, Arnaud de Mesmay, Tim Ophelders, and Regina Rotman. Constructing monotone homotopies and sweepouts. *Journal of Differential Geometry*, 119(3):383–401, 2021.

[7] Erin Wolf Chambers, Arnaud de Mesmay, and Tim Ophelders. On the complexity of optimal homotopies. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1121–1134. SIAM, 2018.

[8] M. Chrobak and S. Nakano. Minimum-width grid drawings of plane graphs. *Comput. Geom.*, 11(1):29–54, 1998.

[9] Hubert de Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl. Bipolar orientations revisited. *Discret. Appl. Math.*, 56(2-3):157–179, 1995.

[10] Adrian Dumitrescu. On some monotone path problems in line arrangements. *Comput. Geom.*, 32(1):13–25, 2005.

[11] Jirí Matousek. Lower bounds on the length of monotone paths in arrangement. *Discret. Comput. Geom.*, 6:129–134, 1991.

[12] B. Monien and I.H. Sudborough. Min cut is NP-complete for edge weighted trees. *Theoretical Computer Science*, 58(1–3):209–229, 1988.

[13] Tim Ophelders and Salman Parsa. Minimum height drawings of ordered trees in polynomial time: Homotopy height of tree duals. In Xavier Goaoc and Michael Kerber, editors, *38th International Symposium on Computational Geometry (SoCG 2022)*, pages 55:1–55:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

[14] Günter Rote. NumPSLA – an experimental research tool for pseudoline arrangements and order types. In Jan Kratochvíl and Giuseppe Liotta, editors, *41st European Workshop on Computational*

Geometry (EuroCG 2025), pages 18:1–18:8, April 2025.

[15] Yu Wu, Per Austrin, Toniann Pitassi, and David Liu. Inapproximability of treewidth and related problems. *J. Artif. Intell. Res.*, 49:569–600, 2014.

A More details on lower bounds

Figure 7 shows the lower bound example for $n = 15$. Figure 8 shows that it can be realized even as a line arrangement.

The greedy algorithm will actually achieve ropelength $(7n-5)/4 = 7K+4$ in these instances.

B More details on "Upper bounds"

We illustrate another example of how the sweep is performed in the following sequence of figures. The construction consists of n pseudolines $c_1, \ldots, c_n$, enumerated in top-to-bottom order at s, that satisfy the following:

- For any $i > 1$, the first crossing along c_i is with pseudoline c_1.
- Let F be the face to the left of the last edge of c_1. Then the top chain of F meets all pseudolines except c_1.

See Figure 9 for the pseudoline arrangement (for $n = 7$) and the initial rope and dual rope.

We show that if these conditions hold, then the rope-length becomes $2n-2$ at some point (hence the bound of Claim 4 is tight). To see this, observe that the first move is to flip across a face, since the active edge (which is the first edge of pseudoline c_n) is bottom incoming. See Figure 10.

The next few moves will *all* be face-flips, because the active edge is always the first edge of pseudoline c_i for some $i > 1$, which is bottom incoming because it ends at the intersection with c_1. So we continue face-flips until the active edge is the first edge of c_1, and in fact the entire rope is exactly c_1. See Figure 11.

Now the active edge is on c_1, hence top incoming, and we do a vertex-flip, which pushes the active edge one further down the rope (i.e., along c_1). See Figure 12.

The next few moves will actually *all* be vertex-flips, because the active edge is always on c_1, hence top-incoming if its head is not t. So we continue doing vertex-flips until the active edge is the last edge of c_1. See Figure 13.

Now the active edge is bottommost incoming at its head t, which means that we do a face-flip at the face F to the left of the active edge. Recall that we constructed our arrangement so that the upper chain of this face F has length $n-1$. Also, pseudoline c_1 has n edges, of which the rope uses all but the last one. Therefore at this point the rope has length $2n-2$. See Figure 14.

We note that rope-length $2n-2$ is not required in this example if we sweep differently. In particular, a sweep with rope-length $n+1$ can be obtained in this example by applying the algorithm to the reflected arrangement in which left and right are swapped.

C More details on "NP-hardness"

In this section, we fill in the details of the NP-hardness proof of Section 5. Recall given a graph G, we created the directed acyclic graph H by replacing every edge e of G by a source s_e and a sink t_e that both are adjacent to both endpoints of e. This doubles the degrees of all vertices of G (so the maximum degree of H is 6). Also the undirected version of H can be obtained by duplicating all edges of G and then subdividing all edges; in particular if G is planar then so is H.

To argue the bounds on the widths, we need some notation. For any vertex order $v_1, \ldots, v_n$ of G, and any $i = 1, \ldots, n$, write L_i $[R_i]$ for the set of edges in G that are incident to v_i and whose other endpoint is left [right] of v_i in the vertex order. Also, let B_i be the set of edges that *bypass* v_i, i.e., have the form (v_h, v_j) for $h < i < j$, and note that the cut before and after v_i have size $|B_i| + |L_i|$ and $|B_i| + |R_i|$, respectively.

For both G and H, we write $C^{\leftarrow}(v)$ and $C^{\rightarrow}(v)$ for the cuts directly before and after a vertex v, respectively, and indicate with a subscript which graph this applies to. (The vertex order will be clear from context.)

Claim 5 *If G has a vertex order $v_1, \ldots, v_n$ of width w, then H has a topological order σ_H of width at most $2w+2$.*

Proof. As sketched earlier, σ_H is obtained by inserting, for each edge e, the source just before the left end of e and the sink just after the right end of e. Put differently, for $i = 1, \ldots, n$, list all sources of edges of R_i (in arbitrary order), then list v_i, the list all sinks of edges in L_i and proceed to the next i. See Figure 6 for an example, and verify that we indeed obtain a topological order. Also notice that scanning σ_H from left to right, the cut-sizes increase when we pass a source and decrease when we pass a sink, so the maximize cut-size of σ_H must occur immediately before or after some original vertex v_i of G.

One verifies that $C^{\leftarrow}_H(v_i)$ contains exactly two edges each for each edge in $L_i \cup B_i \cup R_i$, due to edges in $C^{\leftarrow}_G(v_i)$ and sources for edges in R_i, respectively. Therefore $|C^{\leftarrow}_H(v_i)| = 2(|L_i| + |B_i| + |R_i|)$, and by symmetry, this is also equal to $|C^{\rightarrow}_H(v_i)|$. Since

$$|B_i| + \max\{|L_i|, |R_i|\} = \max\{|C^{\leftarrow}_G(v_i)|, |C^{\rightarrow}_G(v_i)|\} \leq w,$$

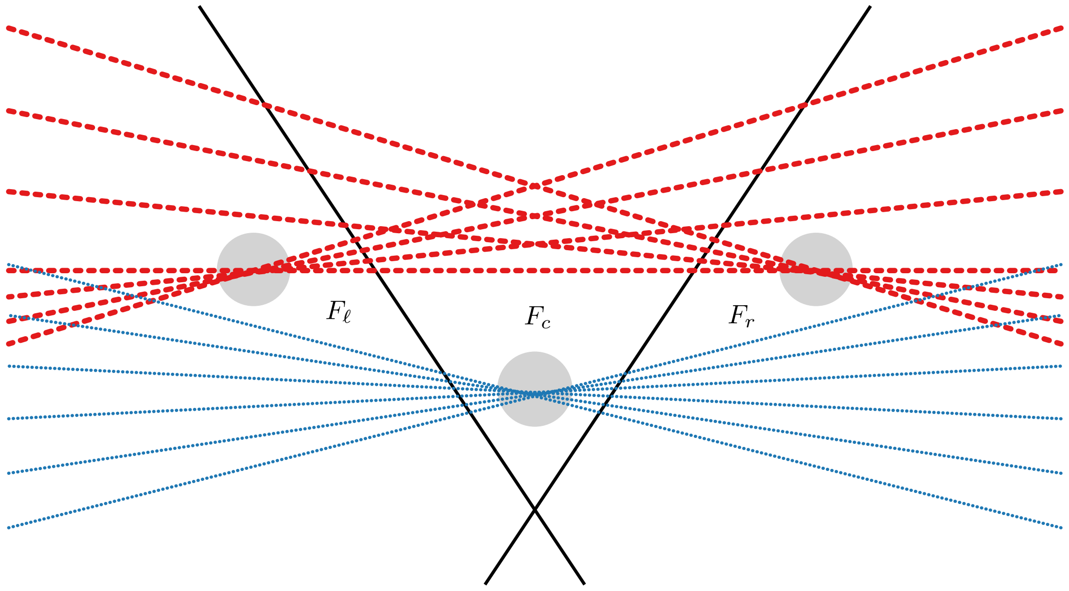


Figure 8: The lower-bound example as an arrangement of straight lines. The slopes of the seven red (dashed) and six blue (dotted) lines are evenly spaced, with red and blue slopes interleaving. This ensures the appropriate intersection pattern when the lines are extended far enough to the left and right. In the three shaded disks, the lines are slightly perturbed from a common intersection point so that they become incident to F_ℓ, F_c, and F_r, respectively.

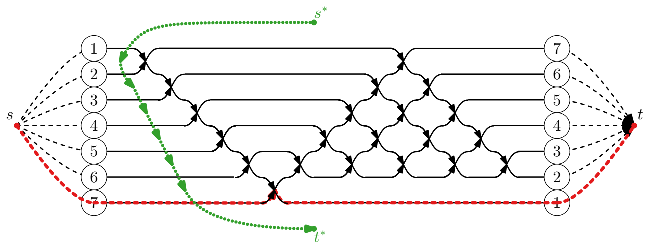


Figure 9: The arrangement, with initial rope and dual rope.

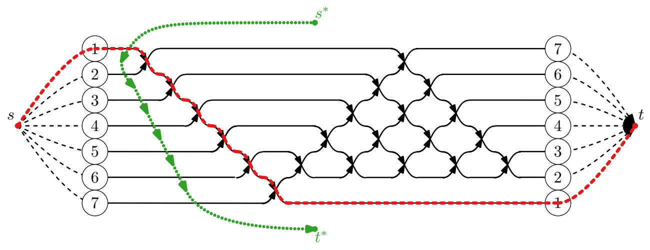


Figure 11: The situation after repeated face-flips until the rope follows c_1.

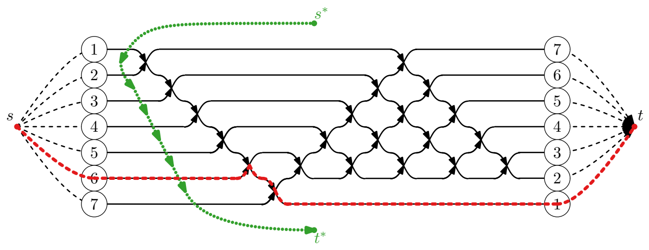


Figure 10: The situation after the first face-flip.

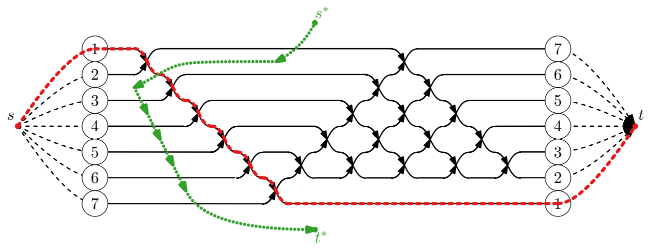


Figure 12: The situation after the first vertex-flip.

the width of σ_H is at most $2(|B_i| + \max\{|L_i|, |R_i|\} + \min\{|L_i|, |R_i|\}) \leq 2w + 2\min\{|L_i|, |R_i|\} \leq 2w + 2$ since $|L_i| + |R_i| \leq \deg_G(v_i) \leq 3$. □

For the other direction, we must convert a topological order of H into a vertex order of G of small width. Recall that in a vertex order of G, the pair (v_i, v_{i+1}) (for some $1 \leq i < n$) is called an *improvable pair* if $L_i = \emptyset = R_{i+1}$, see also Figure 15.

Claim 6 *If H has a topological order σ_H of width $2w+2$, then in the induced vertex order $v_1, \ldots, v_n$ of G, the*

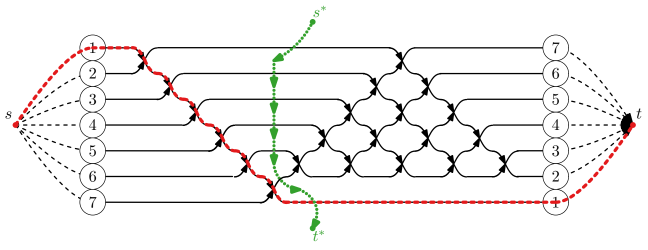


Figure 13: The situation after repeated vertex-flips until the active edge is the last edge of c_1.

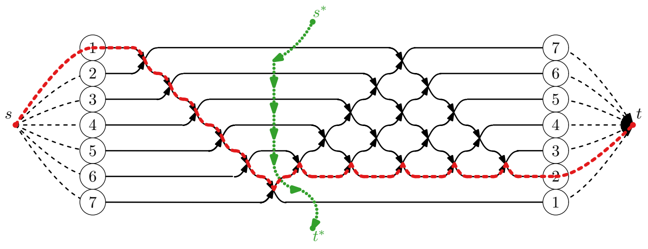


Figure 14: After one more face-flip, the rope length is $2n-2$.

ith cut has width at most $w+1$ for all $i<n$, and equality holds only if (v_i, v_{i+1}) is an improvable pair.

Proof. We have to bound $|B_i|+|R_i|$, and will show that all these edges, *and* the edges of L_i, had contributed to $C^{\rightarrow}_H(v_i)$, so there cannot be too many of them. For $e \in L_i \cup B_i \cup R_i$, the left end was v_i or farther left, while the right end was v_i or farther right. Since σ_H is a topological order, source s_e was strictly before v_i and sink t_e was strictly after v_i in σ_H, and so in σ_H this contributed two edges to $C^{\rightarrow}_H(v_i)$. Therefore

$$2w+2 \geq |C^{\rightarrow}_H(v_i)| \geq 2|L_i| + 2|B_i| + 2|R_i|,$$

which implies that $|C^{\rightarrow}_G(v_i)| = |B_i| + |R_i| \leq w+1$ and equality can hold only if $L_i = \emptyset$. Symmetrically arguing via the cut before v_{i+1} in σ_H, one sees that

$$2w+2 \geq 2|C^{\leftarrow}_H(v_{i+1})| \geq 2(|L_{i+1}| + |B_{i+1}| + |R_{i+1}|)$$

and so $|C^{\rightarrow}_G(v_i)| = |C^{\leftarrow}_G(v_{i+1})| = |L_{i+1}| + |B_{i+1}| \leq w+1$ and equality can only hold if also $R_{i+1} = \emptyset$. □

Figure 15 shows an example where the width of the induced vertex order σ_G is indeed $w+1$. So we are not done yet with the reverse direction of the reduction. But observe that if the pair (v_i, v_{i+1}) is improvable, then by exchanging their order all edges in R_i and L_{i+1} are removed from the cut between them, except the edge $v_i v_{i+1}$ if it exists. Since we have excluded the cases that v_i or v_{i+1} are isolated vertices or $v_i v_{i+1}$ is an isolated edge, the cut strictly improves. All other cuts remain unchanged. We repeat this until no improvable pair remains. In the end, all cut-sizes are at most w as desired, and G has cutwidth at most w.

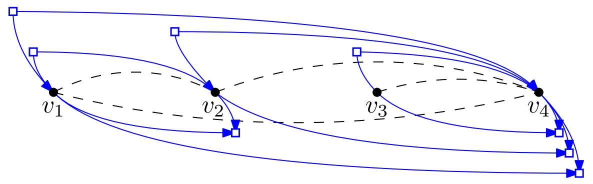


Figure 15: From a topological order of H (blue solid) of width $2w+2=6$ to a vertex order of G (dashed black), but it may not have optimal width: G has cutwidth $w=2$ (see Figure 6), but the cut between v_3 and v_4 has width 3. Note that $L_3 = \emptyset = R_4$, i.e., (v_3, v_4) is improvable.

D Experimental results

We ran some computer experiments, exhaustively trying all pseudoline arrangements with up to $n=9$ pseudolines. Each arrangement was subjected to a rather brute-force attack to find the shortest rope-length, by essentially looking for a path in the graph whose nodes represent all possible ropes. The data that we found are displayed in Table 1. For n of the form $n=4k+3$, the results on the maximum agree with the lower bound of Theorem 1.

n	min	max	#PSLA
2	2	2	1
3	4	4	2
4	5	5	8
5	6	7	62
6	7	9	908
7	8	11	24,698
8	9	12	1,232,944
9	10	14	112,018,190

Table 1: min/max: The shortest and longest rope-length required for pseudoline arrangements with n pseudolines. #PSLA: the number of combinatorial types of x-monotone pseudoline arrangements with n pseudolines (sequence A006245 in the Online Encyclopedia of Integer Sequences).

The lower bound is apparently $n+1$, except for $n=2$. The number of arrangements that require the maximum rope-length grows very quickly. For example, among the arrangements of 7 pseudolines, there are exactly two that require rope-length 11, up to symmetries. On the other hand, with 8 pseudolines, 1184 arrangements among the 1,232,944 arrangements need rope-length 12.

Straight-line Orthogonal Drawing of Complete Ternary Tree Requires $O(n^{1.032})$ Area

Hong Duc Bui*

Abstract

We prove that there is a straight-line orthogonal drawing of the complete ternary tree with n nodes in a grid with area $O(n^{1.032})$, improving the best-known bound $O(n^{1.118})$ by Ali [1]. In the special case of drawings satisfying the subtree separation property, we also prove an almost-matching lower bound $\Omega(n^{1.031})$ of this area, resolving a conjecture posed by Covella, Frati and Patrignani [3].

1 Introduction

We consider the problem of embedding a tree into a grid. Given a tree T and a grid G, an embedding maps each vertex v of T to a distinct vertex v' of G, and each edge uv of T to a polyline $u'v'$ of G, such that no two polylines intersect except at endpoints.

We call such an embedding *orthogonal*, because all grid lines are either horizontal or vertical. If, furthermore, the embedding satisfies that every edge uv gets mapped to either all horizontal segments or all vertical segments, then we call such an embedding *straight-line orthogonal*. In the literature, straight-line orthogonal embeddings in a grid are also called straight-line orthogonal *drawings*.

Determining whether a straight-line orthogonal drawing of a tree exists is simple—each node of the grid has degree at most 4, so a necessary condition is that each node of the tree has degree at most 4, and it can be shown that this condition is also sufficient. As such, most research on drawings of trees is concerned with minimizing the *area* of the grid, where the area is defined as the number of nodes of the grid.

There has been much research on this problem, see Table 1 for a summary. If the drawing is not required to be straight-line (only orthogonal), [6] proves that there exists an embedding with area $O(n)$ for all embeddable trees, where n is the number of nodes in the tree—this is asymptotically optimal, because the area must be at least n. When the drawing is required to be straight-line, the complete binary tree can be drawn in $O(n)$ area by [4], and it is proven in [2] that any binary tree can be drawn in $n \cdot 2^{O(\log^* n)}$ area, which is almost linear.

	Straight-line	Upper bound	Ref.
Comp. Binary	✓	$O(n)$	[4]
Binary	✓	$n \cdot 2^{O(\log^* n)}$	[2]
Comp. Ternary	✓	$O(n^{1.118})$	[1]
Ternary	✓	$O(n^{1.576})$	[3]
Any		$O(n)$	[6]

Table 1: Summary of existing works on tree drawings.

For the ternary case however, the known bounds are less tight. Prior to our work, the best known upper bound for the complete ternary tree is $O(n^{1.118})$ [1], improving upon an existing bound $O(n^{1.262})$ [5]. The best known upper bound for an arbitrary ternary tree is $O(n^{1.576})$ [3].

In this article, we study straight-line orthogonal drawings of the complete ternary tree, and improve the upper bound of the minimum area from $O(n^{1.118})$ to $O(n^{1.032})$. Our method is based on the analysis in [3] of drawings satisfying the subtree separation property. Drawings with this property are more easily analyzed.

We also improve the lower bound of the area needed in the special case of drawings satisfying the subtree separation property to $\Omega(n^{1.031})$. This is the first nontrivial lower bound on the area, with the trivial lower bound being $\Omega(n)$.

This article is organized as follows. In Section 2, we formally define the notations being used. In Section 3, we show the result of a numerical experiment that motivates the proof. In Section 4, we explain the general proof strategy, and prove a weaker upper bound $O(n^{1.051})$ for demonstration. In Section 5, we use a very similar proof strategy to prove the lower bound $\Omega(n^{1.031})$. Finally, in Section 6, we describe our numerical algorithm to provide a certificate of the upper bound $O(n^{1.032})$.

2 Definitions

We define the notation for the complete ternary tree following [3].

Definition 1 *For each positive integer l, let T_l be the rooted complete ternary tree with l layers—that is, each non-leaf node has exactly 3 children, and each root-to-leaf path has exactly l nodes.*

*buihd@u.nus.edu

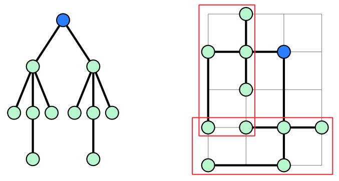


Figure 1: Example of a straight-line orthogonal drawing that does not satisfy the subtree separation property. The tree (left panel, root marked in blue) is embedded in a 4×5 grid (right panel), and the two subtrees rooted at the two children of the root have intersecting bounding rectangles.

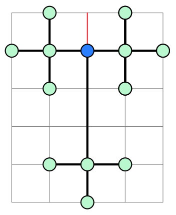


Figure 2: Illustration for $T_3 \subseteq (5, 6)$.

With this definition, T_1 has 1 node, T_2 has 4 nodes, etc.

We have defined straight-line orthogonal drawings of a tree in the introduction. Now we will formally define the subtree separation property.

Definition 2 *A drawing is said to satisfy the subtree separation property if, for every nodes a and b of the tree such that the two subtrees rooted at a and b have no nodes in common, the smallest axis-aligned bounding rectangles in the drawing containing all the nodes of these two subtrees have no grid nodes in common.*

See Figure 1 for an illustration of a drawing that does not satisfy the subtree separation property.

We define the following notation for convenience.

Definition 3 *Given positive integers l, w and h, where w is odd, write $T_l \subseteq (w, h)$ if there is an orthogonal straight-line drawing of T_l in a grid with width w and height h such that: first, the subtree separation property is satisfied; second, the root of the tree is on the middle vertical grid line; and third, the vertical ray from the root to the top of the grid does not intersect any tree nodes or edges.*

See Figure 2 for an illustration that $T_3 \subseteq (5, 6)$. The red segment in the figure marks the vertical ray from the root to the top of the grid. In order for the drawing to satisfy the third condition of the definition above, no nodes or edges can intersect this red segment.

We define a special class of constructions as follows, which has the advantage of being very easy to analyze. This is a slightly modified form of a 1-2 drawing in [3, Section 3].

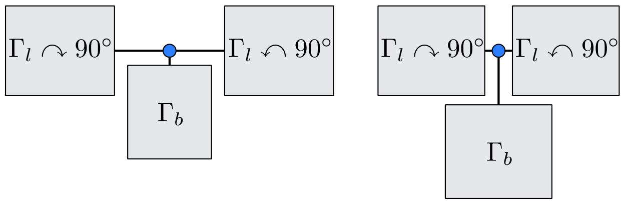

$\Gamma_l \curvearrowright 90°$
$\Gamma_l \curvearrowleft 90°$
Γ_b
$\Gamma_l \curvearrowright 90°$
$\Gamma_l \curvearrowleft 90°$
Γ_b

Figure 3: Illustration of Definition 4, with the left panel illustrating construction 1 and the right panel illustrating construction 2.

It is easier to understand the following definition by looking at a picture than reading the description, see Figure 3.

Definition 4 *We call a straight-line orthogonal drawing of T_l a* symmetric 1-2 drawing *if the following conditions are satisfied. For $l = 1$, the only symmetric 1-2 drawing is the unique drawing on the 1×1 grid. For $l > 1$, let Γ_r and Γ_b be two symmetric 1-2 drawings of T_{l-1}, then:*

- *define a drawing Γ_1 created by* construction 1 *as follows: put a copy of Γ_b below the root at distance 1, put two copies of Γ_r rotated 90° to the left and right of the root at the minimum distance such that the subtree separation property is satisfied.*
- *define a drawing Γ_2 created by* construction 2 *as follows: put two copies of Γ_r rotated 90° to the left and right of the root at distance 1 from the root, then put a copy of Γ_b below the root at the minimum distance such that the subtree separation property is satisfied.*

From the definition, we get the following lemma, which also explains the name.

Lemma 5 *All symmetric 1-2 drawings have odd width, and are vertically symmetric. Furthermore, let the size of Γ_r be (w_r, h_r) and the size of Γ_b be (w_b, h_b), then the size of Γ_1 is $\mathbf{c}_1(w_r, h_r, w_b, h_b)$ and the size of Γ_2 is $\mathbf{c}_2(w_r, h_r, w_b, h_b)$.*

Where we define two functions $\mathbf{c}_1, \mathbf{c}_2 \colon \mathbb{R}^4 \to \mathbb{R}^2$ by

$$\mathbf{c}_1(w_r, h_r, w_b, h_b) = \left(2h_r + w_b, \frac{w_r}{2} + \max\left(\frac{w_r}{2}, h_b + \frac{1}{2}\right)\right),$$

$$\mathbf{c}_2(w_r, h_r, w_b, h_b) = \big(\max(2h_r + 1, w_b), w_r + h_b\big).$$

As we have mentioned, the symmetric 1-2 drawings are very easy to analyze. In particular, we can compute *all* grid sizes (w, h) such that $T_l \subseteq (w, h)$. The algorithm to compute these grid sizes was given in [3, Lemma 5]:

Lemma 6 *For a fixed l, the Pareto-optimal pairs (w, h) at level l can be computed in time polynomial in the number of nodes of T_l.*

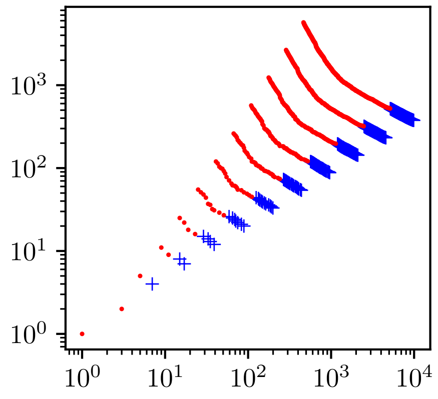


Figure 4: A scatterplot of all Pareto-optimal grid sizes. Blue crosses denote construction 1, red dots denote construction 2. We see from the figure that construction 1 is preferred when $w \gg h$.

We should explain what Pareto-optimal pairs mean in this lemma. Because if $w \leq w'$ and $h \leq h'$ then $T_l \subseteq (w, h) \implies T_l \subseteq (w', h')$, it suffices to consider for each l the pairs (w, h) such that $T_l \subseteq (w, h)$ and there exist no pair (w', h') such that $w' \leq w$, $h' \leq h$, $w' \cdot h' < w \cdot h$, and $T_l \subseteq (w', h')$. We call these pairs *Pareto-optimal* at level l.

Apart from being easy to analyze, the symmetric 1-2 drawings additionally satisfy the following properties, which is proven in [3, Lemma 3].

Lemma 7 *Given any straight-line orthogonal drawing Γ of the complete ternary tree T_l, there exists a rotation of a symmetric 1-2 drawing Γ' whose both width and height are no more than those of Γ.*

We allow 90° rotations, which swaps the width and the height.

3 Motivation: The Pattern of the Pareto-optimal Grid Sizes

We compute the Pareto-optimal grid sizes for small values of l:

- When $l = 1$, the only pair is $(1, 1)$.
- When $l = 2$, the only pair is $(3, 2)$.
- When $l = 3$, there is a pair $(5, 5)$ corresponding to construction 2, and a pair $(7, 4)$ corresponding to construction 1.

We make a scatterplot for all the pairs for each value of l. The result is shown in Figure 4, where both x-axis and y-axis use a logarithmic scale.

From the figure, the pattern is obvious. Our goal is thus to prove that the pattern continues indefinitely.

In order to do so, we need to look at how these grid sizes were computed—the set of Pareto-optimal grid sizes at level l is computed only from the Pareto-optimal grid sizes at level $l - 1$, independent of what happens at earlier levels. As such, our proof will be inductive—assume the Pareto-optimal grid sizes at level $l-1$ satisfy some bound, we prove the Pareto-optimal grid sizes at level l satisfy another bound.

In order to formalize these concepts, we make the following definitions.

Definition 8 ($\leq$-relation for grid sizes) *Let w, h, w', h' be real numbers. We say $(w, h) \leq (w', h')$ if $w \leq w'$ and $h \leq h'$. Similarly, $(w, h) \geq (w', h')$ if $w \geq w'$ and $h \geq h'$.*

Definition 9 *For any set $A \subseteq \mathbb{R}^2$, define the* upper-closure *$C(A) \subseteq \mathbb{R}^2$ to be $C(A) = \{(w, h) \in \mathbb{R}^2 \mid \exists (w', h') \in A, (w', h') \leq (w, h)\}$. We say a set A is* upper-closed *if $A \subseteq \mathbb{R}^2$ and $C(A) = A$.*

Definition 10 *For each $l \geq 1$, define the set E_l to be all pairs $(w, h) \in \mathbb{Z}^2$ such that $T_l \subseteq (w, h)$. Define $S_l = C(E_l)$.*

So for example, at $l = 2$, E_l consists of all pairs of integers (w, h) such that w is odd, $w \geq 3$ and $h \geq 2$, while S_l consists of all pairs of reals (w, h) such that $w \geq 3$ and $h \geq 2$. We see that S_l is the natural extension of E_l to the domain of all reals.

Definition 11 *For any upper-closed set $A \subseteq \mathbb{R}^2$ and real number δ, define the* shift *of A by δ to be $\Delta(A, \delta) = \{(w \cdot \exp \delta, h \cdot \exp \delta) \mid (w, h) \in A\}$.*

Definition 12 *For any upper-closed set $A \subseteq \mathbb{R}^2$, define the* advance *of A to be*

$$N(A) = C\Big(\big\{\mathbf{c}_i(w_r, h_r, w_b, h_b) \mid (w_r, h_r) \in A, (w_b, h_b) \in A, i \in \{1, 2\}\big\}\Big).$$

The functions $\mathbf{c}_1$ and $\mathbf{c}_2$ were introduced for Lemma 5. As such, we get the following:

Lemma 13 *For each $l \geq 1$, $N(S_l) = S_{l+1}$.*

Let A and B be upper-closed sets. We say $A \leq B$ if the "boundary" of A is below the "boundary" of B. For example, if $A = \{(x, y) \mid x \in \mathbb{R}, y \geq 1\}$ and $B = \{(x, y) \mid x \in \mathbb{R}, y \geq 2\}$, then visually, the boundary of A is below the boundary of B. The formal definition is:

Definition 14 *For two upper-closed sets A and B, we say $A \leq B$ if $A \supseteq B$, and $A \geq B$ if $A \subseteq B$.*

Even though the direction of $\leq$ appears reversed compared to the $\subseteq$, by the explanation above, this is the intuitively correct direction. It is also consistent with the $\leq$ notation defined for grid sizes earlier—for (w, h) and $(w', h') \in \mathbb{R}^2$, $(w, h) \leq (w', h')$ if and only if $C(\{(w, h)\}) \leq C(\{(w', h')\})$.

4 Upper Bound: Preliminary

Using these definitions, we explain how the proof of the upper bound proceeds. As previously explained, it will use induction, where δ is a positive real constant and $T \subseteq \mathbb{R}^2$ is a fixed set.[1]

- Base case: $S_{l_0} \leq T$.
- Induction step: If $S_l \leq \Delta(T, d)$ for any $d \geq 0$, then $S_{l+1} \leq \Delta(T, d+\delta)$.

By induction, we get $S_l \leq \Delta(T, \delta \cdot (l - l_0))$ for all $l \geq l_0$. For a suitable set T, this implies the complete ternary tree T_l can be embedded in a grid with area $O(e^{2\delta l}) = O(n^{2\delta/\log 3})$ where n is the number of nodes in the tree T_l.

For example, we may use $T = S_{18}$. Then the base case is trivially satisfied for $l_0 = 18$. The constant δ used is $\log(63761/35808)$, which is in fact the smallest δ such that $S_{19} \leq \Delta(S_{18}, \delta)$. To compute this value of δ, we used the algorithm described in [3, Lemma 5] to compute both S_{18} and S_{19} explicitly. We also have $2\delta/\log 3 < 1.051$, therefore T_l can be embedded in a grid with area $O(n^{1.051})$.

Now we prove that the induction step holds.

Lemma 15 *Let $d \geq 0$. If $N(T) \leq U$, then $N(\Delta(T, d)) \leq \Delta(U, d)$.*

Proof. Expanding out the definitions, it suffices to show the following. For all $i \in \{1, 2\}$, $d \geq 0$, positive reals w_r, h_r, w_b, h_b, let $(w, h) = \mathbf{c}_i(w_r, h_r, w_b, h_b)$, then we need to prove

$$\mathbf{c}_i(w_r \cdot e^d, h_r \cdot e^d, w_b \cdot e^d, h_b \cdot e^d) \leq (w \cdot e^d, h \cdot e^d).$$

So for example, when $i = 2$, for the first component, we need to prove

$$\max(2h_r \cdot e^d + 1, w_b \cdot e^d) \leq \max(2h_r + 1, w_b) \cdot e^d.$$

Since $d \geq 0$, $e^d \geq 1$ and $(2h_r + 1) \cdot e^d \geq 2h_r \cdot e^d + 1$. Other cases are omitted because they are similar. □

Using this, the induction step can be proven. Since $N(T) \leq \Delta(T, \delta)$, we get

$$N(\Delta(T, d)) \leq \Delta(\Delta(T, \delta), d) = \Delta(T, \delta + d).$$

The induction hypothesis gives us $S_l \leq \Delta(T, d)$, so

$$N(S_l) = S_{l+1} \leq N(\Delta(T, d)).$$

Combining the two inequalities, we get $S_{l+1} \leq \Delta(T, \delta + d)$ as desired.

To complete the proof, we just need the following.

[1]We apologize for using T to denote a tree instead of a set in the introduction.

Lemma 16 *Let $T \subseteq \mathbb{R}^2$ be any non-empty upper-closed set, and l_0 be a fixed integer. If $S_l \leq \Delta(T, \delta \cdot (l - l_0))$ for all positive integers $l \geq l_0$, then the area of the smallest grid that T_l can be embedded in is $O(n^{2\delta/\log 3})$, where $n \in \Theta(3^l)$ is the number of nodes in T_l.*

Proof. Pick an arbitrary fixed element $(w, h) \in T$. By definition, $S_l \leq \Delta(T, \delta \cdot (l - l_0))$, so $(w \cdot e^{\delta \cdot (l - l_0)}, h \cdot e^{\delta \cdot (l - l_0)}) \in S_l$. As such, there is a grid with area no more than $w \cdot e^{\delta \cdot (l - l_0)} \cdot h \cdot e^{\delta \cdot (l - l_0)}$ that T_l can be embedded in, this value is $O(n^{2\delta/\log 3})$ as needed. □

5 Lower Bound

Similarly, we will fix a set S, a positive real δ, and a positive integer l_0, and prove:

- Base case: $S_{l_0} \geq S$;
- Induction step: For any l and d, if $S_l \geq \Delta(S, d)$, then $S_{l+1} \geq \Delta(S, d + \delta)$.

As such, for every $l \geq l_0$, we get $S_l \geq \Delta(S, (l - l_0) \cdot \delta)$. For a suitable initial set S, this implies the smallest area of a grid that T_l can be embedded in is $\Omega(\exp(2\delta \cdot l))$.

This time however, the analog of Lemma 15 would be the following (we do not need to use this in the article, as such it is not proved):

> Let $d \leq 0$. If $N(S) \geq U$, then $N(\Delta(S, d)) \geq \Delta(U, d)$.

Note that $d \leq 0$. This means inequalities can only be shifted "backward", not forward. As such, we would need to conceptually define a set S "at infinity", then shift it backwards. To formalize it, we give the following definition.

Definition 17 *For any upper-closed set $A \subseteq \mathbb{R}^2$, define the* advance at infinity *of A to be*

$$\begin{aligned} N^\infty(A) = C\Big(&\big\{(2h_r + w_b, \tfrac{w_r}{2} + \max(\tfrac{w_r}{2}, h_b)) \\ &\quad \mid (w_r, h_r) \in A, (w_b, h_b) \in A\big\} \\ &\cup \big\{(\max(2h_r, w_b), w_r + h_b) \\ &\quad\quad \mid (w_r, h_r) \in A, (w_b, h_b) \in A\big\}\Big). \end{aligned}$$

This should be thought of as $\lim_{d \to +\infty} \Delta(N(\Delta(A, d)), -d)$—shift A to "infinity", advance it, then shift it back.

Also note that N^∞ is invariant under Δ-shifting—formally, for any real d, $N^\infty(\Delta(S, d)) = \Delta(N^\infty(S), d)$.

We get the following:

Lemma 18 *$N(S) \geq N^\infty(S)$. Therefore, if $N^\infty(S) \geq U$, then $N(S) \geq U$.*

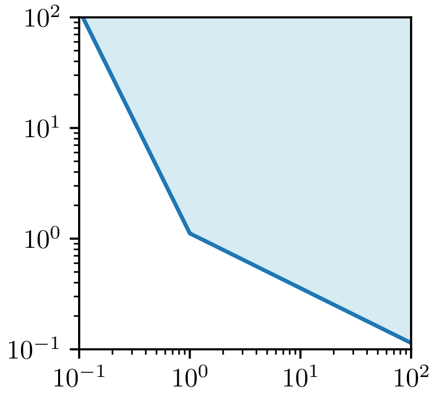


Figure 5: Illustration of the set S used in the lower bound proof.

Using reasoning similar to the previous section, the induction proceeds as follows. Assume the set S satisfies $N^\infty(S) \geq \Delta(S, \delta)$. Then assume the base case $S_1 \geq S$ holds, the induction step can be proven as follows. Using Lemma 18,

$$\begin{aligned} N(\Delta(S,d)) &\geq N^\infty(\Delta(S,d)) \\ &= \Delta(N^\infty(S), d) \geq \Delta(S, d+\delta). \end{aligned}$$

From the induction hypothesis, $S_l \geq \Delta(S,d)$, so

$$S_{l+1} = N(S_l) \geq N(\Delta(S,d)).$$

Combining the two inequalities, we get $S_{l+1} \geq \Delta(S, d+\delta)$ as needed.

Now, the only remaining challenge is to construct such a set S.

Definition 19 *Fix constants $\sigma > 1$ and $\varepsilon \in \mathbb{R}$. Define $S \subseteq \mathbb{R}^2$ to be the upper-closure of the set of points $\{\exp(\omega, \max(\frac{-\omega}{\sigma}, -\omega \cdot \sigma) + \varepsilon) \mid \omega \in \mathbb{R}\}$. Here, we write $\exp(\omega, \eta)$ to denote the pair $(\exp \omega, \exp \eta)$.*

Lemma 20 *For constants $\delta = 0.5667$, $\varepsilon = 0.10995$ and $\sigma = 2.01979$, we have $N^\infty(S) \geq \Delta(S, \delta)$.*

The proof will be deferred for later. In the log-log plot, this set S is an unbounded polygon as illustrated in Figure 5, with the vertex at $\exp(0, \varepsilon)$. Using this lemma, we get the following:

Theorem 21 *Set δ as above. If $T_l \subseteq (w,h)$, then $w \cdot h \in \Omega(n^{2\delta/\log 3}) \geq \Omega(n^{1.031})$, where $n \in \Theta(3^l)$ is the number of nodes in T_l.*

Note that by our definition of $\leq$, the theorem only lower bounds the area of embeddings satisfying the subtree separation property.

Proof. Note that for $l_0 = 2$ then $S_2 \geq S$.

Apply induction by the plan described above, we get $S_l \geq \Delta(S, (l - l_0) \cdot \delta)$ for all $l \geq 2$.

Note that for every $(w,h) \in S$ then $w \cdot h \geq \exp \varepsilon$, therefore for every $(w,h) \in \Delta(S, (l-l_0) \cdot \delta)$ then $w \cdot h \geq \exp(2(l - l_0)\delta + \varepsilon)$, so we are done. ☐

Now we prove Lemma 20.

Proof. Pick $(w,h) \in N^\infty(S)$. Define w_r, h_r, w_b, h_b as in Definition 17, then $(w_r, h_r) \in S$ and $(w_b, h_b) \in S$.

Expanding out these conditions, we get that the assumptions are, for both $\tau \in \{\sigma, \frac{1}{\sigma}\}$:

$$\begin{aligned} \log h_r &\geq \varepsilon - \log w_r \cdot \tau, \\ \log h_b &\geq \varepsilon - \log w_b \cdot \tau. \end{aligned}$$

We need to prove $(w,h) \in \Delta(S, \delta)$. This is equivalent to the following statement: for both $\tau \in \{\sigma, \frac{1}{\sigma}\}$,

$$\log h \geq \delta + \varepsilon + (\delta - \log w) \cdot \tau.$$

In the first case (construction 1), it suffices for us to prove for both $\tau \in \{\sigma, \frac{1}{\sigma}\}$:

$$\log\left(\tfrac{w_r}{2} + h_b\right) \geq \delta + \varepsilon + (\delta - \log(2h_r + w_b)) \cdot \tau.$$

For any positive reals a and b, $\log(a+b) \geq \max(\log a, \log b)$. Define the softmax function $\mathrm{sm}(a,b) = \log(\exp a + \exp b)$, then the left-hand side is $> \log(w_r + h_b) = \mathrm{sm}(\log w_r, \log h_b)$.

Define $\omega_i = \log w_i$, $\eta_i = \log h_i$ for $i \in \{r, b\}$. Then we need to prove

$$\mathrm{sm}(\omega_r - \log 2, \eta_b) \geq \delta + \varepsilon + (\delta - \mathrm{sm}(\eta_r + \log 2, \omega_b)) \cdot \tau.$$

With this new notation, for all $\tau \in \{\sigma, \frac{1}{\sigma}\}$ and $i \in \{r, b\}$, then

$$\eta_i \geq \varepsilon - \omega_i \cdot \tau.$$

Equivalently,

$$\omega_i \geq (\varepsilon - \eta_i) \cdot \tau.$$

Thus, we just need to prove

$$\begin{aligned} &\mathrm{sm}((\varepsilon - \eta_r) \cdot \tau - \log 2, \varepsilon - \omega_b \cdot \tau) \\ &\qquad \geq \delta + \varepsilon + (\delta - \mathrm{sm}(\eta_r + \log 2, \omega_b))\tau. \end{aligned}$$

Set $\varphi = \eta_r - \omega_b$, since the sm function satisfies $\mathrm{sm}(a + d, b + d) = \mathrm{sm}(a,b) + d$ for all a, b, $d \in \mathbb{R}$, this simplifies to

$$\begin{aligned} &\mathrm{sm}((\varepsilon - \varphi)\tau - \log 2, \varepsilon) \\ &\qquad \geq (1+\tau)\delta + \varepsilon - \mathrm{sm}(\varphi + \log 2, 0)\tau. \end{aligned}$$

Equivalently, since $1 + \tau > 0$,

$$\delta \leq \frac{\mathrm{sm}((\varepsilon - \varphi)\tau - \log 2, \varepsilon) - \varepsilon + \mathrm{sm}(\varphi + \log 2, 0)\tau}{1+\tau}.$$

The right hand side only contains one variable φ. We define the function $f_{0,\tau}(\varphi)$ to be the right hand side, and we wish to compute the minimum of $f_{0,\tau}$. Notice that $f'_{0,\tau}(\varphi) = 0$ has an unique solution

$$\varphi = \frac{(\tau-1)\varepsilon - \log 4}{1+\tau}$$

and this can be shown to be the global minimum of $f_{0,\tau}$. At this point, the value of $f_{0,\tau}$ is $\geq \delta$ for both $\tau = \sigma$ and $\tau = \frac{1}{\sigma}$, so we are done.

In the second case (construction 2), we need to prove:

$$\log(w_r + h_b) \geq \delta + \varepsilon + (\delta - \log(\max(w_b, 2h_r))) \cdot \tau.$$

This simplifies to

$$\mathrm{sm}(\omega_r, \eta_b) \geq \delta \cdot (1+\tau) + \varepsilon - \max(\omega_b, \eta_r + \log 2) \cdot \tau.$$

Doing exactly as above, we just need to prove

$$\begin{aligned}\mathrm{sm}((\tau-1)\varepsilon - \eta_r\tau, -\omega_b\tau)& \\ \geq \delta \cdot (1+\tau) - \max(\omega_b, \eta_r + \log 2) \cdot \tau.&\end{aligned}$$

Set $\varphi = \eta_r - \omega_b$ again, this further simplifies to

$$\delta \leq \frac{\mathrm{sm}(0, (\tau-1)\varepsilon - \varphi\tau) + \max(0, \varphi + \log 2) \cdot \tau}{1+\tau}.$$

Set $f_{1,\tau}(\varphi) = \frac{\mathrm{sm}(0,(\tau-1)\varepsilon-\varphi\tau)}{1+\tau}$ and $f_{2,\tau}(\varphi) = \frac{\mathrm{sm}(0,(\tau-1)\varepsilon-\varphi\tau)+(\varphi+\log 2)\cdot\tau}{1+\tau}$, notice that $f_{1,\tau}$ is decreasing, $f_{2,\tau}$ is increasing, and the equation $f_{1,\tau}(\varphi) = f_{2,\tau}(\varphi)$ has a unique solution $\varphi = -\log 2$, at this point

$$f_{1,\tau}(\varphi) = f_{2,\tau}(\varphi) = \frac{\mathrm{sm}(0, (\tau-1)\varepsilon + \tau\log 2)}{1+\tau}.$$

As such, $\max(f_{1,\tau}(\varphi), f_{2,\tau}(\varphi))$ has a global minimum at this point. For both $\tau \in \{\sigma, \frac{1}{\sigma}\}$, this value is $\geq \delta$. □

We are unable to obtain a closed-form formula for these constants. However, it can be efficiently computed to arbitrary precision using software such as Mathematica—namely, find the values of σ, ε and δ such that

$$\begin{aligned}\delta = f_{1,\sigma}(-\log 2) &= f_{1,1/\sigma}(-\log 2) \\ &= f_{0,1/\sigma}\Big(\frac{(1/\sigma-1)\varepsilon - \log 4}{1+1/\sigma}\Big), \quad (1)\end{aligned}$$

then it can be confirmed that $f_{0,\sigma}\big(\frac{(\sigma-1)\varepsilon-\log 4}{1+\sigma}\big) > \delta$.

6 Numerical Proof for Improved Upper Bound

In Section 4, we improved the upper bound on the minimum area required for a straight-line orthogonal drawing of T_l. In order to do so, we used a certain upper-closed set T satisfying $N(T) \leq \Delta(T, \delta)$ for a constant $\delta < 0.577$. Specifically, we used $T = S_{18}$.

If we use a different set T such that $N(T) \leq \Delta(T, \delta')$ for a smaller constant δ', we would be able to improve the upper bound accordingly. In order to simplify the analysis, we use the following.

Lemma 22 *Fix an upper-closed set T such that $C(\{1,1\}) \leq T$, and a constant $\delta > 0$. If $N^\infty(T) \leq \Delta(T, \delta)$, then for all $\varepsilon > 0$, there exists sufficiently large $d > 0$ such that $N(T') \leq \Delta(T', \delta+\varepsilon)$ for $T' = \Delta(T, d)$.*

Proof. Unrolling the definition, we need to prove that with notation as above, for every $(w, h) \in \Delta(T', \delta+\varepsilon)$, then $(w, h) \in N(T')$.

The statement $(w, h) \in \Delta(T', \delta+\varepsilon)$ is equivalent to $w = \exp(d+\delta+\varepsilon)w'$, $h = \exp(d+\delta+\varepsilon)h'$ for $(w', h') \in T$.

By assumption, $N^\infty(T) \leq \Delta(T, \delta)$, so $N^\infty(\Delta(T, d+\varepsilon)) \leq \Delta(T, d+\delta+\varepsilon)$, which means $(w, h) \in N^\infty(\Delta(T, d+\varepsilon))$. Expanding out the definition of N^∞, this means there exists (w_r, h_r) and $(w_b, h_b) \in \Delta(T, d+\varepsilon)$ such that either

$$w = 2h_r + w_b \text{ and } h = \frac{w_r}{2} + \max\Big(\frac{w_r}{2}, h_b\Big) \qquad (2)$$

or

$$w = \max(2h_r, w_b) \text{ and } h = w_r + h_b. \qquad (3)$$

The statement we need to prove is $(w, h) \in N(T')$. With (w_r, h_r) and (w_b, h_b) as above, we get

$$(w_r/\exp\varepsilon, h_r/\exp\varepsilon), (w_b/\exp\varepsilon, h_b/\exp\varepsilon) \in T'. \quad (4)$$

In the first case where Eq. 2 holds, we will use Eq. 4 to get:

$$\left(\frac{2h_r + w_b}{\exp\varepsilon}, \frac{w_r}{2\exp\varepsilon} + \max\Big(\frac{w_r}{2\exp\varepsilon}, \frac{h_b}{\exp\varepsilon} + \frac{1}{2}\Big)\right) \in N(T').$$

We will show that it is possible to pick d large enough such that both $w \geq \frac{2h_r+w_b}{\exp\varepsilon}$ and $h \geq \frac{w_r}{2\exp\varepsilon} + \max(\frac{w_r}{2\exp\varepsilon}, \frac{h_b}{\exp\varepsilon} + \frac{1}{2})$ holds. Assume otherwise. Since $\varepsilon > 0$, $w \geq \frac{2h_r+w_b}{\exp\varepsilon}$ always, so $\frac{w_r}{2} + \max(\frac{w_r}{2}, h_b) < \frac{w_r}{2\exp\varepsilon} + \max(\frac{w_r}{2\exp\varepsilon}, \frac{h_b}{\exp\varepsilon} + \frac{1}{2})$, so $\max(\frac{w_r}{2}, h_b) < \max(\frac{w_r}{2\exp\varepsilon}, \frac{h_b}{\exp\varepsilon} + \frac{1}{2})$. Therefore $\frac{w_r}{2} < \frac{h_b}{\exp\varepsilon} + \frac{1}{2}$, so $\max(\frac{w_r}{2\exp\varepsilon}, \frac{h_b}{\exp\varepsilon} + \frac{1}{2}) = \frac{h_b}{\exp\varepsilon} + \frac{1}{2}$, so $h_b < \frac{h_b}{\exp\varepsilon} + \frac{1}{2} \iff h_b < \frac{1}{2(1-\exp(-\varepsilon))}$. Since $(w_b, h_b) \in \Delta(T, d+\varepsilon)$ and $C(\{1,1\}) \leq T$, by picking d large enough, we can make $h_b < \frac{1}{2(1-\exp(-\varepsilon))}$ impossible. Note that d still only depends on ε and not on h_b.

When Eq. 3 holds instead, we use Eq. 4 to get

$$\left(\max\Big(\frac{2h_r}{\exp\varepsilon} + 1, \frac{w_b}{\exp\varepsilon}\Big), \frac{w_r + h_b}{\exp\varepsilon}\right) \in N(T').$$

Proceed similarly. □

We now explain how the set T was found. Consider the set S in Section 5. With values of σ, ε and δ satisfying Eq. 1 exactly, we get $S \leq \Delta(N^\infty(S), -\delta)$. Define the operator $P(S) = \Delta(N^\infty(S), -\delta)$ and let $P^k(S)$ be P applied k times on S, then we get $S \leq P(S)$, which means $P^k(S) \leq P^{k+1}(S)$ for all k, and we conjecture the following based on numerical evidence:

Conjecture 1 $\lim_{k\to\infty} P^k(S) = \{(w,h) \mid h \geq f(w)\}$ *for an analytic function* f *such that* $f(\exp\omega) - (\varepsilon - \omega\cdot\sigma) \to 0$ *as* $\omega \to -\infty$ *and* $f(\exp\omega) - (\varepsilon - \omega/\sigma) \to 0$ *as* $\omega \to +\infty$*. Furthermore, let* $P^\infty(S)$ *denote that limiting set, then* $P(P^\infty(S)) = P^\infty(S)$*.*

Such a function f must satisfy a certain functional equation; however, we are unable to solve it or prove that the solution exists. If the conjecture holds, pick $T = \Delta(P^\infty(S), d) \cap C(\{w,h\})$ for some large d and suitable w and h, we get $N^\infty(T) \leq \Delta(T, \delta)$, which would have implied the area is $\in \widetilde{\Theta}(n^{2\delta/\log 3})$. However, we are only able to find a numerical approximation of T that gives the bound $O(n^{1.032})$.

The algorithm to compute the certificate T for the upper bound computes $P^k(S)$ for a sufficiently large number k, with some approximations and linear interpolations to make the time complexity manageable, and we only iterate on $\mathbf{c}_2$ in the initial iterations, which heuristically have better numerical stability.

7 Conclusion

In this article, we improve the upper bound on the minimum area required for a straight-line orthogonal drawing of the complete ternary tree, and prove an almost-matching lower bound in the special case of drawings satisfying the subtree separation property.

There are still several interesting open questions that need further research, namely whether there is a non-trivial lower bound when the drawing is not required to satisfy the subtree separation property, and whether the constant can be determined analytically to prove that our lower bound is tight.

References

[1] A. Ali. Straight line orthogonal drawings of complete ternary trees. Technical report, MIT Summer Program in Undergraduate Research, July 2015.

[2] T. M. Chan. Tree drawings revisited. *Discrete & Computational Geometry*, 63(4):799–820, June 2019.

[3] B. Covella, F. Frati, and M. Patrignani. On the area requirements of planar straight-line orthogonal drawings of ternary trees. *Theoretical Computer Science*, 852:197–211, Jan. 2021.

[4] P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Computational Geometry*, 2(4):187–200, Dec. 1992.

[5] F. Frati. Straight-line orthogonal drawings of binary and ternary trees. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Graph Drawing*, pages 76–87, Berlin, Heidelberg, 2008. Springer, Berlin, Heidelberg.

[6] L. G. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, C–30(2):135–140, Feb. 1981.

On Upward Planar Embeddings of Paths with Partially Fixed Vertices

Stephane Durocher[*][†] Myroslav Kryven[‡] Tamara Mchedlidze[§]

Abstract

We study a variant of the problem of determining whether a given oriented n-vertex path Π has an *upward planar straight-line embedding* onto a given set P of n points in the plane, where some vertices of Π are assigned (fixed) to specific points of P. We consider the special case where all the *switches* (sources or sinks) of Π are fixed and an embedding of the remaining vertices of the path is sought. We give a polynomial time dynamic program to compute such an embedding if Π has four switches. If Π has only three switches, we provide a faster $O(n^2)$-time algorithm.

1 Introduction

Straight-line drawings are one of the most common paradigms for embedding a given graph G in the plane to visualize G. In a classical result, Fary [16], Stein [22], and Wagner [23] showed that every planar graph has a crossing-free straight-line drawing. Since then, various efficient algorithms have been developed to construct crossing-free straight-line drawings of planar graphs [11, 21]. Most of these algorithms construct a point set onto which they planarly embed the graph. In many applications, however (e.g., [14,15]) we are not free to choose the point set on which to map graph vertices, but we are required to embed the graph onto a given point set P. Constructing *planar straight-line embeddings* (PSE) of graphs onto sets of points has been well studied for over thirty years [1,8–10,19]. The problem has also been considered in the case where some vertices of the graph are fixed (via some bijective mapping β) to specific points in the point set P, called *planar straight-line embedding with fixed vertices* (PSE$_\beta$) [6,12,20]. This variant is important since it helps preserve the user's preference (e.g., mental map or chip layout) of the embedding of the graph. The directed versions of such problems, constructing an *upward planar straight-line embedding* (UPSE) or *upward planar straight-line embedding with fixed vertices* (UPSE$_\beta$), where the embeddings must be such that the edges point upwards, have received attention recently [5,7,13,17,18]. We adopt the common assumption that the edges in such drawings are drawn strictly upward, that is, no horizontal edges are allowed. Let us formalize these problems. Throughout the paper we assume that the number of vertices is the same as the number of points, namely $|V(G)| = |P| = n$, unless stated otherwise.

(U)PSE(P, G)	
Given:	a (directed) graph G and a set P of points in $\mathbb{R}^2$ such that $\lvert V(G)\rvert = \lvert P\rvert$.
Decide:	whether there exists an (upward) planar straight-line embedding of G onto P.

(U)PSE$_\beta(P, G)$	
Given:	a (directed) graph G, a set P of points in $\mathbb{R}^2$, a subset of vertices $U \subset V(G)$, a subset of points $Q \subset P$, and a bijective mapping β of U onto Q.
Decide:	whether there exists an (upward) planar straight-line embedding of G onto P such that β is respected.

There are two main research directions in this area. The first direction is structural: studying whether graphs from a certain class of (directed) planar graphs such as paths, trees, or outerplanar graphs can be embedded onto a certain type of point set, such as points in general position, convex, or *one-sided convex* (that is, the top-most point and the bottom-most point are consecutive on the convex hull). For example, Pach et al. [19] showed that any outerplanar graph admits a PSE onto any set of points in general position. For the directed version of the problem, however, Binucci et al. [7] showed that a directed tree does not always have an UPSE onto a point set in convex position. On the other hand, Binucci et al. [7] showed that every *oriented path* (that is, a path where each edge is assigned an orientation) admits an UPSE on every convex point set of the same size. This result was extended by Angelini et al. [4] to the class of caterpillars. If, however, the point set P is in general position, it is still an open question whether every oriented path Π with $|\Pi| = |P|$ admits an UPSE on P.

A *switch* of a path is either a source or a sink of the path (an endpoint is considered a switch). A *section* is a

[*]University of Manitoba, stephane.durocher@umanitoba.ca
[†]The work of this author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).
[‡]Amherst College, mkryven@amherst.edu
[§]University of Utrecht, t.mtsentlintze@uu.nl

path between two consecutive switches and the *length* of a section is the number vertices in the section (including the switches). Some partial results are known for oriented paths with at most five switches and at least two sections of length two. Such a path always admits an UPSE on any general point set of the same size [7].

The second research direction is about the complexity of the problem. Not surprisingly, many variants are NP-hard. Cabello [10] showed that computing a PSE is NP-hard by giving a reduction from 3-Partition. There is an efficient algorithm for constructing a PSE of an outerplanar graph onto any point set in general position [8], with even faster algorithms available when the point set is in convex position or the graph is a tree [9]. Kaufmann et al. [17] showed that a reduction, similar to that of Cabello [10], also works for computing an UPSE. Arseneva et al. [5] showed that computing an UPSE$_\beta$ is NP-hard even for trees with only one vertex fixed to a given point. Bagheri and Razzazi [6] showed that computing a PSE$_\beta$ is NP-hard for an undirected tree when more than one vertex is allowed to be fixed. Recently Alegría et al. [2, 3] showed that computing an UPSE is NP-complete even for a tree with no fixed vertices. On the positive side, they showed that for a planar st-graph with n vertices and an st-cutset of size at most k, it is possible to compute an UPSE in $O(n^{4k})$ time with $O(n^{3k})$ space.

Our contribution We consider a variant of the problem UPSE$_\beta(P,\Pi)$, called UPSE$_{\beta'}(P,\Pi)$, where the mapping β' fixes all the switches of Π and no other vertices. We call such an embedding of Π an *upward planar straight-line embedding with fixed switches* (UPSE$_{\beta'}$). We give polynomial time algorithms for the problem for any point set P and any path Π with at most four switches. Our techniques are similar to those used by Alegría et al. [2, 3], however, we provide new insights into the structure of the problem when some of the vertices are fixed. For a path Π_4 with four switches we give a polynomial time dynamic program in Section 2 and for a path Π_3 with three switches we give a $O(n^2 \log n)$ time algorithm that uses the structure of the problem; see Section 3.

2 Dynamic Program for UPSE$_{\beta'}(P,\Pi_4)$

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane sorted by their y-coordinates (and then by x-coordinate) and Π_4 be a path with four switches. To solve UPSE$_{\beta'}(P,\Pi_4)$ we first formulate a subproblem where we are constructing upward drawings of the three sections of Π_4 (recall, that each such section is a path with edges oriented in the same direction). Our DP algorithm will be “growing“ each of these paths from their fixed bottom points until their fixed top points making sure that each point on the way is taken by some path; see Figure 1 for an illustration. Then we will show that one of these subproblems actually gives us a solution to UPSE$_{\beta'}(P,\Pi_4)$.

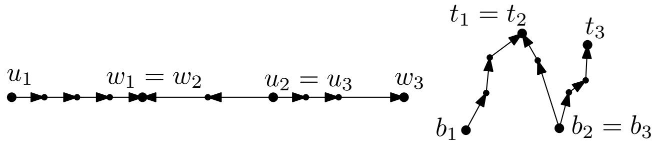


Figure 1: The path Π_4 on the left; UPSE$_{\beta'}$ of Π_4 onto P on the right.

Let $\pi(p_i, p_j; s)$, for $1 \le i \le j \le n$, be a planar strictly y-monotone path from p_i to p_j that has exactly s points. Recall, that by our definition of strict y-monotonicity, $\pi(p_i, p_j; s)$ cannot have horizontal edges. Let

$$f(p_{i_1}, p_{j_1}, s_1;\ p_{i_2}, p_{j_2}, s_2;\ p_{i_3}, p_{j_3}, s_3) \qquad (1)$$

be TRUE if there exist paths $\pi(p_{i_r}, p_{j_r}; s_r)$, for $r = 1, 2, 3$, such that:

A any horizontal line that crosses the paths $\pi(p_{i_1}, p_{j_1}; s_1)$, $\pi(p_{i_2}, p_{j_2}; s_2)$, and $\pi(p_{i_3}, p_{j_3}; s_3)$, crosses them in this order from left to right.

B for each $i \in [\min\{i_1, i_2, i_3\}, \ldots, \max\{j_1, j_2, j_3\}]$, the point p_i belongs to some path $\pi(p_{i_r}, p_{j_r}; s_r)$,

C no two paths $\pi(p_{i_r}, p_{j_r}; s_r)$ and $\pi(p_{i_l}, p_{j_l}; s_l)$ share points except for the switches, that is, p_{j_r} and p_{j_l} or p_{i_r} and p_{i_l} if $|r - l| = 1$ for $r, l \in \{1, 2, 3\}$.

and FALSE otherwise. Condition A fixes the left-to-right order of the paths $\pi(p_{i_1}, p_{j_1}; s_1)$, $\pi(p_{i_2}, p_{j_2}; s_2)$, and $\pi(p_{i_3}, p_{j_3}; s_3)$; this implies that the paths do not cross. Condition B implies that all the points in the range $[\min_{r=1,2,3} p_{i_r}, \max_{r=1,2,3} p_{j_r}]$ are consumed by the paths[1]. Condition C allows us to connect the paths to form an UPSE$_{\beta'}$ of the original path Π_4.

Assume that one endpoint of the path is a source, and the other endpoint is a sink; otherwise the argument is symmetric. Label the switches $U \subset V(\Pi_4)$ of Π_4 as follows $u_1, w_1 = w_2, u_2 = u_3, w_3$ in the order from left to right and let β' be such a mapping that maps these switches to the points $b_1, t_1 = t_2, b_2 = b_3, t_3$ respectively; see Figure 1. Then, by definition, UPSE$_{\beta'}(P,\Pi_4)$ has a solution if and only if

$$f(b_1, t_1, s_1^\star;\ b_2, t_2, s_2^\star;\ b_3, t_3, s_3^\star) \qquad (2)$$

is TRUE, where $s_r^\star$, for $r = 1, 2, 3$, is the size of the monotone subpath of Π_4 between u_r and w_r included.

We show (in Lemma 1) that the problem of computing f has optimal substructure, that is, we can express in polynomial time (2) via subproblems of smaller size.

[1]We sometimes write $p_i \le p_j$ to indicate that $i \le j$.

Lemma 1 *If there exists $\ell \in \{1, 2, \dots, n\}$ such that*

$$f(b_1, p_{j_1}, s_1;\ b_2, p_{j_2}, s_2;\ b_3, p_{j_3}, s_3) \tag{3}$$

is True *for all (j_r, s_r, r) where $j_r \le \ell$, $1 \le s_r \le s_r^\star$, and $1 \le r \le 3$, then we can compute f for each (j_r, s_r, r), where $j_r \le \ell + 1$, $1 \le s_r \le s_r^\star$, and $1 \le r \le 3$ in $O(n)$ time.*

Proof. Let us fix some of the three sections, say the second one, (for the first and the third the argument will be similar) and let us show how to extend it upwards by one more point, that is, how to compute

$$f(b_1, p_{j_1}, s_1;\ b_2, p_{\ell+1}, s_2;\ b_3, p_{j_3}, s_3) \tag{4}$$

for any $1 \le s_r \le s_r^\star$ for all $1 \le r \le 3$. Assume for now that $p_{\ell+1}$ is not the terminal point, that is, $p_{\ell+1} \ne t_2$, otherwise we will use similar arguments.

Let $P_{\le \ell} = \{p_i \mid i \le \ell\}$. If (4) is True, then there is a point $p_{j_2} \in P_{\le \ell} \setminus (\{p_{j_1}, p_{j_3}\} \cup \{b_1, b_3\})$ such that

a) $p_{j_2}p_{\ell+1}$ is not a horizontal segment,

b) the following is True

$$f(b_1, p_{j_1}, s_1;\ b_2, p_{j_2}, s_2 - 1;\ b_3, p_{j_3}, s_3), \text{ and} \tag{5}$$

c) among those sets of three paths for which (5) is True there is one with paths $\pi(b_1, p_{j_1}; s_1)$, $\pi(b_2, p_{j_2}; s_2 - 1)$, and $\pi(b_3, p_{j_3}; s_3)$ that do not cross the segment $p_{j_2}p_{\ell+1}$.

Condition b) is given by assumption. Therefore, we know whether (5) is True by assumption. Thus, it is only left to verify Condition c). Checking Condition c) is trickier, since we may not know all the sets of three paths that solve (5) and check whether they do not cross the segment $p_{j_2}p_{\ell+1}$ (there might be simply too many). However, we show that we can do this in polynomial time using the structure of the solution to (5).

Let the paths $\pi(b_1, p_{j_1}; s_1)$, $\pi(b_2, p_{j_2}; s_2 - 1)$, and $\pi(b_3, p_{j_3}; s_3)$ be some solution to (5). There are only four relevant cases of how the points p_{j_1}, p_{j_2}, and p_{j_3} are placed relative to each other (the other cases are symmetric).

Case 1 ($\mathbf{p_{j_1} < p_{j_2}}$ and $\mathbf{p_{j_3} < p_{j_2}}$) then $p_{j_2}p_{\ell+1}$ does not cross any of the paths.

Case 2 ($\mathbf{p_{j_1} < p_{j_2} < p_{j_3}}$), then the edges of the first path do not interfere with the segment $p_{j_2}p_{\ell+1}$ because the paths are y-monotone. By Condition B each point p_i such that $j_2 < i \le j_3$ is in the third path (because the first and the second path do not contain points above p_{j_2}). Because of the left-to-right order of the paths (that is, any horizontal line crosses the path in the order first, second, than the third; see Condition A), all such points must be strictly to the right of the segment $p_{j_2}p_{\ell+1}$. Therefore, $p_{j_2}p_{\ell+1}$ crosses the third path if and only if there is a point of P to the left of $p_{j_2}p_{\ell+1}$; see Figure 2a.

Case 3 ($\mathbf{p_{j_1} > p_{j_2} > p_{j_3}}$) is symmetric to Case 2.

Case 4 ($\mathbf{p_{j_2} < p_{j_1}}$ and $\mathbf{p_{j_2} < p_{j_3}}$) here we assume that $p_{j_1} < p_{j_3}$ (otherwise the argument is symmetric). First, consider the points that are above p_{j_1}. The segment $p_{j_2}p_{\ell+1}$ does not cross $\pi(b_3, p_{j_3}; s_3)$ above p_{j_1} if and only if all the points p_i for $j_1 < i \le j_3$ are strictly to the right of $p_{j_2}p_{\ell+1}$.

Now, focus on the points that are strictly between p_{j_2} and $p_{\ell+1}$, let the set of such points be denoted by $P(j_2, \ell+1)$ and let $L(j_2, \ell+1)$ be the subset of points in $P(j_2, \ell+1)$ that are strictly to the left of the segment $p_{j_2}p_{\ell+1}$ and $R(j_2, \ell+1)$ be the subset of the points of $P(j_2, \ell+1)$ that are strictly to the right; see Figure 2b. Note that if there is a point on $p_{j_2}p_{\ell+1}$ we can discard such a candidate solution to (4).

The point p_{j_2} is the bottommost point among $p_{j_1}, p_{j_2}, p_{j_3}$, and so, in any solution to (5) the paths $\pi(b_1, p_{j_1}; s_1)$ and $\pi(b_3, p_{j_3}; s_3)$ contain the points $P(j_2, \ell+1)$.

Because of the left-to-right order of the paths (see Condition A), the path $\pi(b_1, p_{j_1}; s_1)$ does not cross $p_{j_2}p_{\ell+1}$ if and only if the points in $P(j_2, \ell+1)$ that are used by the path are exactly the points $L(j_2, \ell+1)$. Otherwise a point $p_i \in L(j_2, \ell+1)$ that is not taken by $\pi(b_1, p_{j_1}; s_1)$ must belong to $\pi(b_3, p_{j_3}; s_3)$ (see Condition B) and there is a crossing between $\pi(b_3, p_{j_3}; s_3)$ and the segment $p_{j_2}p_{\ell+1}$. Similarly, the path $\pi(b_3, p_{j_3}; s_3)$ does not cross $p_{j_2}p_{\ell+1}$ if and only if the points in $P(j_2, \ell+1)$ that are used by the path are exactly the points $R(j_2, \ell+1)$. Note that $p_{j_1} \in L(j_2, \ell+1)$ and $p_{j_3} \in R(j_2, \ell+1)$.

Therefore, (4) is True if and only if

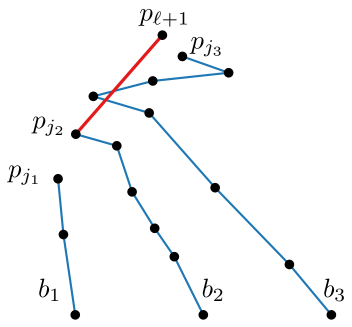
$$\begin{aligned} f\Big(& b_1, p_1^\star, s_1 - |L(j_2, \ell+1)| + 1; \\ & b_2, p_{j_2}, s_2 - 1; \\ & b_3, p_3^\star, s_3 - |R(j_2, \ell+1)| + 1\Big) \end{aligned} \tag{6}$$

is TRUE, where $p_1^\star = \min L(j_2, \ell+1)$ and $p_3^\star = \min R(j_2, \ell+1)$; see Figure 2b. By our assumption, we have already computed f for all points up to $p_{\ell+1}$ (excluding $p_{\ell+1}$) and all sizes s_r ($r = 1, 2, 3$), this includes (6), and we can look up its value in linear time.

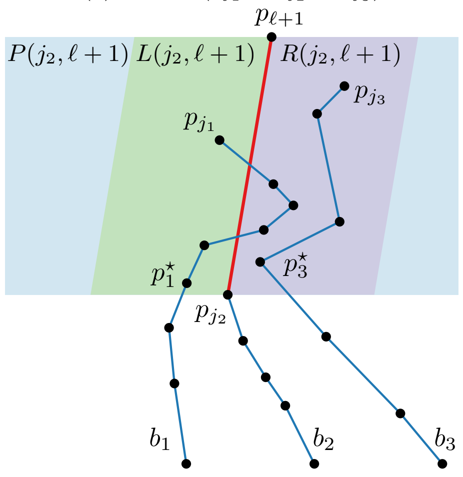
Extending the first or the third path is similar to extending the second path with the difference that all of the points must be to one side of the new edge. □

Lemma 1 yields the following.

Theorem 2 *For a given point set P, a path Π_4 with four switches, and an assignment β' of points from P to the switches, we can decide whether $\mathrm{UPSE}_{\beta'}(P, \Pi_4)$ has a solution, or equivalently whether (2) is* True, *as well as construct the solution, in time $O(n^7)$.*



(a) Case 2 ($p_{j_1} < p_{j_2} < p_{j_3}$)



(b) Case 4 ($p_{j_2} < p_{j_1}$ and $p_{j_2} < p_{j_3}$)

Figure 2: Extending the solution to (4) from (5).

Proof. We fill the table T representing the values of f in (3) in a forward pass. Assume that we have computed f for all (j_r, s_r, r) such that $j_r \le \ell$, $1 \le s_r \le s_r^\star$, and $1 \le r \le 3$. According to Lemma 1 we can correctly compute f for each (j_r, s_r, r), where $j_r \le \ell+1$, $1 \le s_r \le s_r^\star$, and $1 \le r \le 3$, in linear time. Therefore, for each entry of the table T we need to spend linear time. Furthermore, the bottom points b_1, b_2, and b_3 are fixed. Therefore, the whole table can be computed in $O(n^7)$ time.

Note that the initial value

$$f(b_1, b_1, 1;\ b_2, b_2, 1;\ b_3, b_3, 1) \tag{7}$$

is trivially satisfied because each of the paths consists of just a single vertex. □

3 Faster Algorithm for $\mathrm{UPSE}_{\beta'}(P, \Pi_3)$

Let $P = \{p_1, p_2, \dots, p_n\}$ be a set of n points in the plane sorted by their y-coordinates and Π_3 be a path with three switches. In this section we give an $O(n^2 \log n)$-time algorithm for $\mathrm{UPSE}_{\beta'}(P, \Pi_3)$. First, we note some properties of an $\mathrm{UPSE}_{\beta'}$ of Π_3 onto P. Let the switches $U \subset V(\Pi_3)$ of Π_3 be labeled as follows u_1, w, u_2 in the order from left to right and let $\beta'(u_1) = b_1$, $\beta'(w) = t$, and $\beta'(u_2) = b_2$. Let s_1 be the number of vertices between u_1 and w and s_2 be the number of vertices between u_2 and w. Any $\mathrm{UPSE}_{\beta'}$ of Π_3 onto P will consist of two monotone paths $\pi(b_1, t; s_1)$ and $\pi(b_2, t; s_2)$ satisfying Conditions A–C. We will refer to them as the *first* and the *second* path. The vertex t must be the topmost vertex in P and either b_1 or b_2 must be the bottommost vertex in P. Assume w.l.o.g. that b_1 immediately precedes b_2 since otherwise, the first path picks greedily all the points that are below b_2. In addition, we assume, for now, that there are no two points with the same y-coordinate except for maybe b_1 and b_2. Let ℓ_1 be the number of points $L_1 \subset P$ on the left side of the convex hull $H(P_1)$ of the points $P_1 \subset P$ that are vertically between b_1 and t, and let ℓ_2 be the number of points $L_2 \subset P$ on the right side of the convex hull $H(P_2)$ of the points $P_2 \subset P$ that are vertically between b_2 and t. Let us state some useful observations.

Observation 1 *A path Π_3 has no $\mathrm{UPSE}_{\beta'}$ onto a point set P if $s_1 < \ell_1$ or $s_2 < \ell_2$.*

This is due to the fact that in a solution to $\mathrm{UPSE}_{\beta'}(P, \Pi_3)$ the path $\pi(b_1, t; s_1)$ must contain all the points in L_1 and the path $\pi(b_2, t; s_2)$ must contain all the points in L_2; see Figure 3. We show that if $s_1 \ge \ell_1$, $s_2 \ge \ell_2$, and $s_1 + s_2 = n + 1$, then Π_3 always has an $\mathrm{UPSE}_{\beta'}$ onto P. Let us introduce some helpful notation first. Let p be a point on $\pi(b_2, t; s_2)$. If we can remove p from $\pi(b_2, t; s_2)$ and add it to $\pi(b_1, t; s_1)$ without introducing crossings, we say that p can be *flipped* from $\pi(b_2, t; s_2)$ into $\pi(b_1, t; s_1)$; see Figure 4. This results in two new paths $\pi(b_1, t; s_1 + 1)$ and $\pi(b_2, t; s_2 - 1)$ which are closer to the solution. For a point p on the path $\pi(b_2, t; s_2)$ we say that it is *concave* if p forms an angle that is larger than 180° on the left side; see the point $p_{r^\star}$ in Figure 4 for an example of a concave point.

Lemma 3 *Assume that an $\mathrm{UPSE}_{\beta'}$ of a path Π_3 onto a point set P with the paths $\pi(b_1, t; s_1)$ and $\pi(b_2, t; s_2)$ exists, where $s_2 > \ell_2$ and $s_1 + s_2 = n + 1$. Then there exists a point $p_{r^\star}$ on $\pi(b_2, t; s_2)$ that can be flipped from $\pi(b_2, t; s_2)$ into $\pi(b_1, t; s_1)$.*

Proof. Consider such a path $\pi(b_2, t; s_2)$ with $s_2 > \ell_2$, then this path contains all the points in L_2 on the convex hull plus at least one more point. This implies that $\pi(b_2, t; s_2)$ must have a concave vertex p_r. Let $p_i p_j$, for $i < j$ be an edge of $\pi(b_1, t; s_1)$ that a horizontal line through p_r crosses. If the triangle $p_i p_r p_j$ does not contain any other vertex $p_{r'}$ of the second path, p_r can be

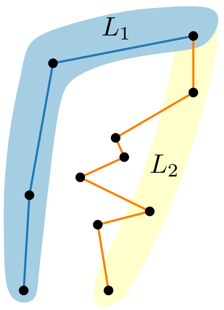


Figure 3: The vertices in $L1$ ($L2$) must be occupied by the left (right) path.

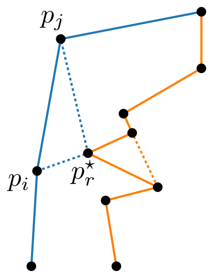


Figure 4: Flipping $p_r^\star$ from $\pi(b_2, t; s_2)$ into $\pi(b_1, t; s_1)$.

flipped from the second path to the first path. Otherwise, we let $p_r := p_{r'}$ and check again whether the triangle $p_i p_r p_j$ does not contain any other vertex of the second path. By repeating this process we find a vertex $p_{r^\star}$ that can be flipped from $\pi(b_2, t; s_2)$ into $\pi(b_1, t; s_1)$; see Figure 4. Furthermore $p_{r^\star}$ is a concave point, therefore, the new edges in the right path do not cross the left path. □

Lemma 4 *The problem* $\text{UPSE}_{\beta'}(P, \Pi_3)$ *has a solution for all* $s_1 \geq \ell_1$ *and* $s_2 \geq \ell_2$ *such that* $s_1 + s_2 = n + 1$.

Proof. We can start with an $\text{UPSE}_{\beta'}$ of a path $\Pi_3^{(0)}$ onto P such that $s_1^{(0)} = \ell_1$, $s_2^{(0)} = n + 1 - \ell_1$ and apply Lemma 3 ($s_1 - \ell_1$) times, every time flipping one point from the second path into the first. □

Lemma 4 implies that to decide whether an $\text{UPSE}_{\beta'}$ of a path Π_3 onto a given point-set P exists, we only need to compute the numbers ℓ_1 and ℓ_2. We can do this by computing the left and right hulls of P in time $O(n \log n)$. The proof of Lemma 4 also implies an $O(n^2)$-time algorithm for computing an $\text{UPSE}_{\beta'}$ of Π_3 onto P, where we perform at most n flips each taking $O(n)$ time.

Theorem 5 *For a given point set* P*, a path* Π_3 *with three switches, and an assignment* β' *of points from* P *to the switches, we can decide whether* $\text{UPSE}_{\beta'}(P, \Pi_3)$ *has a solution in time* $O(n \log n)$ *and construct one in time* $O(n^2)$.

Let us now drop the restriction that P does not have points with the same y-coordinates.

Observation 2 *Assume an* $\text{UPSE}_{\beta'}$ *of* Π_3 *onto* P *exists. If there are two points in* P *with the same* y*-coordinate, then the left point belongs to* $\pi(b_1, t; s_1)$ *and the right point belongs to* $\pi(b_2, t; s_2)$*. Moreover, no three points in* P *have the same* y*-coordinates.*

Observation 2 follows from the left-to-right order of the paths $\pi(b_1, t; s_1)$ and $\pi(b_2, t; s_2)$ in the embedding; see Condition A.

Let $(p_{i_1}, p_{j_1}), (p_{i_2}, p_{j_2}), \ldots, (p_{i_r}, p_{j_r})$ be the pairs of points such that in each pair the points have the same y-coordinates. According to Observation 2, for $l = 1, 2, \ldots, r$, the point p_{i_l} must be in the first path and p_{j_l} must be in the second path. Let a horizontal l-*level*, for $l = 1, 2, \ldots, r-1$ be the horizontal strip with bottom at p_{i_l} and top at $p_{i_{l+1}}$ included. Let P_l be the points in the l-level and $n_l = |P_l|$. In an $\text{UPSE}_{\beta'}$ of Π_3 onto P, for each $l = 1, 2, \ldots, r-1$, the first path goes from p_{i_l} to $p_{i_{l+1}}$ and the second path goes from p_{j_l} to $p_{j_{l+1}}$ using only the points in the l-level. Therefore, to find an $\text{UPSE}_{\beta'}$ of Π_3 onto P we need to find, for each l-level, two non-crossing paths $\pi(p_{i_l}, p_{i_{l+1}}; s_1^l)$ and $\pi(p_{j_l}, p_{j_{l+1}}; s_2^l)$ such that the following two conditions hold

- $s_1^l + s_2^l = n_l$, for $l = 1, 2, \ldots, r-1$, and
- $\sum_{l=1}^{r-1}(s_1^l - 1) = s_1 - 1$ and $\sum_{l=1}^{r}(s_2^l - 1) = s_2 - 1$,

where $s_1^l \geq \ell_1^l$ ($s_2^l \geq \ell_2^l$), for $l = 1, 2, \ldots, r-1$, is the number of points on the left (right) convex chain in the l-level. We can achieve this using Lemma 4 and the algorithm from Theorem 5 for each l-level. We can greedily find the pairs (s_1^l, s_2^l), for $l = 1, 2, \ldots, r-1$, satisfying the above conditions in linear time by first ensuring feasibility and then incrementally filling the sum requirements while respecting the constraints.

4 Discussion

We gave a dynamic program to solve $\text{UPSE}_{\beta'}$ for a path with at most four switches. It remains open whether this can be generalized to paths with more switches. In particular, is there an FPT algorithm parameterized by k to solve $\text{UPSE}_{\beta'}$ for paths with at most k switches.

The main difficulty in extending our DP approach in Section 2 from four switches to five switches (or from three sections to four sections) is that we cannot easily assign vertices to paths when extending a partial solution as we did it using the subsets $L(j_2, \ell + 1)$ and $R(j_2, \ell + 1)$ in the proof of Lemma 1; see, for example, Figure 5.

Beyond the computational aspects of the problem examined in this paper, the open question of Arseneva et

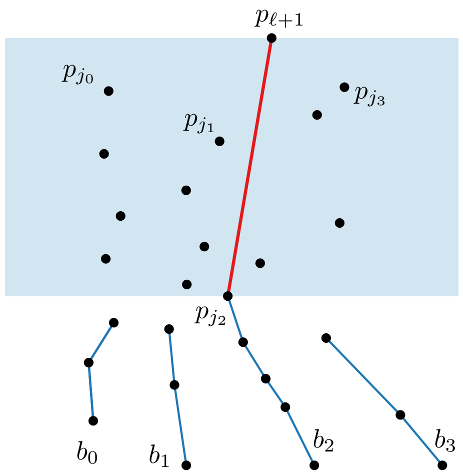


Figure 5: When extending the third path by the vertex $p_{\ell+1}$ from the partial solution for four paths from b_i up to p_{j_i} for $i = 0, 1, 2, 3$ similarly as described in the proof of Lemma 1 we do not know exactly what points to the left of the segment $p_{j_2}p_{\ell+1}$ are taken by the first path and what points are taken by the second path.

al. [5] remains: Does every oriented path Π admit an UPSE on every general point set with $|\Pi| = |P|$?

Acknowledgments We thank the anonymous reviewer for their careful reading and insightful comments, which helped improve the presentation and clarity of this paper.

References

[1] Aichholzer, O., Hackl, T., Lutteropp, S., Mchedlidze, T., Vogtenhuber, B.: Embedding four-directional paths on convex point sets. Journal of Graph Algorithms and Applications **19**(2), 743–759 (2015)

[2] Alegría, C., Caroppo, S., Da Lozzo, G., D'Elia, M., Di Battista, G., Frati, F., Grosso, F., Patrignani, M.: Upward Pointset Embeddings of Planar st-Graphs. In: Felsner, S., Klein, K. (eds.) Graph Drawing. LIPIcs, vol. 320, pp. 24:1–24:18. Springer (2024)

[3] Alegría, C., Caroppo, S., Da Lozzo, G., D'Elia, M., Di Battista, G., Frati, F., Grosso, F., Patrignani, M.: Upward Pointset Embeddings of Planar st-Graphs. Algorithmica **87**, 930–960 (2025)

[4] Angelini, P., Frati, F., Geyer, M., Kaufmann, M., Mchedlidze, T., Symvonis, A.: Upward geometric graph embeddings into point sets. In: Brandes, U., Cornelsen, S. (eds.) Graph Drawing. LIPIcs, vol. 6502, pp. 25–37. Springer (2010)

[5] Arseneva, E., Cano, P., Kleist, L., Mchedlidze, T., Mehrabi, S., Parada, I., Valtr, P.: Upward point set embeddings of paths and trees. In: International Conference and Workshops on Algorithms and Computation. pp. 234–246. Springer (2021)

[6] Bagheri, A., Razzazi, M.: Planar straight-line point-set embedding of trees with partial embeddings. Information Processing Letters **110**(12), 521–523 (2010)

[7] Binucci, C., Di Giacomo, E., Didimo, W., Estrella-Balderrama, A., Frati, F., Kobourov, S.G., Liotta, G.: Upward straight-line embeddings of directed graphs into point sets. Computational Geometry **43**(2), 219–232 (2010)

[8] Bose, P.: On embedding an outer-planar graph in a point set. Computational Geometry **23**(3), 303–312 (2002)

[9] Bose, P., McAllister, M., Snoeyink, J.: Optimal algorithms to embed trees in a point set. In: Brandenburg, F.J. (ed.) Graph Drawing. pp. 64–75. Springer (1996)

[10] Cabello, S.: Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. Journal of Graph Algorithms and Applications **10**(2), 353–363 (2006)

[11] De Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica **10**(1), 41–51 (1990)

[12] Di Giacomo, E., Didimo, W., Liotta, G., Meijer, H., Wismath, S.K.: Point-set embeddings of trees with given partial drawings. Computational Geometry **42**(6), 664–676 (2009)

[13] Di Giacomo, E., Didimo, W., Liotta, G., Wismath, S.K.: Book embeddability of series–parallel digraphs. Algorithmica **45**(4), 531–547 (2006)

[14] Halton, J.H.: On the thickness of graphs of given degree. Inf. Sci. **54**(3), 219–238 (1991)

[15] He, X.: On floorplans of planar graphs. In: Symposium on Theory of Computing. pp. 426–435. ACM (1997)

[16] István, F.: On straight-line representation of planar graphs. Acta scientiarum mathematicarum **11**, 229–233 (1948)

[17] Kaufmann, M., Mchedlidze, T., Symvonis, A.: On upward point set embeddability. Computational Geometry **46**(6), 774–804 (2013)

[18] Mchedlidze, T.: Upward planar embedding of an n-vertex oriented path on $O(n^2)$ points. Computational Geometry **46**(8), 1003–1008 (2013)

[19] Pach, J., Gritzmann, P., Mohar, B., Pollack, R.: Embedding a planar triangulation with vertices at specified points. American Mathematical Monthly **98**, 165–166 (1991)

[20] Patrignani, M.: On extending a partial straight-line drawing. In: Healy, P., Nikolov, N.S. (eds.) Graph Drawing. pp. 380–385. Springer (2006)

[21] Schnyder, W.: Planar graphs and poset dimension. Order **5**(4), 323–343 (1989)

[22] Stein, S.K.: Convex maps. American Mathematical Society **2**(3), 464–466 (1951)

[23] Wagner, K.: Bemerkungen zum Vierfarbenproblem. Jahresbericht der Deutschen Mathematiker-Vereinigung **46**, 26–32 (1936)

Optimal Parallel Algorithms for Convex Hulls in 2D and 3D under Noisy Primitive Operations

Michael T. Goodrich* Vinesh Sridhar†

Abstract

In the noisy primitives model, each primitive comparison performed by an algorithm, e.g., testing whether one value is greater than another, returns the incorrect answer with random, independent probability $p < 1/2$ and otherwise returns a correct answer. This model was first applied in the context of sorting and searching, and recent work by Eppstein, Goodrich, and Sridhar extends this model to sequential algorithms involving geometric primitives such as orientation and sidedness tests. However, their approaches appear to be inherently sequential; hence, in this paper, we study parallel computational geometry algorithms for 2D and 3D convex hulls in the noisy primitives model. We give the first optimal parallel algorithms in the noisy primitives model for 2D and 3D convex hulls in the CREW PRAM model. The main technical contribution of our work concerns our ability to detect and fix errors during intermediate steps of our algorithm using a generalization of the failure sweeping technique.

1 Introduction

Fault tolerance is an important issue in parallel algorithm design, since parallelism naturally introduces more opportunities for faults to occur. In addition, applications can involve the use of faulty primitive operations, such as comparisons. It is ideal if our algorithms can tolerate such noisy primitives. Accordingly, in this paper, we are interested in the design of efficient parallel computational geometry algorithms where primitive operations (such as orientation tests) can return the wrong answer with a fixed probability, $p < 1/2$.

Motivation for this model of noise comes, for example, from potential applications involving quantum computing. A quantum computer is used to answer primitive queries, which return an incorrect answer with a fixed probability at most $p < 1/2$; see, e.g., [4, 18, 20]. A second motivation for this noise model comes from distributed computing applications, in which hash functions can be used to reduce communication costs at the expense of introducing comparison errors, such as in work by Viola [31]. For example, in one problem studied by Viola, two participants with n-bit values seek to determine which of their two values is largest. This can be done by a noisy binary search for the highest-order differing bit position. Each search step performs a noisy equality test on two prefixes of the inputs, by exchanging single-bit hash values on prefixes. The result is an $O(\log n)$ bound on the randomized communication complexity of the problem.

A simple observation, made for sorting by Feige, Raghavan, Peleg, and Upfal [13], is that we can use any parallel algorithm based on correct primitives by performing each primitive operation $O(\log n)$ times (either sequentially or in parallel) and returning the majority answer. This guarantees correctness w.h.p., but increases the span or work of the parallel algorithm by a $\log n$ factor. In this paper, we are interested in the design of parallel computational geometry algorithms with noisy primitive operations that are correct w.h.p. without incurring this overhead.

Related Prior Work. The model for fault tolerance has been extensively studied for binary comparisons for 1-dimensional values. However, the only prior work we are aware of studying fault-tolerant computational geometry algorithms does so in the sequential setting. A recent work by Eppstein, Goodrich, and Sridhar [12] use a technique based on random walks in DAGs to develop optimal sequential algorithms for several computational geometry problems in this setting. Their methods seem to be inherently sequential, however, in that their 2D convex hull algorithm uses a noise-tolerant version of the Graham scan algorithm, and their sequential 3D convex hull algorithm applies their random walks technique to a randomized incremental construction (RIC) algorithm, which seems to be a poor candidate for the RIC parallelization techniques of Blelloch, Gu, Shun, and Sun [6, 7], due to their methods needing correct primitive operations. Thus, to achieve efficient parallel algorithms in the noisy primitives model, new approaches are needed.

Work on searching and sorting 1-dimensional data with faulty comparisons originates with Rényi [29], who studied a binary search problem in 1961 where comparisons between two values return the wrong answer independently with probability $p < 1/2$; see also, e.g.,

*University of California, Irvine, goodrich@uci.edu. Research supported in part by NSF grant CCF-2212129.

†University of California, Irvine, vineshs1@uci.edu.

Pelc [25, 26]. In 1994, Feige, Raghavan, Peleg, and Upfal [13] showed how to sort n items in $O(\log n)$ span with $O(n \log n)$ work in the CRCW PRAM model, with high probability,[1] in the same noisy comparison model, and Leighton, Ma, and Plaxton [22] show how to achieve these bounds in the EREW PRAM model (we review the PRAM model of computation in Appendix A). Both algorithms make extensive use of the AKS sorting network [2, 3], however, which implies that their asymptotic performance bounds have very large constant factors. Recently, in a SPAA 2023 paper, Goodrich and Jacob [16] provide an efficient parallel sorting algorithm where comparisons are either persistently or independently faulty without using the AKS sorting network. Since computations cannot be fully correct with persistent errors, we focus in this paper on the design of efficient parallel computational geometry algorithms using Boolean geometric primitives, such as orientation queries or sidedness tests, that randomly return the wrong answer independently of previous queries with probability at most $p < 1/2$ (where p is known).

Our Results. In this paper, we give efficient parallel algorithms for constructing convex hulls of n points in 2D and 3D when primitive orientation tests return the wrong answer with probability $p < 1/2$. We give algorithms in the CREW PRAM model that have $\Theta(\log n)$ span and $\Theta(n \log n)$ work, w.h.p. The lower bounds follow from lower bounds on computing the minimum/maximum of a set of n points [13, 17].

Our algorithms involve parallel divide-and-conquer and novel versions of the *failure sweeping* technique [13, 14, 16, 23], which is potentially of independent interest. The failure sweeping technique is a paradigm for increasing the success probability of a divide-and-conquer parallel algorithm. Unfortunately, when a success probability depends on the input size, the success probability of recursive calls can be too low. The solution, then, is to design an oracle that can test w.h.p. the solutions to the recursive subproblems, and then "sweep" failed subproblems into a memory space for which we can then apply additional parallel resources to solve. The trick is to show that the total size of all such failure subproblems is relatively small so that we can then solve them using a parallel algorithm that would otherwise have an inefficient work bound. In this paper, we show how to adapt the failure sweeping technique to the case of probabilistically noisy geometric primitives.

We believe that our techniques can be adapted to develop parallel algorithms for other computational geometry problems, such as visibility, point-location, etc., in the noisy primitive setting, and more generally to develop parallel divide-and-conquer algorithms where errors are associated with both construction and validation steps.

[1] In this paper, we take "with high probability" (w.h.p.) to mean that the failure probability is at most $1/n^c$, for some constant $c \geq 1$.

2 Preliminaries

Noisy Geometric Primitives. Convex hull algorithms, as well as other geometric algorithms typically rely on primitive operations, such as orientation tests or visibility tests, that output a Boolean value and operate in constant time. In this work, we assume that each such primitive operation has a fixed probability $p < 1/2$ of being incorrect. We assume *non-persistent* errors, meaning that each operation can be modeled as an independent weighted coin flip. As described above, Feige *et al.* [13], observed that we can convert an algorithm that assumes correct primitives into a noise-tolerate one by repeating each noisy primitive operation $O(\log n)$ times and choosing the majority response. Eppstein, Goodrich, and Sridhar [12] call this the *trivial repetition strategy*, and it immediately implies $O(\log^2 n)$-span, $O(n \log^2 n)$-work algorithms for convex hull construction w.h.p., as well as for a variety of other problems. Our goal is to judiciously use this strategy alongside other noise-tolerant algorithms and our novel failure sweeping techniques to improve this to optimal span and work.

As Eppstein *et al.* note, manipulating non-geometric data, such as pointers and metadata, or manipulating already-processed data is not a noisy operation. For example, rebalancing a tree used as a point-location structure for a planar convex hull is not noisy. However, querying that same structure is a noisy operation. Another important non-noisy operation is the prefix sum. In our paper, we use prefix sums to combine array elements either arithmetically or via concatenation. As a result, no geometric primitives are used, and so they are non-noisy.

Path-guided Pushdown Random Walks. The main technical contribution of Eppstein, Goodrich, and Sridhar is called *path-guided pushdown random walks*. This technique is a generalization of the noisy binary search of [29] to apply to any DAG that obeys the following two constraints: (1) each possible query value must be associated with a unique path P down the given DAG and (2) we can verify whether we are on a correct path using a constant number of noisy comparisons.

As a simple example, consider a binary search tree. Constraint (1) is satisfied by properties of binary search. To satisfy constraint (2), we can maintain pointers to two ancestors v_l and v_r at each node v. They are respectively the largest ancestor of v whose value is smaller than v's and the smallest ancestor of v whose value is larger than v's (one of these will be v's parent). The values of v_l and v_r bound all possible values that v and its descendants can take, so in two comparisons we can determine if a query q could be located in v's subtree, i.e., if we are going down the correct path.

In general, the authors call this second step devising a *transition oracle* and do so for a variety of DAGs used in the construction of several geometric data structures. They show that, if such a transition oracle exists, then in $\Theta(|P| + \log(1/\varepsilon))$ steps we find our desired answer or determine a non-answer correctly with success probability $1 - \varepsilon$. If the height of our DAG is $O(\log n)$ and our confidence bound is w.h.p. in n, this does not increase search runtimes asymptotically. Technical details for path-guided pushdown random walks can be found in Appendix B.

3 Failure Sweeping with Noisy Primitives

Failure sweeping, formalized by Ghouse and Goodrich [14], is a general technique used to improve confidence bounds on algorithms that either fail or exceed a desired span or work bound with some failure probability $p(n)$, given input size n. Say that our desired bounds for span and work done at a subproblem of size n are $T(n)$ and $W(n)$. Our goal is for the entire algorithm to fail with probability at most $p(n)$, but when we recurse to sufficiently small subproblems of size m, e.g., $m = O(\log n)$, $p(m)$ may be exponentially larger than $p(n)$. If unchecked, errors in small subproblems propagate to the returned answer, meaning the algorithm can only guarantee success probability $1 - p(m)$, rather than the intended $1 - p(n)$. This is exactly the case in the noisy primitives setting. Algorithms resilient to noise run in span $T(m)$ if we wish for them to fail with probability at most $p(m)$. Attempting to have all subproblems fail with probability at most $p(n)$ would cause each subproblem to take span proportional to $T(n)$, leading to an inefficient algorithm. Failure sweeping gives us a way to improve this confidence bound from $1 - p(m)$ to $1 - p(n)$ without incurring this span blow-up.

Let n be the size of a parent subproblem and m the size of its child subproblems. The general failure sweeping framework introduces two algorithms: a failure sweeping algorithm that can determine whether one of n's subproblems of size m is incorrect and a "brute-force" algorithm that can recompute an incorrect subproblem w.h.p. in n. We must sweep all of n's subproblems at every level of recursion, so failure sweeping over all subproblems must not take more than $T(n)$ span and $W(n)$ work. On the other hand, our brute-force algorithm only applies to incorrect subproblems. Thus, brute-force recomputation of a single problem of size m may exceed $T(m)$ and $W(m)$, though brute-force recomputation over all failed subproblems may not exceed $T(n)$ and $W(n)$ w.h.p. in n.

To complete the argument, one must show that not too many subproblems fail with probability $p(n)$. Specifically, if each subproblem of size m fails with probability $p(m)$, we must show that no more than q subproblems, determined in the analysis, fail with probability $1-p(n)$. Then, we apply our brute-force algorithm to compute the q subproblems in span $T(n)$ and total work $\leq W(n)$. This way, all subproblems succeed with probability $1 - p(n)$. Applied inductively, this shows that we can "upgrade" our failure probabilities at each level of recursion without asymptotically increasing span or work. This ultimately allows the algorithm to achieve confidence bound $1-p(n_0)$, where n_0 is the initial problem size.

One challenge in designing failure sweep algorithms is in ensuring that m is not too small and not too large compared to n. If m is too small, then many subproblems are likely to fail, forcing us to recompute too many subproblems. If m is too large, then the cost to recompute even a few subproblems may exceed $W(n)$. Indeed, this second issue arises in our 3D convex hull algorithm. We introduce a novel technique called *size reduction*, which is a preprocessing step that transforms large subproblems of size m into $O(m)$ sufficiently small problems of total size $O(m)$ with probability $1 - p(n)$. We are then able to show that not too many of these smaller subproblems fail, guaranteeing that our total brute-force work is no more than $W(n)$. How small these subproblems must be depends both on $W(n)$, the brute-force algorithm, and $p(m)$.

In the case where $p(n)$ is a high-probability bound in n, size reduction may be applied to any algorithm wherein this issue arises as long as one can efficiently transform a large subproblem into $O(m^\alpha)$ sufficiently small subproblems of total size $O(m)$ with probability $1 - p(n)$ for any fixed constant α. Higher confidence guarantees may tolerate a larger fanout.

4 A Deterministic 2D Convex Hull Algorithm for CREW PRAM

Outline of Our Algorithm. Our algorithm closely follows the $O(\log n)$-span, $O(n \log n)$-work algorithm of [1] and [15], which we review in Appendix C.1. However, we swap operations subject to noise with noise-tolerant replacements. The initial sorting operation is replaced with the parallel noisy sorting of [16]. Upper tangents are computed using the path-guided pushdown random walks technique of Eppstein, Goodrich, and Sridhar [12], which we describe in Appendix B.1. The max find operation used to compute the V_j's and W_j's for each upper hull is replaced with the noise-tolerant version described by Feige, Raghavan, Peleg, and Upfal [13] for EREW PRAM. Note that the final parallel prefix operation is not a noisy operation because it only requires us to compare the indices 1 through $\sqrt{n}$ that we assigned each upper hull after sorting. Also, we note that the base case, when $n \leq 2$, can be computed without noisy comparisons as we simply link the two points with an edge.

Each replacement matches the work and span of their non-noisy counterparts. However, they are only correct with high probability in the size of the current subproblem. At small problem sizes, unchecked errors will compound such that we fail to produce an upper hull of all n points with high probability in n. We apply failure sweeping to detect and fix these errors. As described in Section 3, we will develop a failure sweep and brute-force algorithm and use them to show that the entire algorithm succeeds with high probability in n with optimal span and work.

Failure Sweeping. In this section, we will use failure sweeping to show that we can certify and recompute m subproblems of size m in $O(\log(m^2))$ span and $O(m^2 \log(m^2))$ work with high probability in m^2. This way, we upgrade our failure probability to the size of the recursive step to which we return. For now, we will describe individual noisy operations as failing with probability at most $1/m^c$. Later, we show what value c must take.

Consider a subproblem S of size m. We will show how to detect whether S's upper hull is valid. Some of the points in this subproblem belong to the returned upper hull and some do not (we can label each point with a corresponding status). Of the points that belong to the hull, they perform orientation tests on their neighbors using the trivial repetition strategy such that they fail with probability at most m^{-2c}. This verifies that the hull is convex. For the points not on the hull, we can apply noisy binary search to determine the edge of the upper hull directly above or below them (recall that upper hulls are represented in memory as balanced binary search tree of their points sorted by x-coordinate). Adjusting constant factors will allow us to compute each search with failure probability at most m^{-2c}. A final orientation test can determine whether each non-hull point is below the upper hull—valid—or above the upper hull—invalid. This takes $O(\log(m^2))$ time using the trivial repetition test. In total, testing all m points takes $O(\log(m^2))$ span and $O(m \log(m^2))$ work. Every point reports whether it is correct, and we can use a prefix operation to determine whether S is correct in an additional $O(\log m)$ span and $O(m)$ work. If S is found to be incorrect, we use a brute-force algorithm to recompute it. See Figure 1 for an example of failure sweeping on some upper hull of a subproblem S.

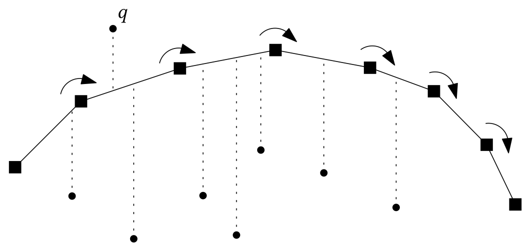


Figure 1: Here we have an example of failure sweeping on some upper hull of a subproblem S. To verify that each (square) hull point is correct, we use the trivial repetition strategy to determine that it is convex with respect to its left and right neighbors. We perform a noisy binary search on the hull to determine if any non-hull points are outside the upper hull. This is true for point q, so this subproblem is flagged for reconstruction.

Brute-force Recomputation. We will show that we can recompute with brute-force a subproblem S of size m with high probability in m^2 using $O(\log(m^2))$ span and $O(m^{3/2} \log(m^2))$ work. We again begin by splitting S into $\sqrt{m}$ groups S_j, each of size $O(\sqrt{m})$. However, we will solve each S_j directly, rather than through recursion.

Let us assume for simplicity that $|S_j| > 6$. First, compute the topmost, leftmost, and rightmost points in each S_j, and call them T_j, R_j, and L_j respectively. L_j and R_j can be recovered in constant time from the fact that the points have already been sorted by their x-coordinate. T_j can be determined using the noisy max-find of Feige *et al.* [13] in $O(\sqrt{m} \log(m^2))$ work and $O(\log(m^2))$ span with failure probability at most m^{-2c}. These three points must be on the upper hull of S_j as they are extreme points in their respective directions. For every other point $p \in S_j$, we find p's clockwise neighbor, i.e., the point r such that the angle between $\overline{pr}$ and a vertical line through p is minimized (this is similar to the "gift-wrapping" process used in the Jarvis march convex hull algorithm [19]). Using the noisy max-find of Feige *et al.* [13] and adjusting constant factors, each max-find fails with probability at most m^{-2c}. We do a similar process to find p's counterclockwise neighbor. Call p's CCW neighbor q and its CW neighbor r.

We can then use T_j, R_j, L_j, q, p, and r to determine whether p is on the upper hull of S_j. With a constant number of orientation tests, we can determine whether either sequence of vertices $[L_j, q, p, r, T_j, R_j]$ or $[T_j, q, p, r, R_j, L_j]$ is a convex chain. Using the trivial repetition strategy, this takes $O(\log(m^2))$ time to guarantee with failure probability at most m^{-2c}. See Figure 2 for a visual of the following lemma.

Lemma 1 *If either* $[L_j, q, p, r, T_j, R_j]$ *or* $[T_j, q, p, r, R_j, L_j]$ *is a convex chain, then* p *is a member of the upper hull of* S_j.

Proof. See Appendix D.2. □

To complete the upper hull for S_j, we simply perform a parallel prefix operation on the points, ordered by their x-coordinate. Points that were found to not be on the hull contribute nothing to the prefix operation. Recall that S_j is just one of $\sqrt{m}$ upper hulls in the subproblem S that we wished to recompute. To compute the upper hull over all points in S, we simply use the original recombination method of [1, 15], with the corresponding noise-tolerant upper tangent and max-find algorithms described above. Again, parameters can be set such that each of those operations fails with probability no more than m^{-2c} with only a constant factor increase in runtime.

It is clear that computing upper hulls for all S_j takes $O(\log(m^2))$ span, as each point $p \in S_j$ can perform its own computations independently in $O(\log(m^2))$ time and all S_j can be processed independently of each other. The work to compute the upper hull of S_j associated with p is performing two max-finds on $O(\sqrt{m})$ points to retrieve q and r plus the work to perform the orientation tests to verify whether p is a hull point or not. This takes $O(\sqrt{m} \times O(\log(m^2)))$ work. Then over all m points, total work is $O(m^{3/2}\log(m^2))$. A constant number of noisy operations are performed per point, so this takes $O(m)$ noisy operations in total.

The remaining operations, a max-find to find T_j for each S_j, a parallel prefix operation to create the upper hull for S_j, and the recombination step for each S_j all take $O(\log(m^2))$ span and $O(m\log(m^2))$ work in total, using $O(m)$ noisy operations.

Lemma 2 *We can failure-sweep an upper hull of m points in $O(\log(m^2))$ span and $O(m\log(m^2))$ work using $O(m)$ noisy operations. We can compute an upper hull in $O(\log(m^2))$ span and $O(m^{3/2}\log(m^2))$ work using $O(m)$ noisy operations. Both hold w.h.p. in m^2.*

High-Probability Bounds for Our Convex Hull Algorithm. Our subproblem S of size m returns to a subproblem S' of size m^2. To maintain the same work recurrence as the original algorithm, subproblem S' must limit its work to $O(m^2\log(m^2))$. We have the "budget" to sweep each subproblem S. However, since recomputing a single subproblem takes $O(m^{3/2}\log(m^2))$ work, recomputing all m subproblems of size m would take $O(m^{5/2}\log(m^2))$ work.

With some arithmetic, we can bound the number of subproblems S that can fail such the we can afford to recompute them all by brute-force.

Corollary 3 *During the recombination step of subproblem S', at most $O(m^{1/3})$ subproblems can fail such that we take $O(m^2\log(m^2))$ work to recompute them.*

Proof. See Appendix D.2. □

Applying our failure-sweeping and brute-force algorithms inductively gives us the following.

Theorem 4 *We can compute the convex hull of n points in the plane in $O(\log n)$ span and $O(n\log n)$ work w.h.p. in n, even with noisy primitive operations.*

Proof. See Appendix D. □

5 A Randomized 3D Convex Hull Algorithm for CREW PRAM

See Appendix C.2 for a review of the non-noisy algorithm of Reif and Sen [27]. As before, there are three steps to our analysis. We first show how to adapt all operations of the original algorithm such that they are noise-tolerant w.r.t. their input size. Then we show how to efficiently perform failure-sweeping and brute-force reconstruction. The first two steps are intricate but straightforward, primarily using old techniques. We defer details to Appendix E and, in particular, prove the following.

Lemma 5 *The intersection of a set of n halfspaces can be computed in $O(\log n)$ span and $O(n^4\log n)$ work with high probability in n under noisy comparisons.*

Theorem 6 *Any parent subproblem of size M can failure sweep all of its child subproblems in $O(M\log M)$ work and $O(\log M)$ span w.h.p. in M. This uses $O(M)$ noisy operations.*

Here, we focus on brute-force construction, for which we apply our generalization of the failure sweep paradigm.

Efficient Brute-Force Reconstruction via Size Reduction. Attempting to apply the strategy that bounds the number of failed subproblems that we used for 2D convex hulls presents a problem. Recall from Reif and Sen's original work [27] as well as Clarkson [8], if the problem at recursive level $l-1$ is size M, then its child problems at recursive level l could be size at most $O(M^{7/8}\log M)$. Indeed, if we were to apply Lemma 5 to solve just a single large subproblem by brute-force, it would take $O(M^{7/2}\log M)$ work to solve it w.h.p. in M. In this section, we develop a novel technique we call *size reduction* to decompose these larger subproblems into problems that are small enough to recompute if needed but are large enough to have similar failure probabilities.

First, however, we observe that sufficiently small subproblems can be trivially recomputed via Lemma 5.

Lemma 7 *We can recompute the values of all subproblems of size $m_s = O(M^{1/8})$ in $O(M\log M)$ work and $O(\log M)$ span w.h.p. in M.*

Proof. Recall that we choose a sample of size $|R| \leq M^{1/8}$ from which we constructed our cones. This implies that there are $O(M^{1/8})$ cones in our sampled hull, each of which induces one subproblem. The total size of all subproblems of size $O(M^{1/8})$ is at most $O(M^{1/8}) \times O(M^{1/8}) = O(M^{1/4})$. Applying the naive brute-force strategy of Lemma 5 gives us the required bound. ☐

To illustrate our approach to size reduction, consider the following algorithm to compute a halfspace intersection of m halfspaces w.h.p. in M. We can perform the original algorithm with the following two modifications. (1) Each recursive level must succeed with high probability in M, so no matter what the current problem size is we tune our noisy operations and sampling probabilities to fail with probability $\leq 1/M$. (2) Irrespective of the current problem size, always choose a sample of size $O(m^{1/8})$.

Each recursive level now takes span $O(\log M)$, total work $O(m \log M)$, and uses $O(m)$ noisy operations. However, because our samples are larger, we reduce the maximum subproblem size by roughly $O(m^{1/8})$ at each level, meaning that it takes at most eight levels of recursion such that the largest problem size is $O(m^{1/8})$. When we get to this level, we simply apply Lemma 5 to solve them w.h.p. in M by brute-force. There are at most $O(m^{1/8})^8 = O(m)$ such subproblems, so it takes $O(\log M)$ span and $O(m^{3/2} \log M)$ work to do this last step. Because all operations are done w.h.p. in M, there is no need to failure sweep these solutions. We simply recombine them as we return from the eight levels of recursion in an additional $O(\log M)$ span and $O(m \log M)$ work per level.

Lemma 8 *The intersection of a set of m halfspaces can be computed in $O(m^{3/2} \log M)$ work and $O(\log M)$ span with high probability in M. This takes $O(m^{3/2})$ noisy operations.*

We can modify this algorithm to instead operate as a preprocessing step prior to our initial recursion. For every subproblem of size $m = \omega(M^{1/8})$, perform the above steps for at most eight levels of recursion until we have decomposed m into at most $O(m)$ subproblems of size $O(M^{1/8})$. Then we simply recurse on these induced subproblems, tuning parameters such that they each succeed high probability in $M^{1/8}$.

Recall that there are at most $O(M^{1/8})$ original subproblems of M. In total, we induce at most $O(M^{1/8} \times M) = O(M^{9/8})$ more subproblems.

Lemma 9 *Say that we have $O(M^{9/8})$ processes, each of which fail with high probability in $M^{1/8}$. Then at most a constant number of them fail with high probability in M.*

Proof. See Appendix E.3. ☐

Theorem 10 *We can recompute all failed subproblems w.h.p. in M in $O(\log M)$ span and $O(M \log M)$ work.*

Proof. Follows from Theorem 6, Lemma 9, and an application of Lemma 8 to any failed subproblems. We note that the added subproblems induced by size reduction does no affect the result of Theorem 6, because total problem size over all subproblems is still $O(M)$ by pruning. The problems at the lowest level of this subproblem are of size at most $O(M^{1/8})$. Then, the cost to recompute a constant number of such subproblems with high-probability in M is $O(\log M)$ span and $O((M^{1/8})^{3/2} \log M) \subset O(M \log M)$ work. ☐

We observe that these are loose worst-case bounds on the number of problems and their size. In reality, pruning guarantees that each level of our recurrence, even the ones that are a part of size reduction, always sum up to $O(n)$. This implies that our induced subproblems all sum to $O(m)$ for each large subproblem of M. Therefore, even if we increase the number of subproblems that we recurse into, our total work does not increase because total work is proportional to problem size.

Now we are ready to prove that failure sweeping, brute-force recomputation, and size reduction allow us to compute the intersection of n halfspaces in optimal span and work w.h.p. in n.

Theorem 11 *We can compute the intersection of n halfspaces in $O(\log n)$ span and $O(n \log n)$ work w.h.p. in n, even with noisy primitive operations.*

Proof. See Appendix E.3. ☐

6 Discussion

In this paper, we have shown how to use our generalization of the failure sweeping technique to compute convex hulls in 2D and 3D in optimal span and work with high probability in n. We believe that our techniques can be adapted to solve more geometric problems in the parallel, noisy primitives model, such as general planar point location, visibility, and convex hulls in higher dimensions. We also hope that our new size reduction technique inspires future work in other settings in which subproblems fail with some probability as a function of their input size.

Acknowledgements

We thank Ryuto Kitagawa for helpful suggestions on writing and organization.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap. Parallel computational geometry. *Algorithmica*, 3:293–327, 1988.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *15th ACM Symp. on Theory of Computing (STOC)*, pages 1–9, 1983.

[3] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983.

[4] J. Allcock, J. Bao, A. Belovs, T. Lee, and M. Santha. On the quantum time complexity of divide and conquer, 2023.

[5] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Parallel algorithms for higher-dimensional convex hulls. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 683–694. IEEE, 1994.

[6] G. E. Blelloch, Y. Gu, J. Shun, and Y. Sun. Parallelism in randomized incremental algorithms. *Journal of the ACM (JACM)*, 67(5):1–27, 2020.

[7] G. E. Blelloch, Y. Gu, J. Shun, and Y. Sun. Randomized incremental convex hull is highly parallel. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 103–115, 2020.

[8] K. L. Clarkson. Applications of random sampling in computational geometry, ii. In *Proceedings of the fourth annual symposium on Computational geometry*, pages 1–11, 1988.

[9] R. Cole. Searching and storing similar lists. *Journal of Algorithms*, 7(2):202–220, 1986.

[10] R. Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.

[11] D. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM Journal on Computing*, 5(2):181–186, 1976.

[12] D. Eppstein, M. T. Goodrich, and V. Sridhar. Computational geometry with probabilistically noisy primitive operations, 2025.

[13] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994.

[14] M. R. Ghouse and M. T. Goodrich. In-place techniques for parallel convex hull algorithms (preliminary version). In *Proceedings of the third annual ACM symposium on Parallel algorithms and architectures*, pages 192–203, 1991.

[15] M. T. Goodrich. *Efficient Parallel Techniques for Computational Geometry.* PhD thesis, Purdue University, 1987.

[16] M. T. Goodrich and R. Jacob. Optimal parallel sorting with comparison errors. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '23, page 355–365, New York, NY, USA, 2023. Association for Computing Machinery.

[17] N. Gupta and S. Sen. Optimal, output-sensitive algorithms for constructing planar hulls in parallel. *Computational Geometry*, 8(3):151–166, 1997.

[18] K. Iwama, R. Raymond, and S. Yamashita. General bounds for quantum biased oracles. *IPSJ Digital Courier*, 1:415–425, 2005.

[19] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information processing letters*, 2(1):18–21, 1973.

[20] K. Khadiev, N. Savelyev, M. Ziatdinov, and D. Melnikov. Noisy tree data structures and quantum applications. *Mathematics*, 11(22), 2023.

[21] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[22] T. Leighton, Y. Ma, and C. G. Plaxton. Breaking the $\Theta(n \log^2 n)$ barrier for sorting with faults. *J. Comput. System Sci.*, 54(2):265–304, 1997.

[23] Y. Matias and U. Vishkin. Converting high probability into nearly-constant time—with applications to parallel hashing. In *Proceedings of the twenty-third annual ACM symposium on Theory of Computing*, pages 307–316, 1991.

[24] M. H. Overmars and J. Van Leeuwen. Maintenance of configurations in the plane. *Journal of computer and System Sciences*, 23(2):166–204, 1981.

[25] A. Pelc. Searching with known error probability. *Theoret. Comput. Sci.*, 63(2):185–202, 1989.

[26] A. Pelc. Searching games with errors—fifty years of coping with liars. *Theoret. Comput. Sci.*, 270(1-2):71–109, 2002.

[27] J. H. Reif and S. Sen. Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems. *SIAM Journal on Computing*, 21(3):466–485, 1992.

[28] J. H. Reif and S. Sen. Optimal randomized parallel algorithms for computational geometry. *Algorithmica*, 7(1):91–117, 1992.

[29] A. Rényi. On a problem in information theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 6:515–516, 1961.

[30] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986.

[31] E. Viola. The communication complexity of addition. *Combinatorica*, 35(6):703–747, 2015.

A Review of PRAM and Work-Span Analysis

In this work, we use the PRAM (Parallel Random Access Machine) model. This parallel model assumes that an algorithm has access to unbounded shared memory which is accessed concurrently at each step of computation. There are variants of the PRAM that specify whether or how processors are allowed to access the same memory location concurrently. The model most prominently used in this work is the CREW (Concurrent Read, Exclusive Write) PRAM, which allows processors to read concurrently but prohibits concurrent writes. Other models we mention are the EREW (Exclusive Read, Exclusive Write) PRAM, which prohibits any concurrent access and the CRCW (Concurrent Read, Concurrent Write), which allows both concurrent reads and writes.

We use the work-span framework to analyze our parallel algorithms. Assume that a parallel algorithm performs all of its operations in s parallel steps. Parallel steps happen sequentially, i.e., all processors are at the same step at the same time. In each step, each processor may perform $O(1)$ constant-time operations. Let w_i be the number of operations performed in step i. We call s the *span* of the algorithm and the total number of operations performed, i.e., $\sum_{i=1}^{s} w_i$, the *work* of the algorithm.

B Review of Path-guided Pushdown Random Walks

The path-guided pushdown random walks technique was first described by Eppstein, Goodrich, and Sridhar in [12]. The technique is a generalization of noisy binary search to search on DAGs in which noisy primitives are used to determine which node to traverse to next. They require that any query value has a unique valid path P in the DAG. At its core, their search is a backtrack search. We maintain a stack S of nodes visited and, if we determine we are on the wrong path, our next step is to backtrack by popping the first vertex off of S. Their unique contribution comes from how to determine if we are on the incorrect path. They model this process using what they call a *transition oracle* T. Given a vertex v, the job of the transition oracle is to determine the next vertex in the DAG to move to, or to determine whether to backtrack. However, with some fixed probability $p_e < 1/2$, the transition oracle T "lies" and returns an arbitrary answer (with the restriction that any vertex it returns must be an outgoing neighbor of v or the vertex at the top of S). If the transition oracle acts truthfully and $v = t$, our goal node, then the transition oracle pushes a copy of t to the stack, an extra insurance to prevent us from deviating.

Assuming that, when T is truthful, it can determine whether we are on the correct path and the next node to advance to using $O(1)$ primitive operations, Eppstein *et al.* prove the following theorem.

Theorem 12 (Theorem 1 of [12]) *Given an error tolerance $\varepsilon = n^{-c}$ for $c > 0$, and a DAG, G, with a path, P, from a vertex s to a distinct vertex t, the path-guided pushdown random walk in G starting from s will terminate at t with probability at least $(1-\varepsilon)$ after $N = \Theta(|P| + \log(1/\varepsilon))$ steps, for a transition oracle, T, with error probability $p_e < 1/15$.*

The basic idea of the proof is that the design of the transition oracle allows each truthful action to undo each action done when T acts arbitrarily. Because $p_e < 1/2$, the good actions outweigh the bad. The authors show that, using a Chernoff bound, we reach our goal node t after N steps and terminate after a further $O(\log(1/\varepsilon))$ steps. The remainder of their work describes how to determine if a query is on the correct path in a DAG in $O(1)$ primitive operations, as this implies that a transition oracle exists for that DAG. This is not trivial to do. For example, it is not obvious how to construct transition oracles for tree-based priority queues. Nevertheless, they were able to describe transition oracles for a variety of geometric DAGs.

B.1 A Transition Oracle for Computing the Upper Tangent of Two Convex Hulls.

Our parallel 2D convex hull algorithm requires us to compute several upper tangents between pairs of upper hulls in parallel. The non-noisy convex hull algorithms of [15] and [1] do this by applying the sequential double binary search technique of Overmars and van Leeuwen [24] to every pair of upper hulls. To make this operation noise-tolerant, we apply the noise-tolerant upper-tangent algorithm developed by Eppstein, Goodrich, and Sridhar [12] using path-guided pushdown random walks.

In the original algorithm of [24], each upper hull is represented as a balanced binary tree of the hull points, sorted by x-coordinate. Their algorithm repeated attempted to draw a tangent line from one hull to the other at the current two points we are considering. Depending on how that tangent line intersected the hulls, Overmars and van Leeuwen developed an analysis of ten cases determining how to recurse on each binary tree at each step. Each step always involves us going deeper in at least one of the trees, therefore the algorithm takes $O(\log n)$ time, assuming that both trees have depth $O(\log n)$.

To apply path-guided pushdown random walks, Eppstein *et al.* considers the implicit decision tree created by the algorithm. Each node represents a state of the algorithm, and each child represents choosing one of ten cases to advance one of the two trees. This decision tree has depth $O(\log n)$ for the same reason the algorithm takes $O(\log n)$ time. They show that, using four invocations of Overmars and van Leeuwen's case analysis, we can verify if we are on the correct path of this decision tree. For more details, see [12] (Section 6.2).

C Review of Non-noisy Algorithms

In this section, we review the classic works on which we base our noise-tolerant algorithms.

C.1 2D Convex Hull Algorithm for CREW PRAM.

Here we review the $O(\log n)$-span, $O(n \log n)$-work parallel convex hull algorithm independently discovered by Aggarwal, Chazelle, Guibas, Ó'Dúnlaing and by Goodrich and Atallah [1,15]. Note that it is sufficient to describe a convex hull algorithm that computes the upper hull, as a symmetric algorithm can compute the corresponding lower hull of a set of points.

We first sort all n points by their x-coordinate in $O(\log n)$ span and $O(n \log n)$ work, e.g., using Cole's merge sort [10]. Then we partition them into $\sqrt{n}$ groups of $\sqrt{n}$ points. This partition is done by their index, e.g., points 1 through $\sqrt{n}$ in the sorted order belong in group 1, and so on. We recursively solve each group in parallel, and each group ultimately returns with an upper hull of their $\sqrt{n}$ points.

Once we have $\sqrt{n}$ upper hulls, we can combine them in parallel. We first compute an upper tangent for every pair of upper hulls using the double binary search of Overmars and van Leeuwen [24].

We next determine the contribution of each upper hull U_j to the combined upper hull of all n points. To do this, we compute V_j, the tangent of smallest slope between U_j and some other U_i where $i < j$. Likewise, we compute W_j, the tangent of largest slope between U_j and some other U_k where $k > j$. Let v_j and w_j be the corresponding tangent points on U_j. If the angle between V_j and W_j around their intersection point pointing upward is $\geq 180°$, then points from v_j to w_j on U_j are on the combined upper hull. Otherwise, U_j contributes no points to the upper hull. The authors' rationale for choosing V_j and W_j is that U_i and U_k are the two hulls that are best able to "cover up" U_j such that it does not contribute to the combined upper hull. Their respective tangents with U_j indicate that they are the "highest" hulls on each side of U_j. Finally, a parallel prefix operation is used to combine contributions from each U_j into the combined upper hull.

Now we discuss the span and work performed at each level of recursion. There are $\binom{\sqrt{n}}{2} = O(n)$ pairs of upper hulls. The double binary search method of Overmars and van Leeuwen [24] takes $O(\log n)$ time sequentially. If we perform each tangent computation in parallel, this takes $O(\log n)$ span and $O(n \log n)$ work. To compute W_j, we perform a max-find operation on the $O(\sqrt{n})$ tangents that touch U_j. Doing this in parallel over all U_j takes $O(\log n)$ span and $O(n \log n)$ work. The same cost applies to computing V_j. It then takes constant span and $O(\sqrt{n})$ work to determine the convex chain each U_j contributes to the combined hull. The final parallel prefix operation of the $O(\sqrt{n})$ convex chains takes $O(\log n)$ span and $O(\sqrt{n})$ work.

For $n \leq 2$, span and work are both constant as the convex hull is simply a line segment between the two points. We conclude that the total span and work for the algorithm are given by the following two recurrences:

$$T(n) = T(\sqrt{n}) + O(\log n)$$
$$W(n) = \sqrt{n}W(\sqrt{n}) + O(n \log n)$$

With $T(2), W(2) = O(1)$. Observe that both recurrences can be bounded by a geometric series.

$$T(n) \leq O(\log n) \times \sum_{i=0}^{\infty} 1/2^i = O(\log n)$$

$$W(n) \leq O(n \log n) \times \sum_{i=0}^{\infty} 1/2^i = O(n \log n)$$

The initial sorting takes $O(\log n)$ span and $O(n \log n)$ work [10], so the algorithm takes $O(\log n)$ span and $O(n \log n)$ work overall in the CREW PRAM model [1, 15].

C.2 3D Convex Hull Algorithm for CREW PRAM.

We adapt our algorithm mostly from the optimal $O(\log n)$-span and $O(n \log n)$-work randomized 3D convex hull algorithm of Reif and Sen [27]. First, we note that Reif and Sen solve the dual problem of halfspace-intersection of n halfspaces in 3D. This allows them to take advantage of an earlier result of Clarkson [8], who showed that a random sample of halfspaces R can be used to divide the remaining $H \setminus R$ halfspaces into subproblems of roughly equal size, admitting a natural divide-and-conquer algorithm. Specifically, these good samples divide the remaining halfspaces into groups with the following two properties: (1) the largest group has size at most $O(|H| \log |R|/|R|)$ and (2) the sum over all groups is of size $O(|H|)$. However, Clarkson only showed that this is the case with constant probability $> 1/2$.

To turn this sampling probability into a high-probability bound, Reif and Sen introduce the notion of *polling*. Rather than one sample R, they take $O(\log n)$ samples R_j and test if any are "good" as defined by Clarkson. It would be too expensive to evaluate each R_j, so instead of sampling each from H, they first sample $n/\log^d n$ elements from H into H_j, for $d > 2$. Then they sample R_j from H_j. By shaving this polylog factor, Reif and Sen show that we can evaluate whether each R_j is a good sample for H_j in $O(\log n)$ span and $O(n \log n)$ work. Through applications of Chernoff bounds, they show that (1) at least one R_j is good with respect to H_j, and (2) R_j is a good sample for the original H as well, despite the polylog factor decrease. Both occur with high probability in n. Note that if a good sample was not found, we could simply spend the same amount of span and work to repeat the process.

Say that we have found a good sample R. In the process to determine whether R was good, Reif and Sen use a non-optimal halfspace intersection algorithm to compute the polyhedron $\mathcal{R}$ in $O(\log n)$ span and $O(n \log n)$ work. We can use this structure to define our divide-and-conquer. Observe that the halfspace intersection $\mathcal{C}$ of H can only get smaller than $\mathcal{R}$, thus the subproblems which we recursively solve must involve some partition of the interior of $\mathcal{R}$. Specifically, this is done by considering some point o assumed to be in the halfspace intersection of H. We draw lines from each vertex of $\mathcal{R}$ to o and triangulate the faces of $\mathcal{R}$ such that we have a set of at most $2|R|$ "cones", their a base a triple of vertices of $\mathcal{R}$ and their apex o. Then each recursive problem would involve computing a halfspace intersection of the subset of halfspaces in $H \setminus R$ that intersect some cone C_i.

To determine the cones each halfspace in $H \setminus R$ stabs, the authors consider the dual problem. Taking the dual transform of each vertex of $\mathcal{R}$ gives us an arrangement of hyperspaces. Each cell of the arrangement corresponds to a combination of cones C_i that some halfplane in the primal could intersect. By parallelizing the multidimensional search method of Dobkin and Lipton [11], they show that, as long as R is a sufficiently small sample, $|R| \leq n^{1/8}$, we can construct a search structure in $O(\log n)$ span and $O(n \log n)$ work that allows for fast queries to return the set of cones of $\mathcal{R}$ that a hyperspace h intersects in the primal in. Note that this sample guarantees that a subproblem can be as large as $O(n^{7/8} \log n)$.

Lastly, Reif and Sen observe that, despite our sample R being "good", it is not good enough to prevent a blowup in problem size as we go down recursive levels. This is because halfspaces may intersect with multiple cones. If X_i is the number of halfspaces of $H \setminus R$ that intersect cone C_i, then Clarkson showed that $\sum_i X_i = O(n)$ [8], meaning the total problem size may increase by a constant factor every level of recursion. There are $O(\log\log n)$ levels of recursion, so this constant factor blow-up implies total problem size may expand to $O(n \log^{O(1)} n)$, increasing work by a polylog factor. Thus, their final step before recursion involves pruning redundant halfspaces to ensure that total problem size at each level of recursion is no more than $O(n)$.

In their work, Reif and Sen classify different types of redundancies and describe a two-step approach to pruning the halfspaces of cone C_i, first by applying a 3D maxima algorithm and then by computing a simplified 3D hull they call a skeletal hull. Information from these outputs can be used to narrow down only the halfspaces that both intersect C_i and contribute a vertex to the final halfspace intersection $\mathcal{C}$ that lies in C_i.

However, a later work by Amato, Goodrich, and Ramos [5] provides a simpler way to prune. They show that we can perform a sufficient amount of pruning from information given to us by the *contours* of C_i. A contour is the result of projecting the halfspaces that intersect C_i onto the three faces of the cone incident to the apex o and then computing the 2D convex chain of the projection on each face. Given these contours, they show that a constant number of binary searches per halfspace and one sorting step per cone can be used to prune a sufficient number of halfspaces. By the fact that the total problem size among all cones is $O(n)$, we can compute contours for all cones with $O(\log n)$ span and $O(n \log n)$ work. In addition, Amato, Goodrich, and Ramos show that it takes $O(\log n)$ span and $O(n \log n)$ work to prune given these contours.

Finally, we recursively solve each subproblem defined by each cone C_i. Each subproblem returns with the subset of vertices of the halfspace intersection $\mathcal{C}$ located in that subproblem, from which an adjacency structure can be computed by an application of sorting [27] (Lemma 2.2). Once the subproblem size is smaller than $O(\log^k n)$ for some constant k, the authors can apply a non-optimal time algorithm (for example the $O(\log^3 n)$-span, $O(n \log^3 n)$-work algorithm of Aggarwal, Chazelle, Guibas, Ó'Dúnlaing, and Yap [1]) to solve the problem directly without increasing total asymptotic complexity. Reif and Sen conclude that the span for their algorithm is bounded by following recurrence:

$$T(n) \leq T(n^{9/10}) + O(\log n)$$

The $n^{9/10}$ comes from the analysis of Clarkson, who showed that with a good sample, no cone has more than $O(n^{7/8} \log n) = O(n^{9/10})$ halfspaces that stab it. Observe that the recurrence is the summation of a converging geometric sequence, so $T(n) = O(\log n)$. For the total work, we simply observe that the pruning step guarantees that each level of recursion has total problem size $O(n_0)$, where n_0 is the initial problem size. Given that the span of the algorithm at each level of recursion is $O(\log n)$, we can upper bound the work like so:

$$W(n) \leq W(n^{9/10}) + O(n_0) \times O(\log n)$$

We again have a geometric series, so $W(n_0) = O(n_0 \log n_0)$. We conclude that, given a set of n halfplanes in 3D, Reif and Sen's algorithm computes their halfspace intersection in $O(\log n)$ span and $O(n \log n)$ work for the CREW PRAM model [27].

D Deferred Details for Our 2D Convex Hull Algorithm

D.1 Omitted Proofs

Now we are ready to prove inductively that our algorithm succeeds with high probability in n and, in doing so, find the appropriate constant c determining the probability that an individual noisy operation fails.

Theorem 13 (Same as Theorem 4) *We can compute the convex hull of n points in the plane in $O(\log n)$ span and $O(n \log n)$ work w.h.p. in n, even with noisy primitive operations.*

Proof. We will prove this by induction. Our inductive hypothesis is that we can compute a subproblem of size m in $O(\log m)$ span and $O(m \log m)$ work with failure probability no more than $m^{-c'}$. When $m \leq 2$, this holds true because no noisy comparisons are needed to draw an edge between two points.

Consider a subproblem S' of size m^2. To complete the proof, we will show that it can be solved in $O(\log(m^2))$ span and $O(m^2 \log(m^2))$ work with failure probability no more than $m^{-2c'}$. S' must combine m subproblems of size m. We expect $m/m^{c'}$ to fail by our inductive hypothesis, yet we require that no more than $m^{1/3}$ fail by Corollary 3. Let X be a random variable representing the number of subproblems that fail. Markov's inequality states that $\Pr[X \geq m^{1/3}] \leq (m/m^{c'}) \times (1/m^{1/3}) = m^{2/3-c'}$. To be a high probability bound in m^2, $2/3 - c' < -2$, so $c' > 8/3$.

Continuing the inductive argument, we must show that S' succeeds in outputting an upper hull of its points with failure probability no more than $m^{-2c'}$. Notice that every noisy max-find and upper tangent computation performed during brute-force reconstruction and the later combination step individually fails with probability no more than m^{-2c}. To determine the constant c, we count the number of noisy operations done and take a union bound over all of them to compute the probability that at least one fails.

By Lemma 2, sweeping for errors takes $O(m)$ noisy operations total, and each brute-force reconstruction takes $O(m)$ operations total. At most $m^{1/3}$ subproblems fail with high probability in m^2, so brute-force reconstruction requires at most $O(m^{4/3})$ noisy comparisons. Combining each subproblem S into an upper hull over S' requires $O(1)$ noisy max-find and upper tangent operations per point, so $O(m^2)$ operations. In total, we use $O(m^2)$ noisy operations to compute the upper tangent of S'.

Taking the union bound over all $O(m^2)$ invocations, if a single noisy operation fails with probability m^{-2c}, at least one fails with probability $\leq m^{-2(c-1)}$. If we take $c \geq 1 + c' > 11/3$ and adjust constant factors associated with each operation accordingly, computing the upper hull for S' takes $O(\log(m^2))$ span and $O(m^2 \log(m^2))$ work and fails with probability $\leq m^{-2c'}$.

We conclude that our algorithm succeeds with high probability in n, and the total span and work for our algorithm are bounded by the following recurrences:

$$T(n) = T(\sqrt{n}) + O(\log n)$$
$$W(n) = \sqrt{n}W(\sqrt{n}) + O(n \log n)$$

These are the same recurrences that bound the span and work of the original algorithms [1, 15]. They evaluate to $O(\log n)$ span and $O(n \log n)$ work respectively. ☐

D.2 Additional Details

Lemma 14 (Same as Lemma 1)
If either $[L_j, q, p, r, T_j, R_j]$ or $[T_j, q, p, r, R_j, L_j]$ is a convex chain, then p is a member of the upper hull of S_j.

Proof. By definition of CW and CCW neighbors, no points in S_j exist in the region to the left of $\overrightarrow{pr}$ and the the right of the vertical line through p. Likewise, no points in S_j exist in the region to the right of $\overrightarrow{pq}$ and the left of the vertical line through p. Because the two regions border each other and $[q, p, r]$ makes a right-hand turn, this region is the same as the union of the regions of the plane to the left of $\overrightarrow{pr}$ and the right of $\overrightarrow{pq}$. See Figure 2.

In either of the two cases, imagine we rotate the coordinate system such that p has the highest y-coordinate of all $\{T_j, R_j, L_j, p, q, r\}$. By the fact that no points exist to the left of $\overrightarrow{pr}$ and to the right of $\overrightarrow{pq}$, p has the highest y-coordinate of all points in S_j in this new coordinate system. Because the structure of a convex hull is invariant to rotation, p must be on the convex hull of S_j. Returning to the original coordinate system, because either $[L_j, p, T_j]$ or $[T_j, p, R_j]$ is a right-hand turn and the fact that L_j, T_j, and R_j are the leftmost, topmost, and rightmost points, p must be on the upper hull of S_j, not its lower hull. ☐

Corollary 15 (Same as Corollary 3) *During the recombination step of subproblem S', at most $O(m^{1/3})$ subproblems can fail such that we take $O(m^2 \log(m^2))$ work to recompute them.*

Proof. Say that $m^{1/b}$ subproblems fail. To recompute them in parallel takes $O((m^{1/b} \times m)^{3/2} \times \log(m^2))$ $= O(m^{3/(2b)+3/2} \log(m^2))$ work. We can only afford $O(m^2 \log(m^2))$ work at subproblem S', so $3/2b + 3/2 \leq 2$. We have that $b \geq 3$, so at most $O(m^{1/3})$ subproblems can fail such that S' can fix them in $O(m^2 \log(m^2))$ work via the brute-force method described in the previous section. ☐

E Deferred Details for Our 3D Convex Hull Algorithm

E.1 Adapting the Non-Noisy Algorithm to the Noisy Primitives Setting

We begin with an explanation of how to modify the non-noisy algorithm such that, under noisy primitive operations, each step succeeds with high probability in the size of the current subproblem. From an application of the trivial repetition strategy, we immediately have the following algorithms to compute a halfspace intersection. Respectively, these are used to compute the halfspace intersection of our sample R and to directly solve sufficiently small subproblems.

Lemma 16 (Same as Lemma 5) *The intersection of a set of n halfspaces can be computed in $O(\log n)$ span and $O(n^4 \log n)$ work with high probability in n under noisy comparisons.*

Proof. Follows from Lemma 2.2 of Reif and Sen [27]. There are $O(n^3)$ candidate vertices in a 3D arrangement and $O(n)$ halfspaces, so there are $O(n^4)$ vertex-halfspace pairs. For each vertex-halfspace pair, we can determine whether the vertex satisfies the halfspace's equation in a single noisy operation. For each vertex, we perform a parallel prefix operation to compute the conjunction of the $O(n)$ results.

There are $O(n^4)$ pairs, so testing each with the trivial repetition strategy takes $O(n^4 \log n)$ work. Each computation can be done in parallel, so this step takes $O(\log n)$ span. The final prefix operations done at each vertex are not noisy. They take $O(\log n)$ span and $O(n^4)$ work overall.

We are left with a set of vertices in the arrangement of the given halfspaces that satisfy every halfspace. This means these vertices must bound the halfspace intersection. An application of noisy sorting [16] can be used to compute an adjacency structure for the polytope from this set in $O(\log n)$ span and $O(n \log n)$ work [27].

We perform $O(n^4)$ noisy operations in this step. If each individually fails with probability no more than $1/n^c$ for some constant c, then by union bound at least one fails with probability $1/n^{c-4}$. Therefore, to guarantee a failure probability of at most $1/n^{c'}$, we set $c > 5 + c'$. This only affects constant factors in the work and span of these operations and so does not change their overall complexity. ☐

Lemma 17 *The intersection of a set of n halfspaces can be computed in $O(\log^4 n)$ span and $O(n \log^4 n)$ work with high probability in n under noisy comparisons.*

Proof. Follows from the $O(\log^3 n)$-span, $O(n \log^3 n)$-work algorithm of Aggarwal, Chazelle, Guibas, Ó'Dúnlaing, and Yap [1] and the trivial repetition strategy applied to every noisy primitive. There are certainly at most $O(n \log^3 n) = O(n^2)$ primitive operations performed throughout the algorithm. We can replace them all with the trivial repetition strategy. Say that we configure each noisy operation to fail with probability at most n^{-c}. Then, by union bound, one of these comparisons fails with probability at most $n^{-(c-2)}$. If we wish the algorithm to fail with probability no more than $1/n^{c'}$, we set $c \geq 2 + c'$. ☐

We next observe that Reif and Sen's *polling* method [27] in and of itself is not subject to noise as randomly selecting halfspaces requires no primitive operations. However, determining whether a sample is good does involve noisy operations. We now show that computing $\mathcal{R}$, the halfspace intersection of R, and the point-location structure that determines what cones a halfspace in $H \setminus R$ intersects can be performed in $O(\log n)$ span and $O(n \log n)$ work w.h.p.

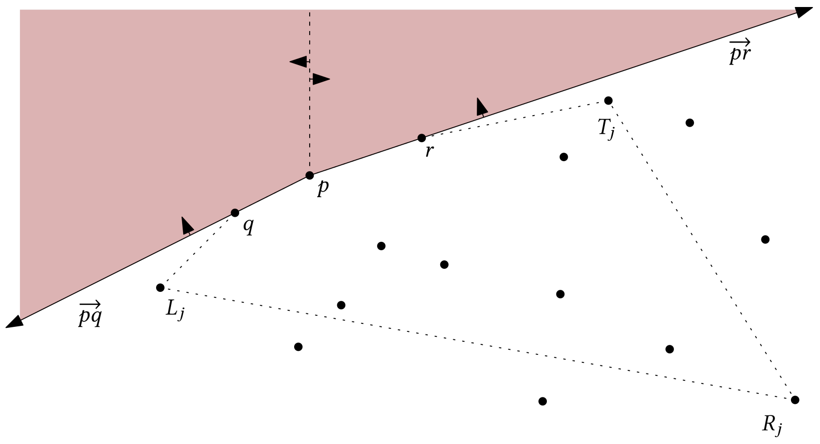


Figure 2: We depict an instance of the failure sweep brute-force algorithm determining that a point, p, is a part of the upper hull in S_j. In this case, $[L_j, q, p, r, T_j, R_j]$ is convex. We have depicted the lines $\overrightarrow{pr}$ and $\overrightarrow{pq}$ as well as the vertical line through p that borders both regions defined by the lines. Notice, as the proof describes, no points in S_j can exist in the region either to the left of $\overrightarrow{pr}$ and to the right of $\overrightarrow{pq}$, depicted with shading. It is clear that p is an extreme point in S_j.

Lemma 18 *Given a point o in the halfspace intersection of H, we can compute the halfspace intersection of R and its cones using $O(|R|^4 \log n)$ work and $O(\log n)$ span under noisy comparisons w.h.p.*

Proof. Follows from Lemma 5. The adjacency structure of R that we compute is correct with high probability in n. Because we have computed this structure, any topological changes we make to produce the cones are not noisy. Thus, we can partition $\mathcal{R}$ with planes such that each face is a trapezoid and partition each trapezoid further into a triangle, and draw edges between each face and our origin o as Reif and Sen describe [27] without noise. □

Lemma 19 *We can preprocess n planes into an arrangement such that, given an arbitrary query point, we can report the cell in which the point is contained. Preprocessing can be done under noisy comparisons in $O(n^7 \log n)$ work and $O(\log n)$ span w.h.p. using $O(n^7)$ noisy operations. Queries can be performed sequentially in $O(\log n)$ time w.h.p. using $O(1)$ noisy operations.*

Proof. This follows from [27] (Lemma 5.1) as well as by the work of [11]. Each preprocessing step involves projection of points down to a lower dimension and a constant number of parallel sorting steps per cell. Projection is not a noisy operation as we are manipulating a geometric object's equations, which are its labels. Each sorting step can be swapped with the noisy sorting algorithm of [16]. Likewise, each query involves a constant number of binary search steps, which can be replaced with the noisy binary search implementation of [13]. □

Corollary 20 *We can compute a preprocessing structure from $\mathcal{R}$ such that, given a query halfspace h, we can determine which cones in $\mathcal{R}$ h stabs. Preprocessing can be performed in $O(|R|^8 \log n)$ work and $O(\log n)$ span w.h.p. using $O(|R|^8)$ noisy operations. Queries can be performed sequentially in $O(\log n)$ time w.h.p. using $O(1)$ noisy operations.*

Proof. Follows from our Lemma 19 and [27] (Lemma 5.2) applied to the dual of each vertex of $\mathcal{R}$. There are $O(|R|^7)$ cells of our arrangement and $O(|R|)$ cones in $\mathcal{R}$. We have each of the $O(|R|^8)$ cone-cell pairs compute whether the cell overlaps the cone in $O(1)$ noisy comparisons. $O(|R|^7)$ parallel prefix operations can be used to compile a list of cones each cell is incident to.

Testing each cone-cell pair takes $O(|R|^8 \log n)$ work and $O(\log n)$ span via the trivial repetition strategy. The parallel prefix operations take $O(|R|^8)$ work and $O(\log |R|) = O(\log n)$ span.

As described in the previous lemma, each query requires a constant number of noisy binary searches and can be accomplished in $O(\log n)$ time sequentially w.h.p. □

Setting $|R| \leq n^{1/8}$ allows us to perform the previous steps in $O(\log n)$ span and $O(n \log n)$ work w.h.p. Once the structure is built, we perform one query per halfspace $h \in H \setminus R$ to determine which cones it stabs. All queries can be done in parallel and there are $O(n)$ halfspaces, so this step takes $O(\log n)$ span and $O(n \log n)$ work. Having done this work, we can use this data structure to determine if our current sample R is "good", i.e. if it distributes halfplanes roughly evenly between cones [8, 27].

E.1.1 Polling work and span bounds

Before we continue, we note that Reif and Sen [27] show that we can adjust parameters such that each instance of polling at recursive depth l fails with probability at most $n^{-(9c/10)^l}$ for some $c > 1$. Let $n_l = n^{(9/10)^l}$, for simplicity. They observe that polling takes work $O(n_l \log n_l)$ and span $O(\log n_l)$. If an instance of polling fails, we can simply repeat the process again. To show that polling takes no more than $O(\log n)$ span overall, Reif and Sen prove the following useful theorem.

Theorem 21 *If a subproblem at recursive depth l takes $O(\log n_l)$ span w.h.p. in n_l, then the entire problem takes $O(\log n)$ span.*

Proof. See [27] (Theorem 2.1) and [28] (Theorem 2). ◻

A work bound follows like so.

Corollary 22 *If total work done by all subproblems at recursive level l is $O(n \log n_l)$ w.h.p. in n_l, then the entire problem takes $O(n \log n)$ work w.h.p. in n.*

In our algorithm, the probability that polling succeeds is the probability that a good sample is found and all noisy operations used to test each sample is correct. Say the probability polling succeeds is $1/n_l^\alpha$ and the probability each individual noisy operations succeeds is $1/n^c$. Because $O(n_l)$ noisy operations are performed to test all samples, by union bound, at least one fails with probability $1/n_l^{c-1}$. Assuming $c > 2$ (requires an adjustment of constant factors for noisy operations), both of these are high-probability bounds.

Then the probability either of these fails is at most $1/n_l^\alpha + 1/n_l^{c-1}$, which is less than $2 \times \max\{1/n_l^\alpha, 1/n_l^{c-1}\}$. This is still a high-probability bound in n_l, so we can invoke Theorem 21 to bound the span of polling even in the presence of noisy primitives.

In the next section, we describe how to prune halfspaces w.h.p. in the size of the current subproblem such that each level of recursion has total problem size $O(n)$. With a similar argument to above we can use Corollary 22 to bound total work of polling to $O(n \log n)$ w.h.p.

E.1.2 Pruning halfspaces

To ensure total problem size among subproblems at a given level of recursion does not exceed some constant multiple of our initial problem size, we rely on a method of Amato, Goodrich, and Ramos [5].

Lemma 23 *In a problem size of n, where H is the total set of halfplanes, we can prune enough redundant halfspaces such that total problem size among all subproblems at this level is at most $O(n_0)$, where n_0 is the initial problem size. This can be done in $O(\log n)$ span and $O(n \log n)$ work w.h.p. in n using $O(n)$ noisy operations.*

Proof. Follows from the pruning strategy and corresponding lemmas described in [5] (Section 4.1.1). Their solution first involves projecting the halfspaces associated with C_i onto the three faces of the cone, which is not a noisy operation. Then they compute the 2D convex hull of each projection, which we can do in $O(\log n)$ span and $O(n \log n)$ work w.h.p. in n using our construction in Section 4. They call these convex chains *contours*.

They consider separately the halfspaces that contribute an edge to a contour and those that do not. We denote the set of halfspaces that stab C_i but do not contribute to any of its contours $H^{nc}_{|C_i}$. Let H^{nc} be the union of all such halfspaces over all cones. The authors show that each halfspace $h \in H^{nc}$ may contribute a vertex to the halfspace intersect of at most one cone [5] (Lemma 4.2). They prove that this halfspace h can be found by locating the closest point p on each of the three contours of C_i and shooting a ray from each p through C_i towards h. If all three such rays pierce h without being stopped first by their respective halfspaces h', h'' that contributed p to the contour, then h may contribute a vertex to the halfspace intersection of H.

Each halfspace $h \in H^{nc}_{|C_i}$ can determine whether it should be pruned in $O(\log n)$ time w.h.p. by projecting itself onto each face and performing a noisy binary searches on each contour to determine the closest hull point to the projected line representing h (recall that we maintain the upper and lower hulls of each contour as a binary search tree, something we took advantage of to perform failure sweeping in Section 4). We can associate each contour point p with the two halfspaces that define it so that they can be recovered in constant time. We can then perform a constant number of noisy comparisons using the trivial repetition strategy to determine which halfspace the ray passes through first in $O(\log n)$ time w.h.p. By the fact that our sample R is good, $|H^{nc}| = O(n)$. Then we can perform each test in parallel in $O(\log n)$ span and $O(n \log n)$ work.

Now we consider the halfspaces that contribute to contours, which we denote as $H^c_{|C_i}$. Amato, Goodrich, and Ramos [5] show that, if a halfspace in $h \in H^c_{|C_i}$ contributes an edge to at least two of the contours, then it can be pruned. To determine which halfspaces should be pruned, we simply label each contour edge with the corresponding halfspace that generated it, which can be done when constructing the contours, and sort the labels lexicographically. Again, by our use of a good sample, the total number of contour halfspaces $|H^c| = O(n)$, thus we can perform noisy sorting in parallel over all C_i in $O(\log n)$ span and $O(n \log n)$ work. In the lexicographically sorted array, if a halfspace is adjacent to itself, then it must contribute an edge to at least two contours, and so should be pruned. In [5], the authors show that after pruning, the set of all contour halfspaces across all cones and all subproblems also has size $O(n_0)$. ◻

Finally, to complete the halfspace intersection before returning from a subproblem, Amato, Goodrich, and Ramos note that we must add back a certain subset of halfspaces in $H^c_{C_i}$ that contributed an edge to the halfspace intersection but not a vertex. Fortunately, they show that at most three such halfspaces exist and can be identified when pruning. We can incorporate these halfspaces like so.

Let $\mathcal{C}_i$ be the halfspace intersection of cone C_i. Process each halfspace one at a time. Have each vertex in $\mathcal{C}_i$ check if it satisfies the halfspace's equation in $O(\log n)$ time using the trivial repetition strategy. Label the vertex based one whether it succeeded or failed. Then examine each edge. For edges that have one vertex included in h and the other vertex not included in h, compute the intersection between h and the edge to produce a new set of vertices that are incident to h.

For each of the three halfspace processed, this takes $O(\log n)$ span and $O(|\mathcal{C}_i| \log n)$ work. Because 3D hulls have complexity linear in their input and $|H^{nc}|, |H^c| = O(n)$, this takes $O(n \log n)$ work over all cones.

We conclude that each operation in a non-noisy 3D hull algorithm can be replaced with noisy operations without increasing the asymptotic complexity of each subproblem. We observe that, for a subproblem of size n, all operations discussed prior use at most $O(n)$ noisy operations, including sorting, searching, and trivial repetitions.

E.2 Failure Sweeping

We have shown that each step of the algorithm of [27], with the pruning method of [5], can be replaced with noise-tolerant versions that individually succeed with high probability in the current problem size. However, as we did for 2D convex hulls, we must negotiate the fact that our success probabilities get weaker as problem size gets smaller. To compensate for this, we once again perform failure sweeping. Unlike for 2D convex hulls, subproblems may not be evenly sized. We will denote m as the size of the problem at recursive level l and M the size of its parent subproblem at recursive level $l-1$.

Once again, we proceed in two steps. We first determine if the provided polyhedron is convex and then determine whether it is a valid halfspace intersection, both w.h.p. in M. By general position, each vertex of this polyhedron has at most three edges incident to it and so has at most three neighboring vertices. We can verify that the polyhedron is convex by having each vertex perform one noisy 3D orientation test against its neighbors using the trivial repetition strategy. This takes $O(\log M)$ time per vertex to succeed w.h.p. in M. There are m vertices, so total work done is $O(m \log M)$ and span is $O(\log M)$. A prefix operation taking $O(m)$ work and $O(\log m)$ span can combine results over all $O(m)$ vertices and determine whether the intersection is convex or not.

Now we must ensure that this polyhedron satisfies all halfspaces assigned to this subproblem. It will be simpler to consider the equivalent dual problem: given a 3D convex hull, verify that all points on the hull are convex with respect to their neighbors and all other points lie within the hull. We have verified the first condition with our initial tests in the primal. When we performed failure sweeping on a 2D hull, we were able to quickly determine which edge of the hull lies directly above each point through noisy binary search on the tree-based structure that represented the hull. In this algorithm, however, no equivalent structure is built. As a result, we will show how to reduce the problem to planar point location and then describe an algorithm that can construct a PPL data structure efficiently under noisy comparisons.

Consider a coordinate system in which the origin lies outside of the hull, and consider the plane p formed by the x- and y-axes. Let the "upper hull" be the portion of the hull that can see this plane and the "lower hull" be the portion that cannot. This test can be done via an constant number of primitive operations per face, and so can be performed in parallel in $O(\log n)$ span and $O(n \log n)$ work using the trivial repetition strategy.

We focus on the upper hull, as the case for the lower hull is symmetric. We can project each point of the upper hull simply by ignoring its z-coordinate. By general position, each face of our hull is a triangle. Therefore, this projection is a triangulation of a convex polygon. We can use this projection to determine if some non-hull point u lies below the upper hull. First, we project u onto p, creating the projected point u^*. We use planar point-location to determine the triangle Δ^* it appears in. This triangle maps to a face of the upper hull, Δ, that intersects the ray $\overrightarrow{uu^*}$. Next, we simply perform an orientation test between u and Δ to determine if u is inside or outside of the upper hull. We can repeat this point location test symmetrically on the lower hull. If any u fails either test, then we conclude that our convex hull is invalid.

E.2.1 Constructing a Planar Point-Location Data Structure on a Triangulated Convex Polygon

While the projections done to produce our triangulated convex polygon on plane p are without noise, the work needed to construct the planar point location data structure does involve noisy primitive operations.

To build a PPL data structure, we modify the randomized parallel algorithm of Reif and Sen [28] that constructs a Kirkpatrick decomposition [21] of a triangulated polygon of n points taking $O(\log n)$ span and $O(n \log n)$ work with high probability in n. Their basic strategy is to, at each step, have each vertex of degree $\leq d = O(1)$ independently flip a coin. Vertices that flip heads and have no neighbors that also flip heads are pruned. The gaps left behind are re-triangulated. While constructing the decomposition, Reif and Sen at the same time construct a point-location search DAG. Each triangle is represented by a node in the DAG. Triangles destroyed in a given iteration have a constant number of new triangles that overlap them. Nodes for each new triangle are given a pointer to every triangle destroyed at this step that they overlap with. Each of these operations takes $O(1)$ time in the non-noisy setting.

They prove that, over $O(\log n)$ steps, we remove a constant fraction of vertices at each step such that the last level of the decomposition is of size $O(\log n)$ with high probability in n. Because there are $O(\log n)$ steps, each takes $O(1)$ time per point, and there are $O(n)$ points, the algorithm takes $O(n \log n)$ work and $O(\log n)$ span. They also observe that the depth of the resulting search structure is $O(\log n)$ [21,28], allowing for $O(\log n)$-time sequential queries.

To convert this into a noise-tolerant algorithm, we begin by discussing how to introduce a bounding triangle to our triangulated convex polygon. We introduce our bounding points a, b, c. To triangulate the boundary of our polygon, we assign each hull point of our polygon a processor. We consider one bounding point at a time. Using $O(1)$ noisy orientation tests, each point on our polygon's hull can determine whether it "sees" bounding point a, and if so can draw an edge to it. It is easy to see that none of these edges cross. This takes $O(\log n)$ time per point using the trivial repetition strategy.

Now we consider b. Say that point p already has an edge to a. If p's neighbor in the direction of b also has an edge to c, then p should not draw an edge to b as it will cross its neighbor's edge to a. Otherwise, p draws an edge to b. Again, this takes $O(\log n)$ time per point using the trivial repetition strategy. Incorporating c involves a similar process. There are $O(n)$ points p and each can be processed independently at each of the three steps, so this takes $O(\log n)$ span and $O(n \log n)$ work.

After this, at each iteration, we have three steps: (1) sampling the independent set, (2) re-triangulating polygons, and

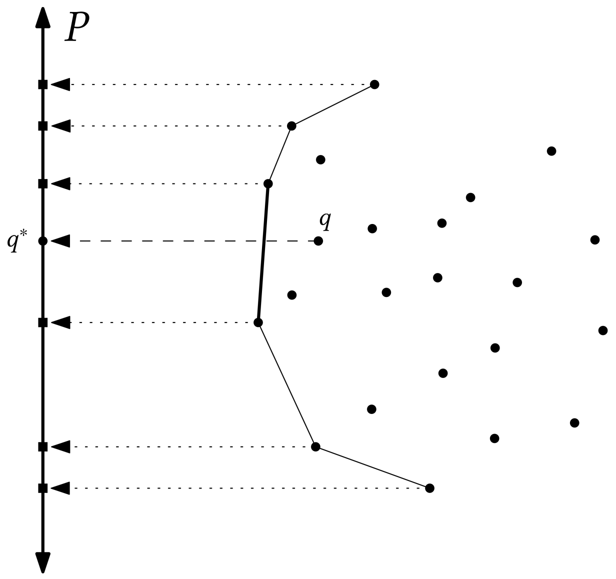


Figure 3: Here we illustrate our approach to failure sweeping a 3D hull. We define a coordinate system with its origin outside of the given hull and let P be the y-axis (the x-y plane in 3D). We define the upper hull as the hull points that can see P. We then perform an orthogonal projection of the upper hull points onto P, inducing a set of intervals. In 3D, this would induce a trianglated polygon onto P. We can project a query point q onto P and determine what region of P q^* lies in. This tells us what portion of the hull $\overrightarrow{qq^*}$ stabs. In 2D, this requires a single binary search. However, in 3D we require a planar point-location structure to determine the triangle q^* lies in.

(3) updating the search DAG. We first observe that the sampling step is without noise as we are just generating, reading, and comparing labels on points. For the re-triangulating step, we can remove noise via an initial noisy sorting computation on the x-coordinates of all $O(n)$ points in $O(\log n)$ span and $O(n \log n)$ work [13, 16]. If we label each point with its position in the sorted order, a single processor can re-triangulate a polygon of constant size in $O(1)$ time by comparing each point's labels with the others to determine in what order to sweep the polygon.

Lastly, we have to update the search DAG, which requires us to determine whether a destroyed triangle intersects with a constant number of new triangles. Because the triangles are created throughout the algorithm, it does not seem that we can preprocess the points to remove noise from these operations. Thus, if we were to construct the DAG as we compute the decomposition, the trivial repetition strategy would require that each of the $O(\log n)$ steps take $O(\log n)$ span. Instead, we apply a two-pass system.

On the first pass, construct the Kirkpatrick decomposition and create a dummy node for each triangle. We can associate each dummy node with a triangle of the decomposition and with the triangles that replaced it with a constant number of links per node. On the second pass, consider each triangle and their corresponding DAG node in parallel. For a triangle destroyed at step i, it determines which of the at most $d = O(1)$ triangles created in the corresponding gap at step i intersect it. This requires $O(1)$ primitive operations, and so can be performed in $O(\log n)$ time using the trivial repetition strategy. Each triangle then draws edges from the new triangles that overlap it to itself, creating our search structure. This takes $O(\log n)$ time per triangle and there are at most $O(n)$ triangles in the search structure [21], so this takes $O(\log n)$ span and $O(n \log n)$ work.

E.2.2 Querying the Planar Point-Location Data Structure Efficiently

As we described earlier, Reif and Sen's algorithm stops when the decomposition is of size $O(\log n)$ [28]. This is sufficient in the non-noisy case as a query point can simply brute-force search through all triangles at this first level in $O(\log n)$ time to determine which node of the DAG it should start at. Here, however, the trivial repetition strategy would add an extra log factor to this brute-force search. The following lemma will help us construct another point-location structure to help us find the initial triangle in $O(\log n)$ time w.h.p. in n. One can think of this approach as a brute-force parallelization of the planar point location methods of Cole [9] and Sarnak and Tarjan [30].

Lemma 24 *Consider a planar subdivision of size $r \le n$. Using $O(\log n)$ span, $O(r^3 \log n)$ work, and $O(r^2)$ space, we can construct a point-location data structure with high probability in n that allows for $O(\log n)$-time queries w.h.p. under noisy primitives. Construction requires $O(r^3)$ noisy operations and querying requires $O(1)$ noisy operations.*

Proof. We begin by drawing a vertical line at each point, subdividing the plane into $r + 1$ slabs. Each slab has at most $O(r)$ regions in it, so there are at most $O(r^2)$ regions. There were $O(r)$ initial triangles, so there are $O(r^3)$ triangle-region pairs. For each pair, we can determine whether the triangle and region intersect. This requires $O(1)$ noisy comparisons per pair. A parallel prefix operation can be used to combine all $O(r)$ outcomes for each region to determine the unique triangle that corresponds to it. In total, this takes $O(r^3 \log n)$ work and $O(\log n)$ span for each operation to succeed w.h.p. in n.

We can sort the slabs by their x-coordinate in $O(r \log n)$ work and $O(\log n)$ span w.h.p. in n. We can then sort each slab's regions vertically in $O(r^2 \log n)$ total work and $O(\log n)$ span w.h.p. in n.

Given a query point q, we can use two noisy binary searches to locate the region in which q lies. The first determines the appropriate slab and the second determines the appropriate region within the slab. □

In our case, $r = O(\log n)$, so total work is $o(n \log n)$ and extra space usage is $o(n)$. Lastly, it remains to show how a processor can navigate the search DAG in $O(\log n)$ time w.h.p. in n. To do this, we will construct a transition oracle for the Kirkpatrick decompositon DAG such that we can apply path-guided pushdown random walks. First, we observe that a given query point must have a unique valid path in the Kirkpatrick decomposition. Level i of the DAG corresponds to triangulation G_i of the decomposition. By definition of a triangulation, no triangles overlap, meaning that there is a unique triangle at level i that corresponds to q. Therefore, there is a unique node at level i of the DAG that is valid for q. This is true at all levels of the DAG, so each q has a single valid path through the DAG.

It also follows that we can determine whether we are on the correct path using a single noisy primitive. We simply check whether q is located inside the triangle represented by our current node in the DAG. Because the Kirkpatrick decomposition search DAG satisfies these requirements, [12] show that a single query point can navigate the DAG in $O(\log n)$ time w.h.p. in n by instantiating path-guided pushdown random walks (see Appendix B). We conclude the following:

Lemma 25 *Given a triangulation of n points in the plane, we can construct a point-location data structure to determine which triangle a query point lies in under noisy primitives. It takes $O(\log n)$ time to construct the data structure and $O(\log n)$ time to query it w.h.p. in n. Construction uses $O(n)$ noisy operations. Querying uses $O(1)$ noisy operations.*

Corollary 26 *We can perform failure sweeping on a halfspace intersection of m points in $O(m \log M)$ work and $O(\log M)$ span w.h.p. in M. This uses $O(m)$ noisy operations.*

Proof. Follows from Lemma 25 and the above discussion. Using the point-location data structure instantiated with failure probabilities in M, we can project each point q to the xy-plane, and determine which triangle the ray $\overrightarrow{qq^*}$ stabs in $O(\log M)$ time per point. From there we can perform one noisy orientation test to determine if q lies inside or outside the hull, which takes $O(\log M)$ time using the trivial repetition strategy. There are $O(m)$ points, so this takes $O(m \log M)$ work and $O(\log M)$ span. □

Theorem 27 (Same as Theorem 6) *Any parent subproblem of size M can failure sweep all of its child subproblems in $O(M \log M)$ work and $O(\log M)$ span w.h.p. in M. This uses $O(M)$ noisy operations.*

Proof. Follows from above done in parallel over all subproblems. By Lemma 23, the total size of all subproblems is $O(M)$. □

E.3 Deferred Proofs

Theorem 28 (Same as Theorem 11) *We can compute the intersection of n halfspaces in $O(\log n)$ span and $O(n \log n)$ work w.h.p. in n, even with noisy primitive operations.*

Proof. We will prove this by induction. Say we are at a parent subproblem of size M. Our inductive hypothesis assumes that we can compute each of its child subproblems of size m with failure probability at most $m^{-c'}$ in $O(\log m)$ span and $O(m \log m)$ work.

Our Section E.1 shows that the remaining steps to split and recombine subproblems take $O(\log M)$ span and $O(M \log M)$ work in total. From our discussion in the previous section, it takes $O(\log M)$ span and $O(m \log M)$ work to perform size reduction on larger subproblems. The total size of all subproblems of M is at most $O(M)$ by pruning, therefore it takes $O(\log M)$ span and $O(M \log M)$ work in total to perform size reduction and the corresponding reconstruction after returning from the recursive calls w.h.p. in M. Lastly, Theorem 6, Lemma 7, Theorem 10, and our discussion of size reduction, it takes $O(\log M)$ span and $O(M \log M)$ work to failure sweep and recompute all failed subproblems w.h.p. in M. This holds as long as $c' \geq 17$.

It remains to determine the failure parameter c for each individual primitive operation. As noted at the end of Section E.1, sampling, constructing the sampled hull and point location structure, and pruning all take $O(M)$ noisy operations in total. In our discussion of size reduction, we observed that each level of reduction takes $O(m)$ noisy operations. Therefore, splitting and recombining problems takes $O(m)$ noisy operations in total, so $O(M)$ noisy operations over all of M's subproblems. By Theorem 6 and Theorem 10, it takes $O(M)$ total noisy operations to failure sweep all problems and recompute any failed subproblems by brute-force. By union bound over all $O(M)$ invocations of noisy primitives, if a single noisy primitive fails with probability $\leq M^{-c}$, at least one fails with probability $O(M^{-(c-1)})$. Then we can take $c \geq c' + 1 \geq 18$ to achieve the desired confidence bound.

We have shown that failure sweeping in combination with our new size reduction technique allow us to efficiently upgrade the success probabilities of subproblems to that of their parents. We conclude by analyzing total span. Span is bounded by the following recurrence:

$$T(n) = T(n^{9/10}) + O(\log n),$$

which evaluates to $O(\log n)$ total span as it follows a geometric sequence. Total work is proportional to problem size times span at each level of recursion. By pruning, each level of recursion has total problem size at most $O(n_0)$, where n_0 is the initial problem size. Then work is bounded by the following recurrence:

$$W(n) = W(n^{9/10}) + O(n_0 \log n)$$

Again, this sums to $O(n_0 \log n_0)$. We conclude that it takes $O(\log n)$ span and $O(n \log n)$ work to compute all the intersection of n halfspaces under noisy primitives.

□

Lemma 29 (Same as Lemma 9) *Say that we have $O(M^{9/8})$ processes, each of which fail with high probability in $M^{1/8}$. Then at most a constant number of them fail with high probability in M.*

Proof. We expect $O(M^{9/8})/M^{c'/8}$ subproblems to fail, for some c' s.t. $c'/8 > 1$. Let X be a random variable representing the number of failed subproblems. Then, for any integer $q \geq 1$, Markov's inequality states that $\Pr[X \geq q] \leq O(M^{9/8})/M^{c'/8} \times 1/q \leq O(1/M^{(c'-9)/8})$. If we set $c' \geq 17$, this is a high-probability bound in M. □

Entropy-Bounded Computational Geometry Made Easier and Sensitive to Sortedness*

David Eppstein† Michael T. Goodrich† Abraham M. Illickan† Claire A. To†

Abstract

We study entropy-bounded computational geometry, that is, geometric algorithms whose running times depend on a given measure of the input entropy. Specifically, we introduce a measure that we call ***range-partition entropy***, which unifies and subsumes previous definitions of entropy used for sorting problems and structural entropy used in computational geometry. We provide simple algorithms for several problems, including 2D maxima, 2D and 3D convex hulls, and some visibility problems, and we show that they have running times depending on the range-partition entropy.

1 Introduction

Beyond worst-case algorithm design is directed at designing algorithms whose running time is asymptotically the best possible relative to some metric of the input instance [17, 18], and there has been considerable work done, for example, on instance-optimal sorting algorithms; see, e.g., [2–4, 8, 11, 15, 19, 20]. Focusing on the sorting problem for a moment, let $X = [x_1, x_2, \ldots, x_n]$ be an input sequence of distinct elements that come from a total order, and let $\mathcal{R} = \{R_1, R_2, \ldots, R_{\rho(X)}\}$ be a division of X into a set of maximal increasing or decreasing runs (i.e., consecutive elements in X). These previous papers define a type of ***entropy***, $\mathrm{H}(X)$, for a set of elements, X, divided into runs as follows [2–4, 8, 11, 15, 19, 20]:

$$\mathrm{H}(X) = -\sum_{i=1}^{\rho(X)} \left(\frac{|R_i|}{n}\right) \log\left(\frac{|R_i|}{n}\right).$$

Sequential comparison-based sorting has a runtime lower bound of $\Omega(n(1 + \mathrm{H}(X)))$, and many instance-optimal sorting algorithms have running time $O(n(1 + \mathrm{H}(X)))$, which can be as small as $O(n)$ depending on the input instance. For example, Auger, Jugé, Nicaud, and Pivoteau [2] show that the popular TimSort algorithm has this time, and similarly efficient stack-based mergesort algorithms have been given by Munro and Wild [15], Jugé [11], Gelling, Nebel, Smith, and Wild [8], Buss and Knop' [4], Takaoka [20], and Barbay and Navarro [3]. At CCCG'24, Schou and Wang introduce PersiSort [19], which also has this running time. Still, we are not aware of any work on geometric problems beyond sorting that concern instance optimality with respect to run-based entropy.

Instead, given an input, S, to a geometric problem, such as convex hull or maxima set, Afshani, Barbay, and Chan [1] introduce the ***structural entropy***, $\mathrm{H}(S)$ for S, and they provide algorithms that run in $O(n(1+\mathrm{H}(S)))$ time for several such problems, where n is the size of the input, including 2D maxima and 2D and 3D convex hulls. In contrast to the entropy used for the sorting problem, however, structural entropy explicitly ignores any near-sortedness in the input, such as can occur, e.g., with convex polygons or monotone polygonal chains.

Our Results. In this paper, we introduce a unification of the entropy used for sorting and the structural entropy of Afshani *et al.* [1], which we call ***range-partition entropy***. This unification applied even for problems that do not have obvious structural entropy: for instance, in the visibility and lower-envelope problems we study, range-partition entropy equals the entropy of an embedded sorting problem. In other cases, such as for 3D convex hulls, sortedness provides no obvious advantage, and case range-partition entropy reduces to the structural entropy of Afshani *et al.*

We provide simple algorithms whose running times depend on the range-partiton entropy, $\mathrm{H}(S)$, of an input set of points, S. For example, we give algorithms for computing maxima sets and convex hulls for n points in the plane in $O(n(1 + \mathrm{H}(S)))$ time. These algorithms perform no worse than any other algorithm for any input on their respective worst permutations, while also taking advantage of sortedness in the input. We also show how to adapt any instance-optimal natural mergesort to compute the lower envelope of monotone polygonal chains or the visibility polygon of a point inside a convex polygon with convex holes in time that depends on the range-partition entropy. In addition, we give a simple randomized algorithm for computing the convex hull of n points in $\mathbf{R}^3$ and show its expected running time to be $O(n(1 + \mathrm{H}(S)))$.

*This research was supported in part by NSF grant 2212129.

†Department of Computer Science, University of California, Irvine.

2 Range-Partition Entropy

Let $S = (p_1, p_2, \ldots, p_n)$ be the set of n points in $\mathbf{R}^d$ input in this given order, for constant $d \geq 1$. Define a ***range partition*** of S to be a set, $\Pi = \{(S_1, R_1), (S_2, R_2), \ldots, (S_t, R_t)\}$, such that

1. The S_i's form a partition of S, i.e., $S = \bigcup_{i=1}^{t} S_i$ and $S_i \cap S_j = \emptyset$ for $i \neq j$.
2. The R_i's are geometric ***ranges***, such as intervals in $\mathbf{R}$, axis-aligned rectangles in $\mathbf{R}^2$, triangles in $\mathbf{R}^2$, or tetrahedra in $\mathbf{R}^3$, such that the set, S_i, is contained in the range, R_i.

For example, if S is a set of points in $\mathbf{R}^2$, then Π could be a partition of S into subsets contained in a set of axis-aligned rectangles, which form the ranges.

Given a sequence, S, of points in $\mathbf{R}^d$, and a range partition, $\Pi = \{(S_1, R_1), \ldots, (S_t, R_t)\}$, for S, we say that Π is ***respectful*** if it satisfies the following constraints:

1. For each $i = 1, 2, \ldots, t$, (S_i, R_i) satisfies a given ***local property***, which depends only on S_i and R_i.
2. For each $i = 1, 2, \ldots, t$, (S_i, R_i) satisfies a given ***global compatibility*** property, which can depend on the other pairs, (S_j, R_j), for $j \neq i$.

Given a set, S, of n points in $\mathbf{R}^d$, the ***entropy***, $\mathrm{H}(\Pi)$, of a partition, $\Pi = \{(S_1, R_1), \ldots, (S_t, R_t)\}$, of S, is

$$\mathrm{H}(\Pi) = -\sum_{i=1}^{t} \left(\frac{|S_i|}{n}\right) \log \left(\frac{|S_i|}{n}\right).$$

The ***range-partition entropy***, $\mathrm{H}(S)$, of S is the minimum $\mathrm{H}(\Pi)$ over all respectful partitions, Π.

We reformulate the sorting problem in this framework by considering the input sequence, $S = (x_1, x_2, \ldots, x_n)$, to be points in $\mathbf{R}$, and we define a partition, $\Pi = \{(S_1, R_1), \ldots, (S_t, R_t)\}$, where each range, R_i, is an interval, $[a, b] \subset \mathbf{R}$. In this case, the local property of each (S_i, R_i) is that S_i is a consecutive subsequence of elements in S given in sorted order, and the global property is that the R_i ranges are disjoint, i.e., $R_i \cap R_j = \emptyset$ for $i \neq j$. Accordingly, the minimum entropy, $\mathrm{H}(\Pi)$, for all respectful partitions, Π, is determined by a partition of the input sequence into maximal non-decreasing or non-increasing runs. Thus, our framework subsumes the notion of entropy used for the sorting problem. Also, as we discuss in more detail in subsequent sections, it also subsumes the structural entropy introduced by Afshani *et al.* [1].

3 2D Maxima Set

The problem we study in this section is to find the maxima set of a given set of points in the plane, where a point is considered ***maximal*** if no other point ***dominates*** it, having both greater x- and y-coordinates. Our algorithm for this problem is a simple a variant of Kirkpatrick and Seidel's "marriage-before-conquest" method [12], which was also studied by Afshani *et al.* [1], except our algorithm takes advantage of near-sortedness.

As preprocessing, we find the point $p_{\max}$ in S with maximum x-coordinate, guaranteed to be a maximum point. We prune from S any point dominated by $p_{\max}$. Our remaining algorithm begins by checking in linear time if the input is sorted (e.g., by x- or y-coordinates). If so, it computes the maxima set in linear time by a simple plane-sweeping stack algorithm. Otherwise, in linear time, we partition the points into left and right subsets based on the median x-coordinate using a stable method, we find and add to the maxima set a point with largest y-coordinate in the right subset, and we remove all points dominated by this point. Then our algorithm recursively solves the maxima set problem for the remaining points in the left and right subsets. See Algorithm 1 in Appendix A.

Given an input sequence of n points, S, in $\mathbf{R}^2$, define the constraints for a partition, $\Pi = \{(S_1, R_1), \ldots, (S_t, R_t)\}$, where each R_i is an axis-aligned rectangle, to be ***respectful*** in the context of computing the maxima set for S (Figure 1):

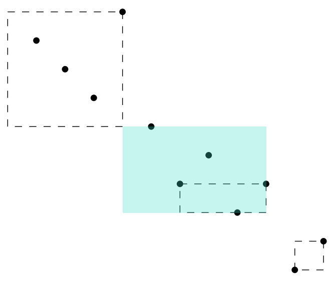
1. The ***local property*** for each (S_i, R_i) is that R_i is an axis-aligned rectangle containing S_i and either S_i forms a sorted subsequence in S or the upper right corner of R_i is dominated by some point in S.
2. The ***global compatibility*** property is that, for $i, j = 1, 2, \ldots, t$, if R_i is in not dominated by a single point in S (which means S_i is in sorted order in S), then it will not intersect another range, R_j, for $j \neq i$.

This generalizes the notion of structurally respectful partitions from Afshani *et al.* [1], which is equivalent to the local condition above for unsorted subsets. It is easy to construct inputs whose structural entropy is much higher than their range-partition entropy, such as a set of n maxima points given in sorted order.

Theorem 1. *Given a sequence, S, of n points in $\mathbf{R}^2$, the* 2DMaximaSet *algorithm runs in $O(n(1 + \mathrm{H}(S)))$ time, where $\mathrm{H}(S)$ is the range-partition entropy of S.*

Proof. Let us analyze 2DMaximaSet via an accounting argument where a constant amount of work in our algorithm costs one cyber-dollar. Let $\Pi = \{(S_1, R_1), \ldots, (S_t, R_t)\}$ be a respectful partition of S with minimum range-partition entropy, $\mathrm{H}(S)$. Since $n(1 + \mathrm{H}(S)) = n + n(\mathrm{H}(S))$, let us focus on the term

$$n(\mathrm{H}(S)) = \sum_{S_k \in \Pi} |S_k| \log(n/|S_k|).$$


(a)

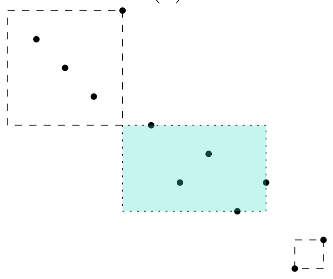

(b)

Figure 1: Respectful partitions. The points in the blue shaded rectangle are sorted among themselves. (a) A respectful partition without making use of the sorted set. The entropy is $\frac{4}{11}\log\frac{11}{4}+\frac{1}{11}\log 11+\frac{1}{11}\log 11+\frac{3}{11}\log\frac{11}{3}+\frac{2}{11}\log\frac{11}{2}\approx 2.118$. (b) A respectful partition making use of sets of both types. The entropy is $\frac{4}{11}\log\frac{11}{4}+\frac{5}{11}\log\frac{11}{5}+\frac{2}{11}\log\frac{11}{2}\approx 1.495$.

Thus, after charging each point in S one cyber-dollar, we can show that 2DMaximaSet runs in time $O(n(1+\mathrm{H}(S)))$ by showing that the processing we perform for each set, $S_k\in\Pi$, contributes at most $O(|S_k|\log(n/|S_k|)+1)$ additional cyber-dollars to the running time of our algorithm. Let T denote the recursion tree for 2DMaximaSet, where each node, v, of T corresponds to a recursive call. Each node v of T is associated with an interval, $I_v=[a_v,b_v]$, of x-coordinates for points of S between discovered maximal points for ancestors of v in T (or with $b_v=p_{\max}$ for each node, v, on the right spine of T).

So, consider a subset, $S_k\in\Pi$ which has the local property for unsorted sets. Say that S_k ***covers*** v if the x-range for R_k spans I_v but not $I_{\mathrm{parent}(v)}$.

S_k is contained in an axis-aligned box, R_k, that is strictly below the staircase. Thus, for any node, v in T, if S_k covers v, then all the points of S_k are removed from any recursive calls associated with v or its descendants in T, because they are all dominated by the upper-right corner of R_k, which in turn is dominated by, or just is the highest point to the right side of the interval and was previously discovered to be a maxima point. If S_k covers a node, v in T, then it does not participate in any recursive calls for descendants of v in T. Thus, the maximum number of points in S_k that survive to level j in T is $O(\min\{|S_k|,\lceil n/2^j\rceil\})$.

Now consider a set S_k that has the local property for a sorted subset and its range, R_k. S_k also satisfies the global compatibility condition for S_k. That is, R_k is an axis-aligned box that contains S_k such that the points of S_k are given in sorted order in S and there is no other range intersecting R_k. We will consider S_k in two parts. Let p be the rightmost point that is above R_k. $S_k^{(\ell)}$ is the subset of S_k that is dominated by p. $S_k^{(r)}$ is the remaining subset. Let $R_k^{(\ell)}$ (similarly $R_k^{(r)}$) be the rectangle that has the same y-range as R_k but the minimum x-range such that it still contains $S_k^{(\ell)}$ (similarly $S_k^{(r)}$). Say that $S_k^{(\ell)}$ (similarly $S_k^{(r)}$) ***covers*** v if the x-range for $R_k^{(\ell)}$ (similarly $R_k^{(r)}$) spans I_v but not $I_{\mathrm{parent}(v)}$.

If $S_k^{(\ell)}$ covers a node v in T, b_v is p or above p and all the points of $S_k^{(\ell)}$ have been pruned away and are removed from any recursive calls associated with v and its descendants. Thus, the maximum number of points in $S_k^{(\ell)}$ that survive to level j in T is $O(\min\{|S_k^{(\ell)}|,\lceil n/2^j\rceil\})$.

If $S_k^{(r)}$ covers a node v in T, a_v and b_v are in $S_k^{(r)}$, and no points above or below it remain in the recursive call associated with v. Since $S_k^{(r)}$ as a subset of S_k is sorted by x-coordinate, during the recursive call associated with v, our algorithm recognizes this and computes the maxima set in linear time and there are no more recursive calls. Thus, the maximum number of points in $S_k^{(r)}$ that survive to level j in T is $O(\min\{|S_k^{(r)}|,\lceil n/2^j\rceil\})$.

Therefore, the maximum number of points in S_k that survive to level j in T is $O(\min\{|S_k^{(\ell)}|,\lceil n/2^j\rceil\})+O(\min\{|S_k^{(r)}|,\lceil n/2^j\rceil\})=O(\min\{|S_k|,\lceil n/2^j\rceil\})$.

Let n_j denote the total number of points in S that survive to level j in T, and note that the total time (in cyber-dollars) charged to the 2DMaximaSet algorithm is proportional to $\sum_{j=0}^{\lceil\log n\rceil} n_j$. The proof follows, then, by the following:

$$\sum_{j=0}^{\lceil\log n\rceil} n_j \le \sum_{j=0}^{\lceil\log n\rceil}\sum_k \min\left\{|S_k|, O\left(\frac{n}{2^j}\right)\right\}$$
$$\le \sum_k \sum_{j=0}^{\lceil\log n\rceil} \min\left\{|S_k|, O\left(\frac{n}{2^j}\right)\right\}$$
$$\le \sum_k O(|S_k|\lceil\log(n/|S_k|)\rceil + |S_k| + \frac{|S_k|}{2} + \frac{|S_k|}{4} + \cdots + 1)$$
$$\le \sum_k O(|S_k|(\lceil\log(n/|S_k|)\rceil + 2)) \quad \in \; O(n(\mathrm{H}(\Pi)+1))$$

□

4 2D Convex Hull

Here we apply a similar modification to leverage sortedness in 2D convex hulls. Finding the convex hull in $\mathbf{R}^2$ involves identifying the smallest convex polygon that encloses all input points and returning the ordered subset of points that lie on the boundary. As with maxima sets, our algorithm follows Kirkpatrick and Seidel's approach as presented in [1] with a modification to check for sortedness.

The algorithm proceeds as follows. We begin by computing the upper hull. To do this, we first prune all points below the line connecting the leftmost and rightmost points of the input. Check if the points sorted. If a they are sorted, find the convex hull in linear time using Graham's scan [9]. Otherwise, the points are partitioned into two subsets based on the median x-coordinate, using a stable method. Next, we identify the two points that form the edge of the upper hull and intersect the vertical line at the median x-coordinate. This step can be done in linear time using the same method as Kirkpatrick and Seidel [13]. All points below this edge are not in the convex set and can be pruned. Recursively solve each half. Finally, the convex hulls of both subsets are concatenated to obtain the upper hull. The lower hull is computed similarly. See Algorithm 2 in Appendix A.

Given an input sequence of n points, S, in $\mathbf{R}^2$, let us define the constraints for a partition, $\Pi = \{(S_1, R_1), \ldots, (S_t, R_t)\}$, where each R_i is a triangle, to be ***respectful*** in the context of computing the convex hull for S (see Figure 2):

1. The ***local property*** for each (S_i, R_i) is that R_i is a triangle containing the points of S_i and either S_i forms a sorted subsequence in S or R_i lies under the convex hull of S.

2. The ***global compatibility*** property is that, for $i, j = 1, 2, \ldots, t$, if R_i does not lie under the convex hull of S (which means S_i is in sorted order in S), then it will not intersect another range, R_j, for $j \neq i$.

As for the sorting problem, this generalizes the notion of structurally respectful partitions of Afshani *et al.* [1], which is equivalent to the local condition above for unsorted subsets. Inputs whose structural entropy is much higher than their range-partition entropy include sets of n points on a circle given in sorted order.

Theorem 2. *Given a set, Q, of n points in $\mathbf{R}^2$ the 2DConvexHull algorithm runs in $O(n(1 + \mathrm{H}(S)))$ time, where $\mathrm{H}(S)$ is the range-partition entropy of S.*

Proof. Let us analyze 2DConvexHull via an accounting argument where a constant amount of

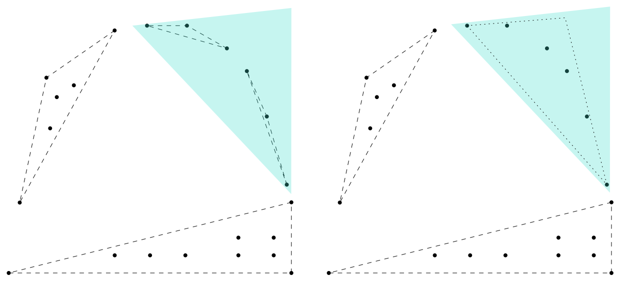

(a) (b)

Figure 2: Respectful partitions for convex hulls. (a) The points in the blue triangle are sorted among themselves but this partition doesn't take advantage of that. The entropy is $\frac{6}{22}\log\frac{22}{6} + \frac{3}{22}\log\frac{22}{3} + \frac{3}{22}\log\frac{22}{3} + \frac{10}{22}\log\frac{22}{10} \approx 1.811$. (b) This partition takes advantage of the sortedness. The entropy is $\frac{6}{22}\log\frac{22}{6} + \frac{6}{22}\log\frac{22}{6} + \frac{10}{22}\log\frac{22}{10} \approx 1.540$.

work in our algorithm costs one cyber-dollar. Let $\Pi = \{(S_1, R_1), \ldots, (S_t, R_t)\}$ be a respectful partition of S with minimum range-partition entropy, $\mathrm{H}(S)$. Since $n(1 + \mathrm{H}(S)) = n + n(\mathrm{H}(S))$, let us focus on the term

$$n(\mathrm{H}(S)) = \sum_{S_k \in \Pi} |S_k| \log(n/|S_k|).$$

Thus, after charging each point in S one cyber-dollar, we can show that 2DConvexHull runs in time $O(n(1 + \mathrm{H}(S)))$ by showing that the processing we perform for each set, $S_k \in \Pi$, contributes at most $O(|S_k| \log(n/|S_k|) + 1)$ additional cyber-dollars to the running time of our algorithm. Let T denote the recursion tree for 2DConvexHull, where each node, v, of T corresponds to a recursive call. Each node v of T is associated with an interval, $I_v = [a_v, b_v]$, of x-coordinates for points of S between discovered convex hull points for ancestors of v in T (or with $b_v = p_{\max}$, the point with the largest x-coordinate for each node, v, on the right spine of T).

So, consider a subset, $S_k \in \Pi$ which has the local property for unsorted sets.

S_k is contained in a triangle, R_k, that is strictly below the convex hull. We will consider S_k in two parts. Let p be the vertex of R_k that is opposite the lowest edge. $S_k^{(\ell)}$ is the subset of S_k that is to the left of p. $S_k^{(r)}$ is the remaining subset. Let $R_k^{(\ell)}$ (similarly $R_k^{(r)}$) be the triangle on the left (similarly right) obtained by splitting R_k along $x = p_x$. $S_k^{(\ell)}$ is contained in $R_k^{(\ell)}$ and$S_k^{(r)}$ is contained in $R_k^{(r)}$. Say that $S_k^{(\ell)}$ (similarly $S_k^{(r)}$) ***covers*** a node v of T if the x-range for $R_k^{(\ell)}$ (similarly $R_k^{(r)}$) spans I_v but not $I_{\mathrm{parent}(v)}$.

If a node v is covered by $S_k^{(\ell)}$ (similarly $S_k^{(r)}$), the edge between the convex hull points with the x-coordinate a_v and b_v lie above $R_k^{(\ell)}$ (similarly $R_k^{(r)}$). During the recursive call corresponding to v, all points below this edge are pruned and all points from $S_k^{(\ell)}$ (similarly $S_k^{(r)}$) are removed from descendants of v. Thus, the maximum number of points in $S_k^{(\ell)}$ (similarly $S_k^{(r)}$) that survive to level j in T is $O(\min\{|S_k^{(\ell)}|, \lceil n/2^j \rceil\})$ (similarly $O(\min\{|S_k^{(r)}|, \lceil n/2^j \rceil\})$). Therefore, the maximum number of points in S_k that survive to level j in T is $O(\min\{|S_k^{(\ell)}|, \lceil n/2^j \rceil\}) + O(\min\{|S_k^{(r)}|, \lceil n/2^j \rceil\}) = O(\min\{|S_k|, \lceil n/2^j \rceil\})$.

Now consider a set S_k that has the local property for a sorted subset and its range, R_k. S_k also satisfies the global compatibility condition for S_k. That is, R_k is triangle that contains S_k such that the points of S_k are given in sorted order in S and there is no other range intersecting R_k. We will consider S_k in five parts. Since R_k is a triangle, there can be at most two continuous sections of the convex hull that are strictly above R_k. Let $S_k^{(\ell)}$, $S_k^{(m)}$, and $S_k^{(r)}$ be the subsets of S_k that is to the left, middle, and right of these sections respectively. Let $S_k^{(m\ell)}$ and $S_k^{(mr)}$ be the subsets of S_k that are below each of these sections. Let $R_k^{(\ell)}$, $R_k^{(m\ell)}$, $R_k^{(m)}$, $R_k^{(mr)}$, and $R_k^{(r)}$ be the intersection of R_k with the corresponding x-ranges. Say that $S_k^{(\mathcal{X})}$ ***covers*** v if the x-range for $R_k^{(\mathcal{X})}$ spans I_v but not $I_{\text{parent}(v)}$.

If $S_k^{(\mathcal{X})}$ is $S_k^{(m\ell)}$ or $S_k^{(mr)}$, and covers a node v in T, a_v and b_v are points in the convex hull that are above $S_k^{(\mathcal{X})}$ and the points of $S_k^{(\mathcal{X})}$ will be pruned away and are removed from any recursive calls associated with v and its descendants. Thus, the maximum number of points in $S_k^{(\mathcal{X})}$ that survive to level j in T is $O(\min\{|S_k^{(\mathcal{X})}|, \lceil n/2^j \rceil\})$.

If $S_k^{(\mathcal{X})}$ is $S_k^{(\ell)}$, $S_k^{(m)}$, or $S_k^{(r)}$, and covers a node v in T, a_v and b_v are in $S_k^{(\mathcal{X})}$, and no points above or below it remain in the recursive call associated with v. Since $S_k^{(\mathcal{X})}$ as a subset of S_k is sorted by x-coordinate, during the recursive call associated with v, our algorithm recognizes this and computes the convex hull in linear time and there are no more recursive calls. Thus, the maximum number of points in $S_k^{(\mathcal{X})}$ that survive to level j in T is $O(\min\{|S_k^{(\mathcal{X})}|, \lceil n/2^j \rceil\})$.

Therefore, the maximum number of points in S_k that survive to level j in T is $O(\min\{|S_k^{(\ell)}|, \lceil n/2^j \rceil\}) + O(\min\{|S_k^{(m\ell)}|, \lceil n/2^j \rceil\}) + O(\min\{|S_k^{(m)}|, \lceil n/2^j \rceil\}) + O(\min\{|S_k^{(mr)}|, \lceil n/2^j \rceil\}) + O(\min\{|S_k^{(r)}|, \lceil n/2^j \rceil\}) = O(\min\{|S_k|, \lceil n/2^j \rceil\})$.

Let n_j denote the total number of points in S that survive to level j in T, and note that the total time (in cyber-dollars) charged to the 2DConvexHull algorithm is proportional to $\sum_{j=0}^{\lceil \log n \rceil} n_j$. The proof follows, then, since $\sum_{j=0}^{\lceil \log n \rceil} n_j \in O(n(\mathrm{H}(\Pi) + 1)))$. □

5 Visibility and Lower Envelope Problems

In this section, we show how to apply range-partition entropy to analyzing some *lower envelope* and *visibility polygon* problems. In these applications, there is no structural components, however, so the range-partition entropy is the same as the entropy in these applications as the entropy used for the sorting problem; hence, our algorithms also provide geometric applications of the entropy used for the sorting problem.

Lower envelope. Given a set of ρ disjoint monotone polygonal chains, $\mathcal{S} = \{S_1, S_2, \ldots, S_\rho\}$, of total size n, the first problem we study is to compute the piecewise-linear lower envelope of the chains in $\mathcal{S}$.

Let us define the constraints for a partition, $\Pi = \{(S_1, R_1), \ldots, (S_t, R_t)\}$, where each R_i is the polygonal chain formed by S_i, to be ***respectful*** in the context of computing the lower envelope for S (see Figure 3):

1. The ***local property*** for each (S_i, R_i) is that R_i is the polygonal chain of S_i and S_i forms a sorted subsequence in S.

2. The ***global compatibility*** property is that, for $i, j = 1, 2, \ldots, t$, R_i will not intersect another range, R_j, for $j \neq i$

In this context, each R_i is simply the same as S_i, and is included only for consistency with the definition of a respectful partition used across problems. Since this application does not depend on its structural components, R_i plays no additional role in the analysis.

Let $\mathcal{A}$ be a stack-based mergesort algorithm, like TimSort [16], be a sorting algorithm that leverages monotonic runs of an input sequence, X, of n elements to run in $O(n(1+\mathrm{H}(X)))$ time, where $\mathrm{H}(X)$ is the range-partition entropy of X (which is the same as the entropy previously studied for the sorting problem). We show how to adapt $\mathcal{A}$ to computing the lower envelope of the disjoint monotone chains in $\mathcal{S}$ in $O(n(1 + \mathrm{H}(S)))$ time. The algorithm, $\mathcal{A}$, works by maintaining a stack of maximal non-decreasing runs and merging consecutive pairs of them using the merge algorithm from mergesort for merging two sorted sequences according to rules based on their sizes that leads to the $O(n(1 + \mathrm{H}(X)))$ running time.

Our first adaptation of the algorithm, $\mathcal{A}$, is to change the merge algorithm to be a merge of two lower envelopes, where we merge the sequences by x-coordinates and determine, at each x-coordinate, the segment with the smallest y-value, compressing the sequence to eliminate segment endpoints not in the lower envelope. We assign each lower envelope a ***weight*** calculated as the sum of all contributing segments to the merged result, rather than the current number of segments in the lower envelope.

The adapted version of the algorithm, $\mathcal{A}$, maintains a stack of active chains. Suppose that the stack contains chains $S_1, S_2, \ldots, S_k$, with corresponding weights $w_1, w_2, \ldots, w_k$. We iterate through the input chains, pushing a new chain onto the stack at each iteration, as dictated by $\mathcal{A}$. After each insertion, we invoke the merge procedure to restore $\mathcal{A}$'s invariant conditions on the stack weights. For example, if $\mathcal{A}$ were TimSort, then we would ensure that the following invariants holds at the end of each iteration:

$$w_{i+2} \geq w_{i+1} + w_i, \tag{1}$$

$$w_{i+1} \geq w_i. \tag{2}$$

These conditions guarantee that the weights of sequences on the stack in TimSort grow at least as fast as the Fibonacci numbers, which ensures that the height of the stack remains logarithmic [2].

To merge two chains as dictated by $\mathcal{A}$, we perform a linear scan over their segments, selecting the smallest y-value at each x-coordinate. Since each input chain is x-monotone and disjoint, the merging process runs in linear time respect to the total number of contributing input segments, up to a constant factor. This is analogous to merging two sorted sequences. See Figure 3.

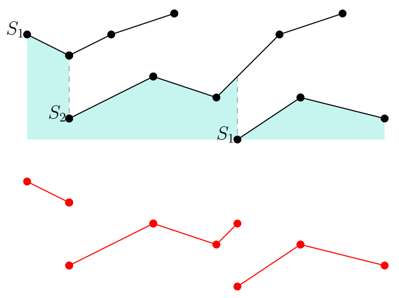


Figure 3: Merging two sets of disjoint monotone chains, where S_1 contains of two sequences and S_2 contains one.

Although the current envelope may contain fewer segments due to pruning of segments after some merges, the total weight still reflects the work that would have been done by $\mathcal{A}$ over the original input segments, preserving the same invariants used in $\mathcal{A}$'s merge strategy.

Theorem 3. *Given an set, S, of n line segments partitioned into ρ disjoint monotone chains, we compute the lower envelope of the chains in S in $O(n(1+\mathrm{H}(S)))$ time, where $\mathrm{H}(S)$ is the range-partition entropy of S.*

Proof. The proof follows from the analysis of the algorithm, $\mathcal{A}$. For example, Theorem 1 in the analysis of TimSort by Auger *et al.* [2] shows that TimSort runs in $O(n + n\mathrm{H}(X))$ time on an input, X, of length n, where $\mathrm{H}(X)$ is the range-partition entropy of X. the run-length distribution. Our adaptation of $\mathcal{A}$ into a lower envelope algorithm can be seen as analogous to $\mathcal{A}$ in the sense that each merge in our lower envelope algorithm runs in time proportional to the time needed to perform the merges in $\mathcal{A}$. Thus, the running time for our lower-envelope algorithm is $O(n(1 + \mathrm{H}(S)))$. □

Visibility polygon. In the next problem we consider, we are given a convex polygon, P, containing a set of convex polygonal obstacles, with total complexity n, and a query point, q, in P's interior, the problem is to compute the region of P that is visible from q. We assume that P and the convex obstacles in P are provided as a set of ρ disjoint convex chains, $S = \{S_1, S_2, \ldots, S_\rho\}$, where each chain represents either the outer boundary or an obstacle.

This visibility problem can be reduced to computing a lower envelope of monotone disjoint sequences, for which we provided an algorithm above. Namely, at each point along the chains, the visible segment of the polygon corresponds to the segment with the minimum radius with respect to q rather than the minimum y-value in the lower envelope setting. For each chain, we prune the vertices outside this range by locating the two endpoints of the visible window from q. Once pruned, the remaining chains can be merged using a linear scan based on the angle each point makes with respect to q. As before, we maintain the ***weight*** of each chain as the sum of all contributing segments, which increases after each merge. See Figure 4.

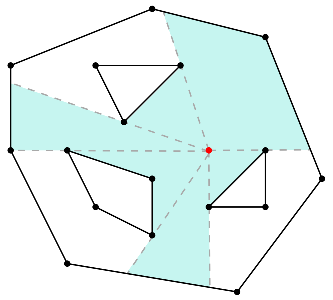


Figure 4: The visibility polygon of a point among disjoint convex chains.

Theorem 4. *Suppose we are given a polygon, P, and convex obstacles in its interior, which are given as a collection, S, of ρ disjoint convex chains consisting of a total of n vertices, where each chain represents the outer boundary or an obstacle. Then our visibility polygon algorithm runs in $O(n(1 + \mathrm{H}(S)))$ time, where $\mathrm{H}(S)$ is the range-partition entropy of S.*

Proof. The proof for the running time follows Theorem 3. Instead of selecting the minimum y-value at each x-value in the proof of Theorem 3, however, we select the minimum radius at each angle with respect to the point q. Thus, our visibility polygon algorithm runs in $O(n(1 + \mathrm{H}(S)))$ time. □

References

[1] Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-optimal geometric algorithms. *Journal of the ACM*, 64(1):A3:1–A3:38, March 2017. doi:10.1145/3046673.

[2] Nicolas Auger, Vincent Jugé, Cyril Nicaud, and Carine Pivoteau. On the worst-case complexity of TimSort, 2019. Previously announced at ESA 2018. arXiv:1805.08612v3.

[3] Jérémy Barbay and Gonzalo Navarro. On compressing permutations and adaptive sorting. *Theoretical Computer Science*, 513:109–123, 2013. doi:10.1016/j.tcs.2013.10.019.

[4] Sam Buss and Alexander Knop. Strategies for stable merge sorting. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1272–1290. SIAM, 2019. doi:10.1137/1.9781611975482.78.

[5] Timothy M Chan. Fixed-dimensional linear programming queries made easy. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 284–290, 1996. doi:10.1145/237218.237397.

[6] Timothy M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996. doi:10.1007/BF02712873.

[7] Timothy M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete & Computational Geometry*, 16(4):369–387, 1996. doi:10.1007/BF02712874.

[8] William Cawley Gelling, Markus E. Nebel, Benjamin Smith, and Sebastian Wild. Multiway powersort. In *Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 190–200. SIAM, 2023. doi:10.1137/1.9781611977561.ch16.

[9] R. L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972. doi:10.1016/0020-0190(72)90045-2.

[10] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi:10.1080/01621459.1963.10500830.

[11] Vincent Jugé. Adaptive shivers sort: An alternative sorting algorithm. *ACM Trans. Algorithms*, 20(4):31:1–31:55, August 2024. Previously announced at SODA 2020. doi:10.1145/3664195.

[12] David G. Kirkpatrick and Raimund Seidel. Output-size sensitive algorithms for finding maximal vectors. In *1st ACM Symposium on Computational Geometry (SoCG)*, page 89–96, 1985. doi:10.1145/323233.323246.

[13] David G. Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986. arXiv:https://doi.org/10.1137/0215021, doi:10.1137/0215021.

[14] Jiří Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992. doi:10.1007/bf02293051.

[15] J. Ian Munro and Sebastian Wild. Nearly-optimal mergesorts: fast, practical sorting methods that optimally adapt to existing runs. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th European Symposium on Algorithms (ESA)*, volume 112 of *LIPIcs*, pages 63:1–63:16. Schloss Dagstuhl, 2018. doi:10.4230/LIPIcs.ESA.2018.63.

[16] Tim Peters. listsort.txt, March 21, 2024. URL: https://github.com/python/cpython/blob/main/Objects/listsort.txt.

[17] Tim Roughgarden. Beyond worst-case analysis. *Communications of the ACM*, 62(3):88–96, 2019. doi:10.1145/3232535.

[18] Tim Roughgarden. *Beyond the Worst-case Analysis of Algorithms.* Cambridge University Press, 2021. doi:10.1017/9781108637435.

[19] Jens Kristian Refsgaard Schou and Bei Wang. PersiSort: a new perspective on adaptive sorting based on persistence. In *Canadian Conference on Computational Geometry (CCCG)*, pages 287–312, 2024.

[20] Tadao Takaoka. Partial solution and entropy. In Rastislav Královic and Damian Niwinski, editors, *Mathematical Foundations of Computer Science 2009, 34th International Symposium, MFCS 2009, Novy Smokovec, High Tatras, Slovakia, August 24-28, 2009. Proceedings*, volume 5734 of *Lecture Notes in Computer Science*, pages 700–711. Springer, 2009. doi:10.1007/978-3-642-03816-7_59.

[21] F. Frances Yao, David P. Dobkin, Herbert Edelsbrunner, and Michael S. Paterson. Partitioning space for range queries. *SIAM Journal on Computing*, 18(2):371–384, 1989. doi:10.1137/0218025.

A Pseudocode

In this section, we provide pseudocode descriptions of our 2D algorithms.

Algorithm 1 2DMaximaSet(S): Given a set of n points, S, in $\mathbf{R}^2$, compute the maxima set, X, of S.

```
1: if $S$ is sorted or $n \leq 1$ then
2:     Compute the maxima set for $S$ in $O(n)$ time, add the maxima points to $X$, and return.
3: Partition into $S_\ell$ and $S_r$ by median $x$-coordinate using a stable method.
4: Compute $q \in S_r$ with maximum $y$-coordinate.
5: Add $q$ to the output set $S$, and delete $q$ and prune all points in $S_\ell$ and $S_r$ that are dominated by $q$.
6: Recursively compute the maxima set for $S_\ell$, and add the maxima set of $S_\ell$ to $X$.
7: Recursively compute the maxima set for $S_r$, and add the maxima set of $S_r$ to $X$.
```

Algorithm 2 2DConvexHull(S): Given a set of n points, S, in $\mathbf{R}^2$, compute the convex hull, X, of S.

```
1: Prune all the points strictly below the line between the leftmost and rightmost points of $S$.
2: if $S$ is sorted or $n \leq 1$ then
3:     Compute the convex hull of $S$ in $O(n)$ time, and return.
4: Partition into $S_\ell$ and $S_r$ by median $x$-coordinate $m$ using a stable method.
5: Identify edge $qq'$ of the convex hull that intersects the line $x = m$ and prune the points below it.
6: Recursively compute the convex hull for $S_\ell$.
7: Recursively compute the convex hull for $S_r$.
8: Concatenate and return.
```

B 3D Convex Hull

In this section we simplify an algorithm of Afshani et al. [1] for convex hulls of points in $\mathbb{R}^3$, by finding a simple randomized replacement for a key subroutine used by Afshani et al. For the set of points $S \subset \mathbb{R}^3$, we say that a partition Π of S is structurally respectful if for each $S_k \in \Pi$, there is a tetrahedron Δ_k that contains all the points of S_k and lies under the convex hull. The structural entropy $\mathrm{H}(\Pi)$ of a partition Π is defined as $\sum_{S_k \in \Pi}(|S_k|/n)\log(n/|S_k|)$. The entropy of S is the minimum over all structurally respectful partitions of S.

Theorem 5 (Theorem 3.9 in [1]). *Given a set, S, of n points in $\mathbf{R}^3$, algorithm* `hull3d` *runs in $O(n(1+\mathrm{H}(S)))$ time, where $\mathrm{H}(S)$ is the range-partition entropy of S.*

In this algorithm, Afshani et al. assume an efficient subroutine to partition the points using Matoušek's partition theorem [14] or the recursive use of the eight-sectioning theorem [21]. The first of these methods runs in the required $O(n \log n)$ time, but the second requires $O(n^6 \log n)$ time. Our contribution for this section is to show how to find an approximate eight-section in expected linear time by using random sampling and combinatorial partitioning to divide the space into fair enough octants. Recursively doing this gives us the required $O(n \log n)$ time for the subroutine. For completeness, we provide the description of the convex hull algorithm using this subroutine in Section C.

Algorithm 3 Given a set Q of n points in $\mathbb{R}^3$, the algorithm finds a balanced partition of the points by dividing the space into octants as a subroutine to compute the 3D convex hull of Q.

Algorithm EightPartition(Q):

```
1: Define $S$ be a subset of points, chosen uniformly at random from $Q$.
2: Let $|S| \approx n^{1/10}$.
3: Define $P \leftarrow \emptyset$ to store the plane partitions
4: Define $O \leftarrow \emptyset$ to store the octant partitions
5: for all triplet of points $(p_1, p_2, p_3) \in S$ do
6:     Define the plane $\pi_i$ passing through $p_1, p_2, p_3$.
7:     for all perturbations of the plane $\pi_i$ do
8:         Classify all points as above, on, or below $\pi_i$.
9:         Add this partition to $P$.
10: for all triplet of planes $(\pi_1, \pi_2, \pi_3) \in P$ do
11:     Add the resulting partition to $O$.
12: for all octant partitions do
13:     Compute the number of points in each region.
14:     if the partition is fair enough then
15:         Return this partition.
```

Lemma 6. *For n points in $\mathbb{R}^3$, we can find a partition of this set into octants such that each octant has between $\frac{n}{16}$ and $\frac{3n}{16}$ points, in expected linear time.*

Proof. First, we sample the points with probability $p = n^{-9/10}$. By linearity of expectation, the expected sample size is $\mu = n^{1/10}$. By the multiplicative Chernoff bound with $\delta = 0.5$, the probability that the sample size will be between $\frac{n^{1/10}}{2}$ and $\frac{3n^{1/10}}{2}$ is at least $1 - 2e^{-\frac{n^{1/10}}{12}}$. We repeat the sampling process until the sample size falls within this range.

Partitioning planes are generated by selecting three points from this subset. These three points define a plane in 3D space. We slightly perturb this plane such that each of the three points lies either above, on, or below the plane. This results in 27 distinct ways in which the three points can define the plane. Since there are $O(n^{1/10})$ points in the sample, the number of ways to choose three points from this sample is $O(n^{3/10})$, and for each combination of three points, there are 27 distinct partitions. Therefore, we obtain at most $27n^{3/10}$ unique partitions from single planes. To construct a full division into octants, three planes are combined, resulting in up to $O(n^{9/10})$ possible partitions of the space into eight regions.

The algorithm examines these partitions to find one that distributes the points evenly across all octants. We are guaranteed that there is such a fair partition due to [21]. Because checking whether a single partition is balanced takes $O(n^{1/10})$ time, the runtime is $O(n)$.

Consider a subset of the original set of points that has cn points. The expected number of points that are sampled from this subset is $cn^{1/10}$. By a special case of Hoeffding's inequality [10], the probability that the fraction of points that are in this subset among all points in the sample deviates from c by more than 1/16 is less than $2e^{\frac{-n^{1/10}}{128}}$.

After getting a partition that evenly divides the sample into octants, we check if the same partition also divides all the points evenly with every octant having between $\frac{n}{16}$ and $\frac{3n}{16}$ points. By the union bound, the probability that at least one octant falls outside this range is less than $16e^{-\frac{n^{1/10}}{128}}$. If any octant falls outside this range, we repeat the sampling. The expected number of times we would have to repeat is less than 2. □

Recursive application of this method gives us the following lemma which corresponds to Lemma 3.3 in [1] in the case where the points are in $\mathbb{R}^3$.

Lemma 7. *For any set of points Q in $\mathbb{R}^3$, we can partition Q into r subsets $Q_1, \dots, Q_r$, each of size $\Theta(n/r)$ and find r polyhedral cells $\gamma_1, \dots, \gamma_r$ each with $O(\log r)$ faces such that Q_i is contained in γ_i and every plane intersects with $O(r^{\log_8 7})$ cells. We can do this in $O(n \log r)$ time in expectation.*

C 3D Convex Hull Algorithm

For completeness we present here `hull3d`, the algorithm of Afshani et al. [1] for three-dimensional convex hulls, but replacing their partitioning subroutine by our linear-time approximate eight-partition from Section B.

Algorithm 4 Given a set Q of n points in $\mathbb{R}^3$, the algorithm finds the convex hull of the points.

Algorithm `hull3d`(Q):

```
1: for $j = 0, 1, \dots, \lfloor \log(\delta \log n) \rfloor$ do
2:     Partition $Q$ into $r_j = 2^{2^j}$ subsets $Q_1, \dots, Q_{r_j}$ and
       cells $\gamma_1, \dots, \gamma_{r_j}$ by Theorem 7.
3:     for $i = 1, \dots, r_j$ do
4:         if $\gamma_i$ is strictly below the upper hull of $Q$ then
           prune all points in $Q_i$ from $Q$.
5: Compute the upper hull of the remaining points
   directly.
```

Steps 3 and 4 are done using Lemma 3.8 from [1], which refers to [5–7]. Step 5 is done using any worst case optimal convex hull algorithm which takes $O(n \log n)$ time.

PTAS for Stabbing Unit Squares and Variants

Tanmay Inamdar* Sounak Modak† Kushal Singanporia†

Abstract

In this article, we consider the problem of finding a minimum number of horizontal segments, such that each of the given rectangles is *stabbed* by at least one chosen segment (i.e., the segment intersects both vertical sides of the rectangle). This problem was first studied by [Chan et al., ISAAC 2018], who gave an $O(\log \log n)$-approximation for the problem, by observing that it can be modeled as a special case of a geometric hitting set problem in $\mathbb{R}^3$, which was known to admit an $O(\log \log n)$-approximation. To the best of our knowledge, no subsequent improvement over this result is known. In this work, we design a polynomial-time approximation scheme (PTAS) for the problem, in the special case when the given rectangles are in fact disjoint unit squares, and the horizontal segments have a bounded length. We also design a PTAS for the Maximum Coverage variant of the setting, where the goal is to stab the maximum number of squares using at most k horizontal segments. In spite of multiple restrictions imposed on the input, our results constitute as the first step towards improving the $O(\log \log n)$-approximation for stabbing arbitrary rectangles by arbitrary horizontal segments. Finally, we sketch an extension of our PTASes to the setting where the bounded-length segments can be vertical as well as horizontal. We note that this complements a known APX-hardness result when the segments can have unbounded length.

1 Introduction

In the Set Cover problem, the input consists of a *set system* $(\mathcal{U}, \mathcal{F})$, where $\mathcal{U}$ is a finite universe of size n, and $\mathcal{F}$ is a family of subsets of $\mathcal{U}$, and the aim is to find a minimum-size sub-family $\mathcal{F}' \subseteq \mathcal{F}$ whose union covers the universe. Due to its wide applicability, Set Cover is perhaps the most important combinatorial optimization problem. In the seminal work of Karp [12], it was shown to be NP-hard. Faced with this intractability, the problem has received significant attention in the field of polynomial-time approximation algorithms, where the goal is to design efficient algorithms that provably output solutions within a certain guaranteed factor from an optimal solution. In this regime, it is well-known that a greedy algorithm achieves an approximation factor of $\ln n + 1$ [16], and moreover this factor is essentially tight, assuming $\mathsf{P} \neq \mathsf{NP}$ [5].

However, this is not the end of the research for approximability of Set Cover. In fact, there are a large number of set systems whose structure allows us to overcome the lower bound of $\Omega(\log n)$, and achieve near-constant approximation factors, e.g., [1, 2, 4, 7, 10, 15, 17, 18]. One of the most prominent classes of examples that have shown remarkable success along this line comes from computational geometry. Informally speaking, Geometric Set Cover refers to an instance of Set Cover, where the set system has an underlying geometric realization. A classic example is when each element of the universe corresponds to a point, say in $\mathbb{R}^2$, and each set in the family corresponds to the subset of points contained in a certain kind of geometric object, e.g., unit disk. In the "dual" setting, the roles of points and geometric objects are reversed – the goal is to find the smallest set of points such that each geometric object contains at least one of the selected points. This setting is known as *piercing* or *hitting* of geometric objects by points. Although the exact terminology is different based on the roles of geometric objects playing the roles of elements and sets, they all fall in the broad umbrella of Geometric Set Cover.

Motivated from applications in resource allocation and scheduling, the special case of Geometric Set Cover, where the goal is to find the minimum number of lines/segments in the plane that *cross* all of the given rectangles has been extensively studied in both approximation algorithms and parameterized complexity. In this setting, the rectangles as well as lines/segments are typically restricted to be axis-parallel, and a line/segment that intersects two parallel sides of a rectangle is said to *stab* the rectangle. There are different variants of rectangle stabbing problems, based on (1) whether the rectangles are disjoint or overlapping, (2) lines/segments are vertical, horizontal, or both, and (3) whether a set of lines/segments is explicitly given in the input, or the solution can contain any line/segment in the plane. Much of the early research in this area considered lines rather than segments, and the set of lines was not explicitly given in the input. Gaur et al. [9] designed a 2-approximation for stabbing arbitrary

*Indian Institute of Technology Jodhpur
taninamdar@gmail.com
Supported by IITJ Research Initiation Grant (grant number I/RIG/TNI/ 20240072
†Institute of Mathematical Sciences
sounakmodak@gmail.com, kushal03132@gmail.com

rectangles with (non-given) horizontal and vertical lines, which remains the best approximation for the problem to this day.

More recently, Chan et al. [3] initiated the study of stabbing with segments instead of lines. They considered different variants, including a variant where the goal is to minimize the total length of the horizontal segments to stab all the rectangles, for which they designed a constant-factor approximation, which was subsequently improved to a quasi-polynomial-time approximation scheme (QPTAS) in an unpublished work of Eisenbrand et al. [6], and later to a polynomial-time approximation scheme (PTAS) by Khan et al. [13][1]. Among the different variants of rectangle stabbing [11, 13] by segments introduced in [3], the most relevant to this work is the one called CARDINALITY STABBING, where the set of horizontal segments is given in the input, and the goal is to find a minimum number of segments that stab all of the given rectangles. Chan et al. [3] showed that this variant is APX-hard, and gave an $O(\log\log n)$-approximation in polynomial time.

Our Results. The aforementioned approximation algorithm of Chan et al. [3] for CONSTRAINED STABBING works even when the rectangles can be of arbitrary sizes and may intersect each other, and the input segments can have arbitrary length. To the best of our knowledge, prior to our work, it was not known whether one can improve upon the $O(\log\log n)$ factor by restricting the input. In this work, we consider the following special case of CONSTRAINED STABBING.

STABBING DISJOINT UNIT SQUARES

Input: An instance $\mathcal{I} = (\mathcal{R}, \mathcal{S})$, where $\mathcal{R}$ is a set of n axis-parallel, disjoint unit squares and $\mathcal{S}$ is a set of m horizontal segments, each of length at most d.

Question: Find minimum-size $\mathcal{S}' \subseteq \mathcal{S}$ such that $\mathcal{S}'$ stabs all n squares of $\mathcal{R}$.

Note that, this is a considerable restriction of the original setting, since a segment of length at most d can stab at most d disjoint unit squares. Therefore, the classical greedy algorithm for SET COVER already gives a $(\ln d + 1)$-approximation in our setting. Our contribution here is to improve the approximation ratio to $1+\epsilon$. More formally, our main result is the following.

Theorem 1 *For any $\epsilon > 0$, there exists an algorithm that takes an instance of* STABBING DISJOINT UNIT SQUARES*, runs in time $m^{O(d\log d/\epsilon^2)} \cdot n^{O(1)}$ and returns a $(1+\epsilon)$-approximation.*

[1]A QPTAS is an $(1 \pm \epsilon)$-approximation algorithm that runs in $2^{(\log n)^c f(1/\epsilon)}$ for any fixed $\epsilon > 0$, where $c \geq 1$ is a constant. A PTAS is a special case where $c = 1$, which notably runs in polynomial time.

At a high level, this result is based on the classical *shifting strategy* introduced by Hochbaum and Maass [10]. At a high level, we first divide the given instance into a set of vertical sub-instances of width $O(d/\epsilon)$, via a straightforward application of shifting strategy. Then, we further horizontally sub-divide each vertical sub-instance into even smaller sub-instances whose optimal solution is bounded by $O(d\log d/\epsilon^2)$, enabling us to efficiently find it via exhaustive enumeration. The second step in our approach is inspired from a PTAS by Eisenbrand et al. [6] for minimizing the total length of the segments. We note that, although we follow a similar roadmap, we have to handle several nuances due to the significant differences between the two problems. In Section 3, we prove Theorem 1 by formally stating the algorithm and its analysis. Subsequently, we also sketch an approach that can extend to the setting where the given segments of length at most d can be vertical as well as horizontal.

Next, we consider the "MAX COVERAGE" variant of STABBING DISJOINT UNIT SQUARES, defined as follows.

MAXIMUM STABBING OF DISJOINT UNIT SQUARES

Input: An instance $\mathcal{I} = (\mathcal{R}, \mathcal{S}, k)$, where $\mathcal{R}$ is a set of n axis-parallel, disjoint unit squares, $\mathcal{S}$ is a set of m horizontal segments, each of length at most d, and k is a non-negative integer.

Question: Find a subset $\mathcal{S}' \subseteq \mathcal{S}$ of size at most k, such that $\mathcal{S}'$ stabs the maximum number of squares from $\mathcal{R}$.

For this version, we design a PTAS using a combination of the shifting strategy and dynamic programming, which is inspired from the work of Gandhi et al. [8]. More formally, we prove the following theorem in Section 4.

Theorem 2 *For any $\epsilon > 0$, there exists an algorithm that takes an instance of* MAXIMUM STABBING OF DISJOINT UNIT SQUARES*, runs in time $n^{O(d/\epsilon^2)}$ and returns a $(1+\epsilon)$-approximation.*

We will also sketch an adaptation of this algorithm to handle vertical segments.

Finally, it will be apparent from our proof that both of our algorithms can also be adapted to the setting where the input consists of disjoint rectangles, where the length and breadth of rectangles are $\Theta(1)$. Due to space constraints, some of the proofs are deferred to the appendix.

2 Preliminaries

We assume that squares are closed sets, i.e., their boundaries are included. We assume, without loss of generality, that no segment or square boundary lies at an integer y-coordinate; this can be ensured via an appropriately shifting the origin. For expressions such as $\frac{d}{\epsilon}$ that may not be integers, we assume throughout that they are rounded up using the ceiling function, i.e., $\lceil \frac{d}{\epsilon} \rceil$, wherever necessary. We note that the total number of segments m can be reduced to $O(n^3)$ by restricting attention to segments aligned with specific x and y coordinates corresponding to square boundaries. Each segment can be shifted vertically to lie in one of $O(n)$ "equivalence classes" given by the top/bottom y-coorindates of any of the n squares, or the intervals between them. Next, we restrict the left and right endpoints of each segment such that they align with the left/right boundary of a square. Note that these operations can be performed without changing the set of squares stabbed by each segment. This bounds the number of segments by $O(n^3)$.

3 A PTAS for Stabbing Disjoint Unit Squares

Throughout this section, we fix an optimal solution $\mathcal{O} \subseteq \mathcal{S}$ of size $\mathrm{OPT}(\mathcal{I})$.

3.1 Algorithm Overview

Our (1+ϵ)-approximation algorithm proceeds as follows, with each step justified by structural lemmas.

- Select a suitable vertical offset z and remove the rectangles intersected by the corresponding vertical lines $\mathcal{L}_z$. *(Lemma 3)*
- Stab the removed rectangles using a segment set $\mathcal{S}_E \subseteq \mathcal{S}$ of cost $O(\epsilon \cdot \mathrm{OPT}(\mathcal{I}))$. *(Lemma 3)*
- In each strip, perform a horizontal sweep to select a set S_h of segments that induce sub-instance boundaries. *(Lemma 5)*
- Solve each sub-instance via brute-force over small segment subsets.
- The total solution cost remains bounded by $(1 + c'\epsilon) \cdot \mathrm{OPT}(\mathcal{I})$. *(Lemma 4)*

Now, we proceed to a formal description of each of the steps. We introduce a family of vertical lines to help partition the instance into simpler parts. For any integer offset $z \in [0, \frac{d}{\epsilon})$, let $\mathcal{L}_z$ denote the set of vertical lines with x-coordinates given by $z + i \cdot \frac{d}{\epsilon}$ for $i \in \mathbb{Z}$. The offset z determines the horizontal shift of this regularly spaced grid of lines across the instance. In the analysis that follows, we use a probabilistic argument to show that some value of z yields favorable partitioning properties.

The goal of introducing $\mathcal{L}_z$ is to partition the set of rectangles into vertical regions such that each optimal stabbing segment interacts with only a limited portion of the input. This decomposition localizes the problem

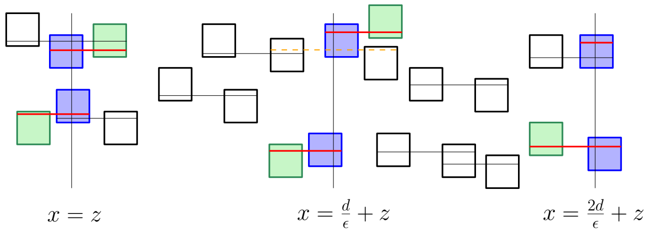


Figure 1: Squares intersecting the lines of $\mathcal{L}_z$ are shown as shaded blue squares. For each such square, we select an arbitrary segment that stabs it (shown in red). These segments, along with other rectangles they stab—even those contained strictly between the lines of $\mathcal{L}_z$ (shown as shaded green)—are not part of any of the sub-instances. However, the remaining segments (e.g., dashed orange segment) that cross the lines of $\mathcal{L}_z$ will be part of both sub-instances.

and allows us to bound the number of segments needed within each region, enabling more efficient solutions.

For a given value of z, let E_z denote the subset of squares in $\mathcal{R}$ that are intersected (stabbed) by the vertical lines of $\mathcal{L}_z$. We now claim the following.

Lemma 3 [Good-z Value] *There exists a value $0 \leq z < \frac{d}{\epsilon}$, satisfying that the number of squares of $\mathcal{O}$ that intersect any line of $\mathcal{L}_z$ is at most $\epsilon \cdot \mathrm{OPT}(\mathcal{I})$. Furthermore, for such a z, we can find in polynomial time a subset $\mathcal{S}_E$ of line segments from $\mathcal{S}$ that stabs all square in E_z, such that the total number of segments in $\mathcal{S}_E$ is at most $\epsilon \cdot \mathrm{OPT}(\mathcal{I})$.*

Since we have shown that a suitable value of z exists, we can evaluate all z values in the range $[0, d/\epsilon)$. From this point onward, we assume that such a z has been fixed, and we work with the corresponding set of vertical lines $\mathcal{L}_z$. After removing the squares in E_z and the segments used to stab them, some segments may remain that intersect vertical lines in $\mathcal{L}_z$ and stab squares from two adjacent vertical strips. Such segments may appear in more than one sub-instance and can be double-counted in the analysis. We also remove any additional squares that are stabbed by the segments used for E_z, even if those squares are not themselves in E_z. We now show that this potential overcount is bounded, and that the total cost over all sub-instances remains within a small fraction of $\mathrm{OPT}(\mathcal{I})$.

To formalize this, we define the vertical sub-instances that remain after the removal step described above. Let $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_q$ denote these sub-instances for some $q > 0$, where each $\mathcal{I}_j = (\mathcal{R}_j, \mathcal{S}_j)$ is defined as follows: $\mathcal{R}_j$ is the set of squares fully contained within the j-th vertical strip, and $\mathcal{S}_j$ is the set of segments that lie fully or partially within that strip.

Lemma 4 [Crossing Segments] *The total optimal cost across all vertical sub-instances satisfies: $\sum_{j=1}^{q} \mathrm{OPT}(\mathcal{I}_j) \leq (1 + c'\epsilon) \cdot \mathrm{OPT}(\mathcal{I})$, for some constant $c' > 0$.*

We consider a specific sub-instance $\mathcal{I}_j = (\mathcal{R}_j, \mathcal{S}_j)$, and in the next lemma, we describe how to sub-divide it further that will ultimately help us obtain a $(1+\epsilon)$-approximation for the sub-instance.

Lemma 5 (Strip Partitioning and S_h Bound) *There exists a set of horizontal segments $S_h \subseteq \mathcal{S}$ of size at most $\epsilon \cdot \mathsf{OPT}(\mathcal{I}_j)$, computable in polynomial time, that stabs a subset of $\mathcal{R}$ and partitions the remaining squares within each strip into disjoint rectangular sub-instances.*

Proof. Initially, all segments and squares are *unmarked*. We process the segments according to their y-coordinates in increasing order. At each y-coordinate corresponding to a segment s, we run the $(\ln d+1)$-factor approximation algorithm for the set cover instance given by all unmarked segments whose y-coordinate is at most that of s, and all unmarked squares stabbed by such segments. Note here that since each segment can stab at most d unit squares, greedy algorithm gives the desired guarantee. The process stops at a segment s^*, when the approximation algorithm on the corresponding sub-instance returns an output of size at least $d(\ln d+1)/\epsilon^2$. At this point, (1) we add all unmarked segments whose y-coordinate is equal to that of s^* to S_h, (3) create a sub-instance $\mathcal{I}_{j,1}$, consisting of all unmarked squares and unmarked segments inside the vertical strip below the y-coordinate of s^*, and finally (3) mark all the squares and segments that were part of the last call to the approximation algorithm corresponding to s^*.

Note that for the segment s' just before s^*, the size of the approximate solution was strictly smaller than $d(\ln d+1)/\epsilon^2$. Now, observe that the set of squares that are newly introduced in the sub-instance corresponding to s^* are those unmarked squares that are only stabbed by segments at the same y-coordinate as s^*. It is easy to see that the number of such squares is bounded by d/ϵ. Therefore, the size of optimal solution for the sub-instance is at least d/ϵ^2, and at most $d(\ln d+1)/\epsilon^2 + d/\epsilon = O(d \log d/\epsilon^2)$.

Now, we continue this procedure to create the sub-instances $\mathcal{I}_{j,2}, \ldots, \mathcal{I}_{j,t}$. Moreover, we have $\sum_{i=1}^{t} \mathrm{OPT}(\mathcal{I}_{j,i}) \leq \mathrm{OPT}(\mathcal{I}_j)$, since the sub-instances $\mathcal{I}_{j,i}$ are independent.
The overall algorithm runs in polynomial time since it iterates over polynomially many segments and invokes a polynomial-time approximation algorithm at each step. Since at each step, at most d/ϵ segments are added to S_h only after the size of optimal solution of the sub-instance below is at least d/ϵ^2, we can bound the number of added segments as follows:$|S_h| \leq \sum_{i=1}^{t-1} \epsilon \cdot \mathrm{OPT}(\mathcal{I}_{j,i}) \leq \epsilon \cdot \mathrm{OPT}(\mathcal{I}_j)$. ☐

Now we have all the ingredients required to prove Theorem 1. Due to space constraints, we give a formal proof in the appendix. Note that the running time is dominated by the time required to brute-froce in each of the smaller sub-instances, where the size of the optimal solution is bounded by $O(d \log d/\epsilon)$.

Handling horizontal as well as vertical segments. We now sketch the modifications required to extend our PTAS to work in the setting where the segments may be vertical as well as horizontal. First, we increase the range of z values to $[0, \frac{cd \ln d}{\epsilon})$ for some suitably large $c \geq 1$. Note that this also makes the horizontal distance between two consecutive lines of $\mathcal{L}_z$ to be $\Theta(\frac{d \log d}{\epsilon})$. We consider a vertical strip of width 2 centered around each line of $\mathcal{L}_z$, and say that a horizontal (resp. vertical) segment is *bad* if it crosses a line of $\mathcal{L}_z$ (resp. if it lies within the strip of width 2 around a line of $\mathcal{L}_z$). Note that each segment is bad w.r.t. at most $O(1)$ distinct values of z. Then, by using a probabilistic argument, one can show that, there exists a value of z such that at most $\frac{\epsilon}{\ln d+1}\mathrm{OPT}(\mathcal{I})$ segments (of both kinds) from a fixed optimal solution $\mathcal{O}$ are bad. We guess such a value of z, which we now fix. Now, consider all squares that intersect a line of $\mathcal{L}_z$, called E_z. We consider all such squares and all segments that can stab E_z, and use an $(\ln d + 1)$-approximation to find a solution of size at most $\epsilon \cdot \mathrm{OPT}(\mathcal{I})$. Now, we decompose the instance into vertical sub-instances.

Now we further divide each vertical sub-instance $\mathcal{I}_j$ into smaller sub-instance using a modification to the procedure described in Lemma 5. To this end, we follow a similar procedure of considering segments by increasing y-value (for horizontal segments, this will be their y-value, for vertical segments, this will be the y-value of their lower endpoint), but set the threshold of size of approximate solution to be $d^2 \log d/\epsilon$. It follows that between two consecutive segments, at most d^2/ϵ new squares can be introduced. For each such square, we add a segment that stabs it to our solution. Note that due to the threshold, the number of segments thus added is at most ϵ times the optimal solution for the sub-instance below. This decomposes the vertical sub-instance further into smaller sub-instances, where the size of optimal solution is bounded by $O(d^2 \log d/\epsilon)$, which can be solved by brute force. This strategy leads to a PTAS with running time $n^{O(d^2 \log d/\epsilon)}$.

4 A PTAS for Maximum Stabbing of Disjoint Unit Squares

In this section, we present a PTAS for Maximum Stabbing of Disjoint Unit Squares. Our high-level strategy mirrors the earlier two-phase approach: we partition the instance horizontally and vertically into smaller sub-instances, allowing a small approximation loss in the objective value. For the sake of analysis, we fix an optimal solution $\mathcal{O}_k(\mathcal{I}) \subseteq \mathcal{S}$ of size k that stabs a subset $\mathcal{R}^* \subseteq \mathcal{R}$ of size t.

For an instance $\mathcal{I}' = (\mathcal{R}', \mathcal{S}')$ and an integer ℓ, let $\mathrm{OPT}_\ell(\mathcal{I}')$ denote the maximum number of squares in $\mathcal{R}$ that can be stabbed using any ℓ segments from $\mathcal{S}$. In particular, for the original instance $\mathcal{I}$ we set $\mathrm{OPT}_k := \mathrm{OPT}_k(\mathcal{I}) = |\mathcal{R}^*|$ for notational convenience.

Step 1. Partitioning the instance into vertical strips. We begin by defining a modified family of vertical lines, similar to the construction in the previous section. For $0 \le z < \frac{2d}{\epsilon}$, let $\mathcal{L}_z$ denote the set of vertical lines of the form $x = z + i \cdot \frac{2d}{\epsilon}$ for all integers $i \in \mathbb{Z}$.

A unit square s is said to be *h-bad* with respect to z if the maximum horizontal distance from any point in s to the nearest line in $\mathcal{L}_z$ is at most d; otherwise, we classify S as *h-good* with respect to z. Note that the term *h-bad* refers to horizontal distance, although it is measured from vertical lines—a distinction that should not be confused. We now establish the following lemma, which plays a role analogous to Lemma 3.

Lemma 6 *There exists a value $0 \le z < \frac{2d}{\epsilon}$ such that the number of squares in $\mathcal{R}^*$ that are h-bad with respect to z is at most $\epsilon \cdot \mathsf{OPT}_k$.*

By enumerating all possible values of z, we fix one that satisfies the guarantee of Lemma 6. This value of z remains fixed for the remainder of the discussion. Based on the vertical lines in $\mathcal{L}_z$, we define vertical sub-instances $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_p$, where each $\mathcal{I}_j = (\mathcal{R}_j, \mathcal{S}_j)$ consists of:

- $\mathcal{R}_j$: the set of squares that are *h-good* with respect to $\mathcal{L}_z$ and lie entirely within the region between the j-th and $(j+1)$-th vertical lines in $\mathcal{L}_z$, and
- $\mathcal{S}_j$: the set of segments that can stab at least one square in $\mathcal{R}_j$.

Step 2. Solving a sub-instance within a vertical strip. Fix a vertical sub-instance $\mathcal{I}_j = (\mathcal{R}_j, \mathcal{S}_j)$. We further decompose this sub-instance using a second application of the shifting technique—this time in the vertical direction, and in a simpler form.

For $0 \le z' < \frac{1}{\epsilon}$, let $\mathcal{L}'_{z'}$ denote the set of horizontal lines defined by $y = z' + i \cdot \frac{1}{\epsilon}$, for all integers $i \in \mathbb{Z}$. A square $S \in \mathcal{R}_j$ is said to be *v-bad* with respect to z' if it intersects any line in $\mathcal{L}'_{z'}$, and *v-good* otherwise. In general, we say that a square is *good* if it is both h-good as well as v-good, and *bad* otherwise. We now state the following lemma, which is analogous to Lemma 6. The proof follows directly from the same probabilistic argument and hence is omitted.

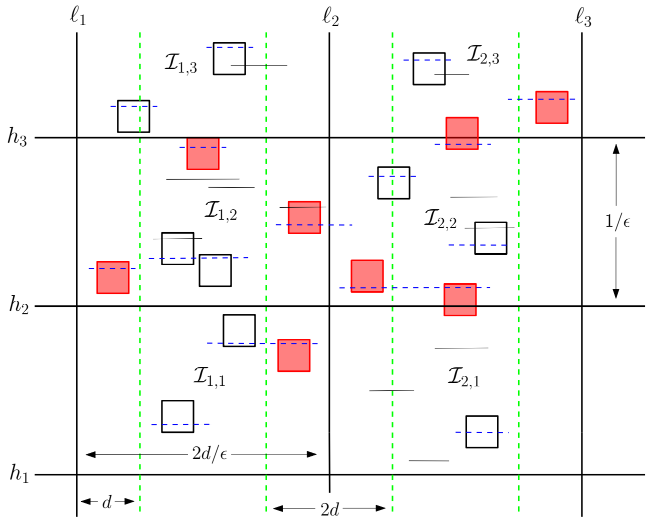


Figure 2: Squares colored red are bad squares, dotted segments shows optimal solution. Notice that our algorithm will not consider bad squares (Optimum included).

Lemma 7 *There exists some $0 \le z' < \frac{1}{\epsilon}$ such that at most $\epsilon \cdot \mathsf{OPT}_k$ squares in $\mathcal{R}^*$ are v-bad w.r.t. z'.*

We fix such a value of z' and decompose each vertical sub-instance $\mathcal{I}_i$ into smaller sub-instances $\mathcal{I}_{i,1}, \ldots, \mathcal{I}_{i,q}$. Each $\mathcal{I}_{i,j} = (\mathcal{R}_{i,j}, \mathcal{S}_{i,j})$ is defined as follows.

- $\mathcal{R}_{i,j}$ consists of only good squares from $\mathcal{R}_i$ that are fully contained within the region bounded by vertical lines i, $i+1$ and horizontal lines j, $j+1$,
- $\mathcal{S}_{i,j}$ is the set of segments from $\mathcal{S}_i$ that stab at least one square in $\mathcal{R}_{i,j}$.

Note that each sub-instance $\mathcal{I}_{i,j}$ is fully contained within a rectangle of dimensions $\frac{1}{\epsilon} \times \frac{2d}{\epsilon}$.

Lemma 8 *For each $i \ge 1$ and $j \ge 1$, let $\mathcal{I}_{i,j} = (\mathcal{R}_{i,j}, \mathcal{S}_{i,j})$ denote the j-th rectangular sub-instance of the i-th vertical sub-instance. Define $k_{i,j} := |\mathcal{S}_{i,j} \cap \mathcal{O}_k(\mathcal{I})|$ and $t_{i,j} := |\mathcal{R}_{i,j} \cap \mathcal{R}^*|$. Then:*

$$\sum_{i,j} \mathrm{OPT}_{k_{i,j}}(\mathcal{I}_{i,j}) \ge (1 - 2\epsilon) \cdot \mathrm{OPT}_k(\mathcal{I}).$$

Step 3: Solving Bounded Sub-Instances via Preprocessing

Each sub-instance $\mathcal{I}_{i,j} = (\mathcal{R}_{i,j}, \mathcal{S}_{i,j})$ lies within a rectangle of area at most $\frac{2d}{\epsilon^2}$. Since the squares are unit-sized and disjoint, the number of squares in any sub-instance is bounded by $\lambda := \frac{2d}{\epsilon^2}$. We similarly restrict attention to the subset of segments $\mathcal{S}_{i,j} \subseteq \mathcal{S}$ that intersect the same region. We exhaustively enumerate all subsets of segments from $\mathcal{S}_{i,j}$ of size at most λ and, for each, compute how many squares in $\mathcal{R}_{i,j}$ it can stab. Based on this, we construct a feasibility table: $T[i,j,s,\ell] = \texttt{true}$ iff there exists a subset of $\mathcal{S}_{i,j}$ of size at most ℓ that stabs at least s squares in $\mathcal{R}_{i,j}$.

- **Initialization:** $T[i,j,0,\ell] := \texttt{true}$ for $0 \le \ell \le \lambda$
- **Monotonicity Rule:** If $T[i,j,s,\ell] = \texttt{true}$, then also set $T[i,j,s',\ell] := \texttt{true}$ for all $s' \le s$.

This preprocessing requires time $m^{O(\lambda)}$ per sub-instance and produces a compact DP table indexed by (s,ℓ) pairs.

Step 4: Combining Sub-Instance Solutions under Global Budget

Let M denote the total number of sub-instances $\mathcal{I}_{i,j}$. To compute a global solution using at most k segments overall, we apply a dynamic programming routine across sub-instances.

Define a DP table: Let $DP[m,b]$ denote the maximum number of squares that can be stabbed using at most b segments across the first m sub-instances, here the sub-instances $\mathcal{I}_{i,j}$ are ordered arbitrarily. In the following, we slightly abuse the notation and in the table T, use m as a short-hand for i,j, where mth sub-instance is $\mathcal{I}_{i,j}$.

- **Initialization:** $DP[0,b] := 0$ for all $b \in [0,k]$
- **Transition:** For each sub-instance $m \in [1,M]$ and each budget $b \in [0,k]$,

$$DP[m,b] = \max_{0\le \ell \le b} \Big(DP[m-1, b-\ell] + \max\{s \mid T[m,s,\ell] = \texttt{true}\} \Big)$$

- **Final Result:** $\max_{0\le b\le k} DP[M,b]$ gives the best total coverage using $\le k$ segments.

The above routine runs in $O(M \cdot k \cdot \lambda)$ time, assuming constant-time access to the precomputed T-table per sub-instance. Note that M, k, and λ are polynomial (in fact, linear) in m and n, which implies that the running time of the algorithm is dominated by the $n^{O(d/\epsilon^2)}$ time required to fill the $T[\cdot]$ table in each bounded-size sub-instance. This completes the proof of Theorem 2.

Handling vertical and horizontal segments. The modifications required to handle vertical segments are straightforward. In the first step, we remove all vertical segments that lie within a distance d from a line of $\mathcal{L}_z$. The rest of this step remains unaffected. In the second step, the vertical distance between lines of $\mathcal{L}'_{z'}$ is again chosen to be $\frac{2d}{\epsilon}$, and we perform a similar analysis for the vertical segments, using a horizontal strip of height d around each line of $\mathcal{L}'_{z'}$. Using similar arguments, it can be shown that at least $(1-\Omega(\epsilon))$-fraction of squares stabbed by an optimal solution remain unaffected for some choice of z, z'. Then, a similar enumeration and dynamic programming can be used to find a near-optimal solution. Note that the running time of the PTAS now becomes $n^{O(d^2/\epsilon^2)}$ since each bounded sub-instance has size $\frac{2d}{\epsilon} \times \frac{2d}{\epsilon}$.

5 APX-Hardness for Unbounded Segments

We note that a result of Kowalska and Pilipczuk [14], that shows APX-hardness for the problem of finding the minimum number of horizontal/vertical segments to cover all the given points in $\mathbb{R}^2$. It is not too difficult to see that, by scaling the instance appropriately, one can ensure that the minimum distance between two points, as well as that between two non-intersecting segments is at least a constant, say 5. Then, we replace each point p in the scaled instance by a unit square S_p centered at p; and if any segment s has p as its endpoint, then we slightly extend s so that it completely stabs S_p. It is straightforward to see the bijection between the set of feasible solutions for the original instance of Segment Set Cover, and the new instance of Stabbing Unit Squares, establishing the APX-hardness of the latter problem. Note that due to our initial scaling, the length of the segments is not necessarily bounded.

6 Conclusion

In this work, we consider a restricted variant of rectangle stabbing problem, where the rectangles are disjoint unit squares, and the horizontal segments have bounded length. We give PTASes for the Set Cover as well as Maximum Coverage variants of the problem. It is not too difficult to see that the "disjoint square" requirement can be relaxed to consider rectangles whose length and breadth are $\Theta(1)$. Even in this setting, our techniques will yield PTASes for the two variants with the same running time, up to the constants hidden in the big-Oh notation in the exponent.

We also sketch the modifications needed to adapt our PTASes for handling axis-parallel segments of both kinds, under the assumption that the segments have bounded length. In contrast, we observe that a known result in the literature implies APX-hardness for the problem *without* an upper bound on the lengths of the segments.

References

[1] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[2] T. M. Chan, E. Grant, J. Könemann, and M. Sharpe. Weighted capacitated, priority, and ge-

ometric set cover via improved quasi-uniform sampling. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1576–1585, 2012.

[3] T. M. Chan, T. C. van Dijk, K. Fleszar, J. Spoerhase, and A. Wolff. Stabbing Rectangles by Line Segments - How Decomposition Reduces the Shallow-Cell Complexity. In W.-L. Hsu, D.-T. Lee, and C.-S. Liao, editors, *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 61:1–61:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[4] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete &amp; Computational Geometry*, 37(1):43–58, 2007.

[5] I. Dinur and D. Steurer. Analytical approach to parallel repetition. In D. B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014.

[6] F. Eisenbrand, M. Gallato, O. Svensson, and M. Venzin. A QPTAS for stabbing rectangles. *CoRR*, abs/2107.06571, 2021.

[7] T. Erlebach and E. J. Van Leeuwen. Ptas for weighted set cover on unit squares. In *Proceedings of the 13th International Conference on Approximation, and 14 the International Conference on Randomization, and Combinatorial Optimization: Algorithms and Techniques*, APPROX/RANDOM'10, pages 166–177, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, 2004.

[9] D. R. Gaur, T. Ibaraki, and R. Krishnamurti. Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. *Journal of Algorithms*, 43(1):138–152, 2002.

[10] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, Jan. 1985.

[11] S. Jana and S. Pandit. Covering and packing of rectilinear subdivision. *Theor. Comput. Sci.*, 840:166–176, 2020.

[12] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[13] A. Khan, A. Subramanian, and A. Wiese. A PTAS for the horizontal rectangle stabbing problem. *Math. Program.*, 206(1):607–630, 2024.

[14] K. Kowalska and M. Pilipczuk. Parameterized and Approximation Algorithms for Coverings Points with Segments in the Plane. In O. Beyersdorff, M. M. Kanté, O. Kupferman, and D. Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024)*, volume 289 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[15] N. H. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

[16] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978.

[17] K. R. Varadarajan. Epsilon nets and union complexity. In *Proceedings of the 25th ACM Symposium on Computational Geometry, Aarhus, Denmark, June 8-10, 2009*, pages 11–16, 2009.

[18] K. R. Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 641–648, 2010.

A Missing Proofs

In this section, we provide the proofs omitted from the main paper due to space constraints. For convenience, we restate the respective lemmas.

A.1 Proofs from Section 3

Lemma 3 [Good-z Value] *There exists a value $0 \leq z < \frac{d}{\epsilon}$, satisfying that the number of squares of $\mathcal{O}$ that intersect any line of $\mathcal{L}_z$ is at most $\epsilon \cdot \mathrm{OPT}(\mathcal{I})$. Furthermore, for such a z, we can find in polynomial time a subset $\mathcal{S}_E$ of line segments from $\mathcal{S}$ that stabs all square in E_z, such that the total number of segments in $\mathcal{S}_E$ is at most $\epsilon \cdot \mathrm{OPT}(\mathcal{I})$.*

Proof. Suppose we select z uniformly at random from $[0, d/\epsilon)$. Now we define a random variable for each segment $s_i \in \mathcal{O}$:

$$X_{s_i} = \begin{cases} 1 & \text{if } s_i \text{ crosses a line of } \mathcal{L}_z, \\ 0 & \text{otherwise.} \end{cases}$$

Note that the probability that a vertical line from $\mathcal{L}_z$ intersects a fixed segment s is the ratio of the segment's length to the spacing between vertical lines. Since each segment has length at most d, this probability is at most $\frac{d}{d/\epsilon} = \epsilon$. Thus, for each $s_i \in \mathcal{O}$, we have that, $\mathbb{E}[X_{s_i}] = \Pr[X_{s_i} = 1] \leq \epsilon$.

Let $X = \sum_{s_i \in \mathcal{O}} X_{s_i}$. By linearity of expectation, we obtain:

$$\mathbb{E}[X] = \sum_{s_i \in \mathcal{O}} \mathbb{E}[X_{s_i}] \leq \epsilon \cdot \mathrm{OPT}(\mathcal{I}).$$

This shows the existence of a required $z \in [0, \frac{d}{\epsilon})$. Now, observe that square in E_z can only be stabbed by a segment that also crosses a line $\mathcal{L}_z$, and further no segment can stab two distinct squares in E_z. Since this property is also satisfied by $\mathcal{O}$, it follows that $|E_z|$ is bounded by $\epsilon \cdot \mathrm{OPT}(\mathcal{I})$. Such distinct segments can be found by arbitrarily picking one segment for each square in E_z. $\square$

Lemma 4 [Crossing Segments] *The total optimal cost across all vertical sub-instances satisfies: $\sum_{j=1}^{q} \mathrm{OPT}(\mathcal{I}_j) \leq (1 + c'\epsilon) \cdot \mathrm{OPT}(\mathcal{I})$, for some constant $c' > 0$.*

Proof. For each sub-instance $\mathcal{I}_j$, we define a corresponding feasible solution $\mathcal{O}_j \subseteq \mathcal{O}$ as follows. $\mathcal{O}_j$ consists of those segments in $\mathcal{O}$ that either lie entirely within the vertical strip defining $\mathcal{I}_j$, or intersect one of the vertical boundaries of the strip and stab at least one square in $\mathcal{R}_j$. By construction, these segments stab all the squares in $\mathcal{R}_j$, and hence $\mathcal{O}_j$ is a feasible solution for $\mathcal{I}_j$. Therefore,

$$\mathrm{OPT}(\mathcal{I}_j) \leq |\mathcal{O}_j|.$$

We now decompose the original optimal solution $\mathcal{O}$ into two disjoint subsets:

- $\mathcal{O}^{\text{in}}$: segments that lie entirely within a single vertical strip,
- $\mathcal{O}^{\text{cross}}$: segments that intersect a vertical line separating adjacent strips.

Clearly, $\mathcal{O} = \mathcal{O}^{\text{in}} \uplus \mathcal{O}^{\text{cross}}$, and each segment in $\mathcal{O}^{\text{cross}}$ may appear in at most two sub-instance solutions.

Thus, the total number of segments used across all $\mathcal{O}_j$ satisfies:

$$\sum_{j=1}^{q} |\mathcal{O}_j| \leq |\mathcal{O}^{\text{in}}| + 2|\mathcal{O}^{\text{cross}}| \leq |\mathcal{O}| + 2|\mathcal{O}^{\text{cross}}|.$$

Following a similar argument as in the z-value lemma (Lemma 3), we can bound the number of crossing segments by $|\mathcal{O}^{\text{cross}}| \leq c\epsilon \cdot \mathrm{OPT}(\mathcal{I})$, for some constant $c > 0$. While the exact lemma concerns expected values over a random choice of z, the underlying analysis applies here to control the total number of cross-strip segments in the fixed partition. Therefore,

$$\sum_{j=1}^{q} \mathrm{OPT}(\mathcal{I}_j) \leq \sum_{j=1}^{q} |\mathcal{O}_j| \leq (1 + 2c\epsilon) \cdot \mathrm{OPT}(\mathcal{I}),$$

which proves the lemma by setting $c' = 2c$. $\square$

Theorem 1 *For any $\epsilon > 0$, there exists an algorithm that takes an instance of* Stabbing Disjoint Unit Squares, *runs in time $m^{O(d \log d/\epsilon^2)} \cdot n^{O(1)}$ and returns a $(1 + \epsilon)$-approximation.*

Proof. As established in Lemma 5, each sub-instance created via partitioning contains at most $O(d \log d/\epsilon^2)$ squares. For each such sub-instance $\mathcal{I}_{j,i}$, we exhaustively enumerate all subsets of segments of size at most $O(d \log d/\epsilon^2)$ and select the smallest feasible one that stabs all the contained squares. The number of such subsets is

$$\binom{m}{O(d \log d/\epsilon^2)} = m^{O(d \log d/\epsilon^2)}.$$

Thus, solving a single sub-instance $\mathcal{I}_{j,i}$ takes time $m^{O(d \log d/\epsilon^2)}$. Each vertical strip induces $t \leq n$ such sub-instances.

Other components of the algorithm contribute only polynomial overhead:

- The set S_h, computed by horizontal sweeping, has size at most $\epsilon \cdot \mathrm{OPT}(\mathcal{I}_j)$ and is obtained in polynomial time.
- The set E_z, determined by the lines of $\mathcal{L}_z$, also has size at most $\epsilon \cdot \mathrm{OPT}(\mathcal{I})$ and is computable in polynomial time.

Lets focus on solution by each $\mathcal{I}_j$, it has subinstances which consists of optimally solved $\mathcal{I}_{i,j}$ and S_h bounded by $\epsilon \cdot \mathrm{OPT}(\mathcal{I}_j)$. Hence it has bound $(1+\epsilon) \cdot \mathrm{OPT}(\mathcal{I}_j)$. Lemma 4 implies that combining the solutions for vertical sub-instances and E_z will give a solution of total size at most $(1+\epsilon) \cdot (1+c'\epsilon) \cdot \mathrm{OPT}(\mathcal{I}) + \epsilon \cdot \mathrm{OPT}(\mathcal{I}) = (1+O(\epsilon)) \cdot \mathrm{OPT}(\mathcal{I})$. This completes the proof. ☐

A.2 Proofs from Section 4

Lemma 6 *There exists a value $0 \le z < \frac{2d}{\epsilon}$ such that the number of squares in $\mathcal{R}^*$ that are h-bad with respect to z is at most $\epsilon \cdot \mathsf{OPT}_k$.*

Proof. For a randomly chosen $z \in [0, \frac{2d}{\epsilon})$, the probability that a given square in $\mathcal{R}^*$ is h-bad is at most

$$\Pr[\text{a square is h-bad}] \le \frac{2d}{2d/\epsilon} = \epsilon.$$

Taking expectation over all $|\mathcal{R}^*| = \mathsf{OPT}_k$ squares, we have

$$\mathbb{E}[\#\text{ h-bad squares in } \mathcal{R}^*] \le \epsilon \cdot \mathsf{OPT}_k.$$

Hence, there must exist some $z \in [0, \frac{2d}{\epsilon})$ for which the number of h-bad squares is at most $\epsilon \cdot \mathsf{OPT}_k$, as desired. ☐

Lemma 8 *For each $i \ge 1$ and $j \ge 1$, let $\mathcal{I}_{i,j} = (\mathcal{R}_{i,j}, \mathcal{S}_{i,j})$ denote the j-th rectangular sub-instance of the i-th vertical sub-instance. Define $k_{i,j} := |\mathcal{S}_{i,j} \cap \mathcal{O}_k(\mathcal{I})|$ and $t_{i,j} := |\mathcal{R}_{i,j} \cap \mathcal{R}^*|$. Then:*

$$\sum_{i,j} \mathrm{OPT}_{k_{i,j}}(\mathcal{I}_{i,j}) \ge (1-2\epsilon) \cdot \mathrm{OPT}_k(\mathcal{I}).$$

Proof. Consider sub-instance $\mathcal{I}_{i,j}$. The subset $\mathcal{S}_{i,j} \cap \mathcal{O}_k(\mathcal{I})$ contains $k_{i,j}$ segments that can stab $t_{i,j}$ squares in $\mathcal{R}_{i,j} \cap \mathcal{R}^*$. This implies:

$$\mathrm{OPT}_{k_{i,j}}(\mathcal{I}_{i,j}) \ge t_{i,j}.$$

By Lemmas 10 and 11, at most $\epsilon \cdot \mathrm{OPT}_k(\mathcal{I})$ squares from $\mathcal{R}^*$ are h-bad and at most $\epsilon \cdot \mathrm{OPT}_k(\mathcal{I})$ are v-bad. Thus, at least $(1-2\epsilon) \cdot \mathrm{OPT}_k(\mathcal{I})$ squares from $\mathcal{R}^*$ are good and retained in the modified instance.

Because the sub-instances are disjoint, we have:

$$\sum_{i,j} t_{i,j} \ge (1-2\epsilon) \cdot \mathrm{OPT}_k.$$

Combining with the inequality $\mathrm{OPT}_{k_{i,j}}(\mathcal{I}_{i,j}) \ge t_{i,j}$, the lemma follows. ☐

Inside-Out Dissections of Polygons and Polyhedra

Reymond Akpanya* Adi Rivkin† Frederick Stock‡

Abstract

This work is motivated by a question posed at last year's CCCG open problems session. In particular, we investigate inside-out dissections of polygons and polyhedra. We present various examples of polygons and polyhedra that can be inside-out dissected. We prove that a regular polygon can be inside-out dissected with at most 6 pieces. Further, we establish the existence of inside-out dissections for every polyhedra in Euclidean 3-space.

1 Introduction

In 2014 Joseph O'Rourke [18] uploaded a post on Math Stack Exchange addressing the following question: Can a polygon be subdivided into finitely many pieces, meaning polygons, that can be rearranged into a congruent copy of the original polygon such that the perimeter of the new polygon consists entirely of interior cuts of the original? He called these rearrangements "inside-out dissections". When this question was originally posed, Joseph O'Rourke presented an inside-out dissection for an arbitrary triangle that uses 9 pieces. Aaron Meyerowitz responded to his post with an inside-out dissection of an arbitrary triangle that only uses 4 pieces [17] (see Figure 1).

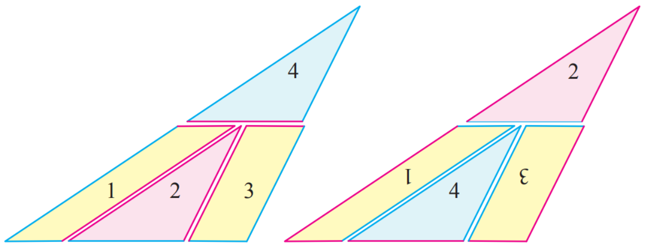


Figure 1: An inside-out-dissection of an obtuse-angled triangle using four pieces by Aaron Meyerowitz.[17]

Since every polygon with n vertices can be triangulated with $n-2$ triangles, it follows that every polygon can be inside-out dissected with at most $4(n-2)$ pieces. This raises the following question:

Question. *What is the minimum number of pieces required to inside-out dissect an arbitrary polygon with n edges?*

There was some initial discussion on the forum, but this and other questions remained unanswered. Therefore, Joseph O'Rourke posed this problem at the open problem session at CCCG 2024 [4]. The two main questions discussed at the conference were:

Question 1 *Let P be an arbitrary n-gon. Does there exist an inside-out dissection of P with k pieces $P_1, \ldots, P_k$, where $k < 4(n-2)$?*

Question 2 *Can every polyhedron be inside-out dissected?*

Question 1 was answered by A. Rivkin at CCCG 2024 by presenting a method that can inside-out dissect any n-gon with $2n+1$ pieces, see Theorem 1. Inspired by this result, we give an improved upper bound for some polygons by showing that all regular n-gons can be inside-out dissected with at most 6 pieces.

When Question 2 was presented at the open problems session, there were no known results on inside-out dissections of polyhedra. Figure 8 illustrates an inside-out dissection that was constructed by R. Akpanya. In this work, we answer Question 2 by showing that every polyhedron P with k faces can be inside-out dissected with at most $1+\sum_{i=1}^{k} 3\cdot 24\cdot(106n_i-216)$ pieces, where the i-th face of P has n_i edges.

The topic of polygonal dissections has been well-studied by the computational geometry (and the broader mathematical) community. A polygonal dissection problem gives two polygons P and Q as inputs. It then asks whether a set of polygons S can be produced via cuts of P, where the elements of S can then be transformed to create Q. Variations of dissection problems are obtained by:

1. Placing constraints on P or Q.
2. Restricting the types of cuts that are allowed.
3. Limitations on the transformations can be applied to elements of S.

The Wallace–Bolyai–Gerwien theorem [5, 14] from the early 1800s states that if P and Q are two polygons with the same area, then P can be dissected and turned into Q by using only translations and rotations.

*RWTH Aachen University, Aachen, Germany, Reymond.akpanya@rwth-aachen.de

†Dept. of Computer Science, Technion–Israel Institute of Technology, Haifa, Israel, adi.rivkin@campus.technion.ac.il

‡Miner School of Computer & Information Sciences, Lowell, MA. USA, Frederick_Stock@student.uml.edu

A more exotic class of dissections is the class of *hinged dissections*. In a hinged dissection, pieces of a polygon P can be translated and rotated, however, each piece must be "hinged" to another (where hinges are points on the boundary of each polygon) and every transformation must be made subject to these hinges. Abbot et al. [1] proved that given starting and target polygons, P and Q, where P and Q have the same area, there is always a hinged dissection from P to Q. Hinged dissections on their own are a widely researched topic see, [2, 9, 11] to just name a few papers. The first published example of a hinged dissection is Dudeney's dissection [10], which shows a hinged dissection between a square and a triangle, using only 4 pieces. Even this simple construction of an early polygonal dissection is still of interest to the community. For instance, this construction was recently (late 2024) proved to be optimal (with respect to the number of pieces) [8].

In 2023, Eppstein [12] considered dissections of polygons into the minimum number of rectangles. Here, the starting polygon P is any axis-aligned polygon and the pieces of such a dissection can only be constructed by axis-aligned cuts of P. However, the target shape Q is no longer predefined, and we instead consider the optimization problem: What is the fewest number of rectangles that can be constructed? An exact algorithm is provided in the case that the only allowed transformations are translations, and Eppstein gives a 2-approximation when both translations and rotations are allowed.

One of the oldest examples of a polygonal dissection is the Tangram puzzle. In this puzzle the initial polygon P is always a square, and the set S is a predefined set of 7 pieces. Q is chosen by the player from a list of predefined shapes. Finally, pieces of S can be transformed by rotations, translations, and flips (see [19] for a formal analysis). For some other works on dissections we refer to [15, 16].

Regarding dissections of polyhedra, there is the notion of the Dehn Invariant [7]. Given two polyhedra, they can be dissected and rearranged into each other if and only if they have equal volume and Dehn Invariant. Crucially, this only addresses the existence of a dissection but not an inside-out-dissection, leaving Question 2 open.

2 Preliminaries

In this section, we present our notions of polygons and polyhedra. We further provide a formal definition of an inside-out dissection of a polygon (polyhedron). For this purpose, we denote the convex hull of a set $M \in \mathbb{R}^n$ as $\operatorname{conv}(M)$ in this paper.

Remark 1 *1. It is unclear how an inside-out dissection for a non-simple polygon would be defined. Hence, we assume that all polygons in this work are simple. Furthermore, we note that the polygons in this paper do not necessarily have to be convex.*

2. In the literature, there are different notions of polyhedra, see [20] for instance. Here, a polyhedron is considered to be a subset of the real 3-space whose boundary solely consists of polygonal faces, straight line edges and vertices. Hence, the polyhedra in this work are allowed to be non-convex. Further, we will exploit the fact that a convex polyhedron can be constructed as the convex hull $\operatorname{conv}(M)$ of a finite subset $M \subset \mathbb{R}^3$. Moreover, δP denotes the boundary of P.

Definition 1 *We say that polygons (polyhedra) $P_1, \dots, P_k$ form a decomposition of a polygon (polyhedron) P, if*

1. $\bigcup_{i=1}^{k} P_i = P$,
2. $P_i \cap P_j = \delta P_i \cap \delta P_j$ *for all* $1 \leq i < j \leq k$.

With these notions in place, we are able to formally define inside-out dissections.

Definition 2 *Let P be a polygon (polyhedron). An* **inside-out dissection** *of P is a decomposition of P into finitely many polygons (polyhedra) $P_1, \dots, P_k$ such that*

1. *the polygons $P_1, \dots, P_k$ can be rearranged by only applying rotations and translations to form a decomposition of a polygon (polyhedron) P' that is congruent to P,*
2. *the boundary of the polygon (polyhedron) P' is composed of internal cuts of P, i.e. no line segment (polygon) contained on the boundary of P appears as a line segment (polygon) on the boundary of P'.*

In the case that such a decomposition of P exists, we denote it by $(P_1, \dots, P_k)$ and say that P can be **inside-out dissected**. *We refer to the polygons (polyhedra) $P_1, \dots, P_k$ as* **pieces** *of P. Furthermore, we define $\mathcal{I}(n)$ as the smallest natural number such that every n-gon can be inside-out dissected with $k \leq \mathcal{I}(n)$ pieces.*

As described in Section 1 every polygon can be inside-out dissected. In the following sections, we will provide various examples of inside-out dissections of polygons and polyhedra.

At last year's CCCG, A. Rivkin established a method to inside-out dissect an arbitrary polygon with n edges with $2n+1$ pieces, thereby improving the best previous upper bound of $4(n-2)$ pieces. This method has been illustrated in [4] without giving a formal proof. Thus, we give a formal proof of this method in this paper for completeness.

Theorem 1 *Every polygon with n edges can be inside-out dissected with $2n+1$ pieces, i.e. $\mathcal{I}(n) \leq 2n+1$.*

Proof. First, we show that a triangle T can be inside-out-dissected with 2 pieces per edge and one extra "scrap" piece. For every edge e of T, construct two congruent isosceles triangles t_1, t_2 such that their congruent edges are of length $|e|/2$, one of which lies on the perimeter of T. Thus, t_1 and t_2 can be rotated by π and then translated to replace each other. This rotation places the other (internal) edge of length $|e|/2$ of t_1 and t_2 on the perimeter of T, and the edge that was originally on the exterior of T is now internal to T. This can be done for each edge of T, inside-out dissecting T in 7 pieces (illustrated in Figure 2).

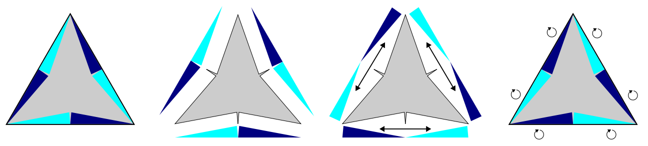


Figure 2: A triangle can be inside-out dissected with 7 pieces.

This method can be expended to any polygon P with n edges, yielding an inside-out dissection of P using $2n+1$ pieces. For every edge of P we construct two isosceles triangles just as we did in the triangle example, and perform the same procedure, rotating and exchanging each triangle (demonstrated in Figure 3).

□

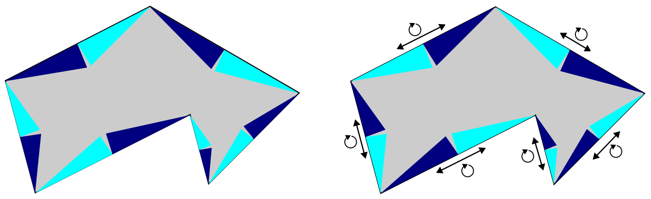


Figure 3: A polygon can be inside-out dissected with $2n+1$ pieces.

3 Inside-out Dissections of Polygons

Now, we consider inside-out dissections of regular polygons. We obtain the result that any regular polygon can be inside-out dissected using at most 6 pieces.

Proposition 1 *Any regular polygon P can be inside-out dissected with at most 6 pieces.*

Proof. For $n = 3, 4,$ and 5 we have ad hoc constructions (Figure 1 and Figure 4). For $n \geq 6$, the rough idea is to cut large rotationally symmetric pieces from P

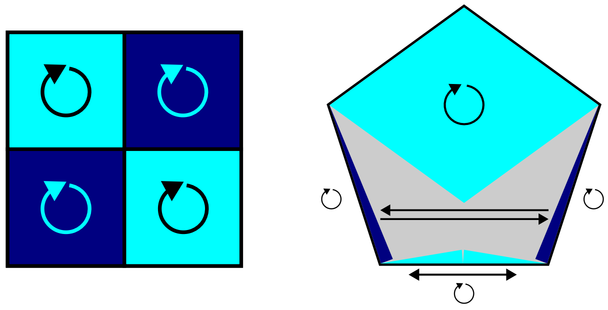


Figure 4: Regular squares and pentagons can be inside-out dissected with at most 6 pieces.

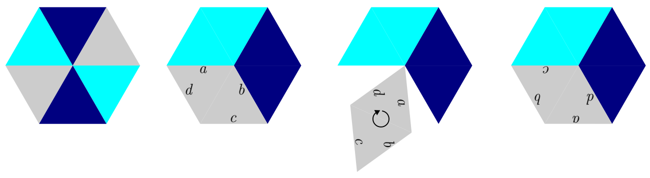


Figure 5: An inside-out dissection of a regular hexagon.

and rotate them so that perimeter edges become internal. This approach is inspired by the method presented in the proof of Theorem 1, where isosceles triangles are cut out and swapped. We start by demonstrating the technique for regular hexagons. For the regular polygon P, each internal angle is $\frac{\pi(n-2)}{n}$. Thus, for $n = 6$, every internal angle is $\frac{\pi(6-2)}{6} = \frac{2\pi}{3}$. We can then cut a hexagon into three congruent rhombi by bisecting every other angle. Each rhombus can now be rotated π radians, inside-out dissecting a regular hexagon with three pieces, demonstrated in Figure 5.

We now move to the case that P is any regular polygon. In the same manner as we did for the hexagon, one can use the internal angle to compute the maximum rotationally symmetric $2k$-gon that can be cut out from P. These $2k$-gons consist of two identical convex polygonal chains, p_1 and p_2. One of the chains, say p_1, is simply a segment of the perimeter of P with k edges and its two endpoints vertices of P. The other chain, p_2, is the reflection of p_1. The polygon formed by $p_1 \cup p_2$ is a polygon with $2k$ edges, and $2k-2$ internal angles equal to exactly the internal angle of P. It then has equal angles θ that are not an internal angle of P.

We can therefore find a closed-form equation for θ as follows:

$$\begin{aligned} & \pi(2k-2) = 2(k-1)\left(\frac{\pi(n-2)}{n}\right) + 2\theta \\ \Leftrightarrow \quad & \pi(k-1) - (k-1)\left(\frac{\pi(n-2)}{n}\right) = \theta \\ \Leftrightarrow \quad & \pi(k-1)\left(1 - \frac{n-2}{n}\right) = \theta. \end{aligned}$$

With an expression for θ, we can compute the maximum k we can cut from the perimeter of P. This can be achieved by enforcing that θ should be half as large as an interior angle of P :

$$\begin{aligned} & \theta = \pi(k-1)\left(1-\frac{n-2}{n}\right) \leq \pi\left(\frac{n-2}{n}\right)\left(\frac{1}{2}\right) \\ \Leftrightarrow \quad & (k-1)\left(1-\frac{n-2}{n}\right) \leq \frac{n-2}{2n} \\ \Leftrightarrow \quad & (k-1)\left(\frac{2}{n}\right) \leq \frac{n-2}{2n} \\ \Leftrightarrow \quad & (k-1) \leq \frac{n-2}{4} \\ \Leftrightarrow \quad & k = \left\lfloor \frac{n-2}{4}+1 \right\rfloor . \end{aligned}$$

So, we can inside-out dissect P with $\lceil \frac{n}{k} \rceil + 1 \leq 6$ pieces. □

For illustration, we present inside-out dissections of a regular 7-gon and a regular 10-gon using rotationally symmetric pieces as described in the proof of Proposition 1.

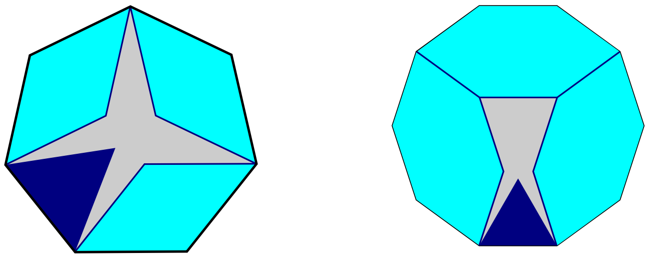


Figure 6: Two regular polygons with their pieces (Left) 7-gon, (right) 10-gon.

In [13] we provide a tool that allows us to visualise our constructed inside-out dissections for arbitrary regular polygons.

4 Inside-out Dissections of Polyhedra

Here, we study inside-out dissections of polyhedra in Euclidean 3-space.

As a first example of a polyhedron that can be inside-out dissected, we consider the regular tetrahedron, i.e. the Platonic solid consisting of four equilateral triangles as faces. In order to illustrate a corresponding inside-out dissection, we give the following remark.

Remark 2 *Let $\mathcal{T}$ be a regular tetrahedron with all edge lengths being 1. Then T can be decomposed into four tetrahedra and one regular octahedron with all edge lengths being $\frac{1}{2}$. Additionally, if $\mathcal{O}$ is a regular octahedron which all edge lengths being 1, then $\mathcal{O}$ can be decomposed into eight regular tetrahedra and six regular octahedra with all edge lengths being $\frac{1}{2}$. These decompositions are illustrated in Figure 7.*

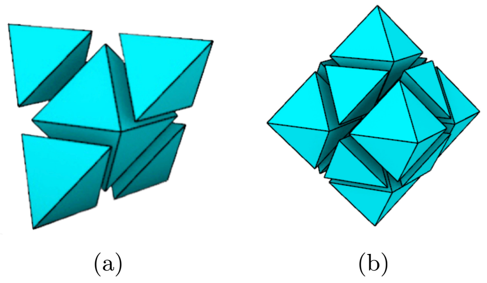


Figure 7: Decomposition of (a) a regular tetrahedron into four regular tetrahedra and one regular octahedron and (b) a regular octahedron into six regular tetrahedra and eight regular octahedra. [3]

The above remark can be verified by examining the tetroctahedrille, i.e. the space-filling of the Euclidean 3-space consisting of regular tetrahedra and octahedra [6] and exploiting the fact that the vertices of the different tetrahedra and octahedra in this space-filling form a 3-dimensional lattice. By iterating the process of decomposing tetrahedra and octahedra, we observe that a regular tetrahedron can be decomposed into 24 regular tetrahedra and 10 regular octahedra, see Figure 8.

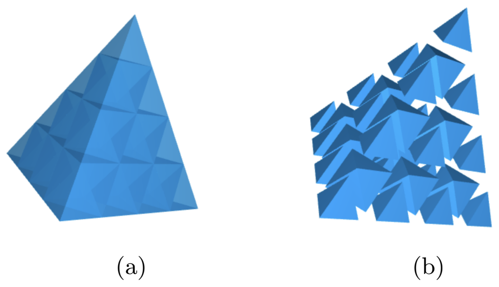


Figure 8: (a) Decomposition of a regular tetrahedron into 24 regular tetrahedra and 10 regular octahedra (b) an exploded view of the described subdivision.

Hence, by applying suitable transformations to the tetrahedra and octahedra of the above decomposition, we obtain the desired inside-out dissection of a regular tetrahedron.

Proposition 2 *Let $\mathcal{T}$ be a regular tetrahedron. Then $\mathcal{T}$ can be inside-out dissected with* 34 *pieces forming regular tetrahedra and octahedra.*

Proof. We refer to a face of a polyhedron of the decomposition in Figure 8 as boundary face if it is contained in the boundary of $\mathcal{T}$. Thus, it can be observed that the above decomposition has exactly $2 \cdot 6 + 4 + 4 = 20$

tetrahedra and $6+4=10$ octahedra that have at least one boundary face. More precisely, there are exactly (1) 4 tetrahedra having exactly 3 boundary faces, (2) 12 tetrahedra having exactly 2 boundary faces, (3) 4 tetrahedra having exactly 1 boundary face, (4) 4 octahedra having exactly 3 boundary faces and (5) 6 octahedra having exactly 2 boundary faces. In order to construct the desired inside-out dissection of the tetrahedron $\mathcal{T}$ the tetrahedra described in (1) and (3) have to be swapped by applying a suitable transformation and the polyhedra in (2), (4) and (5) have to be rotated so that the boundary faces of these polyhedra do not form boundary faces of the rearranged tetrahedron. □

By observing the decomposition of the regular octahedron shown in Figure 7b and pursuing a similar strategy as presented in Proposition 2, we can show that a regular octahedron can be inside-out dissected.

Proposition 3 *Let $\mathcal{O}$ be a regular octahedron. Then $\mathcal{O}$ can be inside-out dissected with* 14 *pieces forming regular tetrahedra and octahedra.*

Since Proposition 1 establishes that any regular polygon can be inside-out dissected with at most 6 pieces, we are able to derive an infinite family of polyhedra, where every polyhedron of this family can be inside-out dissected with at most 12 pieces.

Corollary 1 *Let P be a regular prism. Then P can be inside-out dissected with at most* 12 *pieces.*

Proof. The polyhedron P has two types of faces, n square faces and two faces that are congruent regular n-gons, call this n-gon p. Without loss of generality, assume that P is oriented such that the two regular n-gon faces are the top and bottom faces of P, and the square faces are therefore oriented vertically between these two faces p_t and p_b (the top and bottom face respectively). By Proposition 1 the polygons p_t and p_b can both be inside-out dissected with at most 6 pieces such that the pieces of p_t are congruent to the pieces of p_b. Let the height of P (the distance between p_t and p_b) be h. Dissect both p_t and p_b and then extrude each piece to obtain prisms of height $h/2$. Now, every extruded prism having a face that forms a piece of p_t can be swapped with an extruded prism having a face that forms a piece of p_b congruent to the given piece of p_t. This can be achieved such that the rearranged prisms form an inside-out dissection of the polyhedron P. □

Next, we introduce Theorem 2 which establishes the existence of inside-out dissections of polyhedra. The main idea to prove this result is to subdivide the surface of a given polyhedron P into isosceles triangles and exploit this triangulation to derive the desired inside-out dissection of P. The key to constructing such a triangulation consisting of isosceles triangles is that every polygon admits a triangulation consisting of acute triangles, see [21]. Before proceeding with the proof of the desired result, we describe how acute triangles can be further subdivided into three isosceles triangles.

Remark 3 *Let t be an acute triangle with vertices $v_1, v_2, v_3 \in \mathbb{R}^2$. Since t is acute, the circumcenter c, i.e. the intersection point of all perpendicular bisectors of t, is contained within the interior of t, see Figure 9 for illustration. By construction the triangles $(v_1, v_2, c), (v_1, v_3, c), (v_2, v_3, c)$ all form isosceles triangles that subdivide t.*

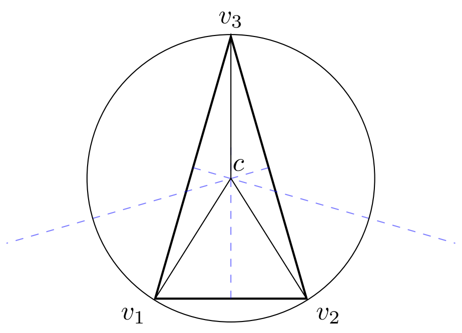


Figure 9: Exploiting the circumcenter of an acute triangulation to construct a subdivision of the given triangle into three isosceles triangles.

Inspired by Theorem 1, we aim to inside-out dissect a polyhedron by using rotationally symmetric pieces. We construct these pieces such that two faces of a given piece form congruent isosceles triangles as detailed below.

Remark 4 *Let t be an isosceles triangle with edge lengths (a, b, b), where a, b are positive real numbers. Because of t being isosceles, we know that the inequality $b > \frac{a}{2}$ holds. We construct a tetrahedron that is rotationally symmetric and has two faces that are congruent to t as follows: For $\alpha \in (0, \frac{\pi}{2})$ and $r := \sqrt{b^2 - \frac{a^2}{4}}$ we define the tetrahedron τ by its corresponding four vertices $v_1, \ldots, v_4 \in \mathbb{R}^3$ given as follows:*

$$v_1 := (\tfrac{a}{2}, 0, 0), v_2 := (-\tfrac{a}{2}, 0, 0),$$
$$v_3 := (0, r\cos(\alpha), r\sin(\alpha))$$
$$v_4 := (0, -r\cos(\alpha), r\sin(\alpha)).$$

Figure 10 illustrates the arising tetrahedron. By construction we see that the sets

$$t_1 := \text{conv}(\{v_1, v_2, v_3\}), t_2 := \text{conv}(\{v_1, v_2, v_4\})$$

form triangles with edge lengths (a, b, b). Moreover, we see that the symmetry group of τ admits a 180 *degree rotation around the z-axis.*

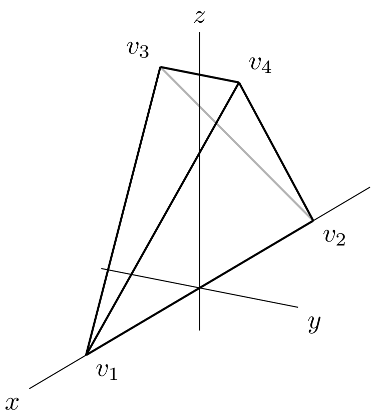


Figure 10: A rotationally symmetric tetrahedron with four isosceles faces

With the help of the above remarks, we are able to prove the main result of this section.

Theorem 2 *Every polyhedron can be inside-out dissected.*

Proof. Let P be a polyhedron and f an arbitrary face of P. Since f is a face of P, it forms a polygon in $\mathbb{R}^2$. By [21] we know that f can be triangulated such that each arising triangle is acute, see Figure 11a for illustration. For simplicity, we denote these acute triangles by $t_1, \ldots, t_n$. Note that the circumcenter of t_i is contained within the interior of t_i for all $1 \leq i \leq m$. Thus, by connecting the circumcenter of each triangle t_i to its vertices as described in Remark 3, we obtain a more refined subdivision of f consisting of isosceles triangles, see Figure 11b for illustration.

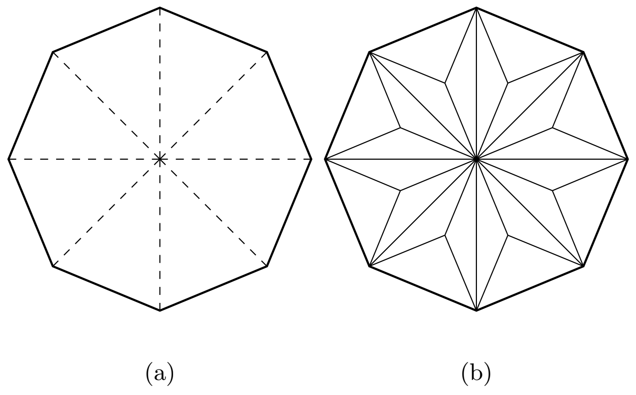


Figure 11: An acute triangulation triangulation of a regular octagon (a); a triangulation of a regular octagon consisting of isosceles triangles (b).

Since f is arbitrary, we know that the boundary of P can be triangulated with isosceles triangles $t'_1, \ldots, t'_m$. Note that if (a_i, b_i, b_i) are the edge lengths of the triangle t'_i, then $b_i > \frac{a_i}{2}$ holds and the edges of length b_i are not contained in any edge of P (by construction). Hence, we can exploit these triangles to construct polyhedra $p_1, \ldots, p_m$ such that

- the polyhedron p_i is a rotationally symmetric tetrahedron as described in Remark 4 for all $1 \leq i \leq m$,
- the triangle t'_i forms a face of the polyhedron p_i for all $1 \leq i \leq m$,
- the polyhedra p_i and p_j intersect at most in their boundary for all $1 \leq i < j \leq m$.

As each tetrahedron p_i is rotationally symmetric, it can be rotated to replace its external face f with its internal face that is congruent to f. Hence, the pieces $P \setminus \{p_1, \ldots, p_m\}, p_1, \ldots, p_m$ form the desired inside-out dissection of P. □

Note that [21] establishes that every n-gon can be triangulated with at most $24 \cdot (106n - 216)$ triangles forming acute triangles. We can exploit this fact to calculate an upper bound on the number of pieces of the inside-out dissection of a given polyhedron derived in the proof of Theorem 2. Let P be a polyhedron and $f_1, \ldots, f_k$ exactly the faces of P, where the face f_i forms a n_i-gon for $1 \leq i \leq k$. In our proof, every face of P is first triangulated with acute triangles. We then obtain an inside-out dissection of P by subdividing every arising acute triangle into three isosceles triangles. Thus, the derived inside-out dissection of P consists of at most

$$1 + \sum_{i=1}^{k} 3 \cdot 24 \cdot (106n_i - 216)$$

pieces.

5 Conclusion & Outlook

In this paper, we have studied inside-out dissections of polyhedra and polygons. We have established the existence of inside-out dissections for polyhedra and examined inside-out dissections of regular polygons. In future work, we want to address the following questions:

(1) Theorem 1 establishes that any n-gon can be inside-out dissected with $2n + 1$ pieces. We want to investigate whether this bound is sharp. Thus, we explore the existence of an n-gon that cannot be inside-out dissected with fewer than $2n + 1$ pieces.

(2) We want to improve the upper bound on the number of pieces of inside-out dissection of polyhedra established in the proof of Theorem 2.

(3) The inside-out dissections that have been produced for arbitrary polygons in Theorem 1 and arbitrary polyhedra in Theorem 2 contain one non-convex piece. Is it possible to inside-out dissect every polyhedron and every polygon with pieces that are all convex?

References

[1] T. G. Abbott, Z. Abel, D. Charlton, E. D. Demaine, M. L. Demaine, and S. D. Kominers. Hinged dissections exist. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 110–119, 2008.

[2] J. Akiyama and G. Nakamura. Dudeney dissections of polygons and polyhedrons—a survey. In *Discrete and computational geometry (Tokyo, 2000)*, volume 2098 of *Lecture Notes in Comput. Sci.*, pages 1–30. Springer, Berlin, 2001.

[3] R. Akpanya, T. Goertzen, and A. C. Niemeyer. Topologically Interlocking Blocks inside the Tetroctahedrille, 2024.

[4] R. Akpanya, B. Rivier, and F. Stock. Open Problems from CCCG 2024. In *Proceedings of the Candian Conference on Computational Geometry, July 2024*, page 167, 2024.

[5] F. Bolyai. *Tentamen juventutem studiosam in elementa matheseos purae, elementaris ac sublimioris, methodo intuitiva, evidentiaque huic propria, introducendi.* Typis Collegii Refomatorum per Josephum et Simeonem Kali, Maros Vásárhely, 1832–1833.

[6] J. H. Conway, H. Burgiel, and C. Goodman-Strauss. *The Symmetries of Things.* A K Peters, Ltd., Wellesley, MA, New York, 2008.

[7] M. Dehn. Ueber den rauminhalt, Sept. 1901.

[8] E. Demaine, T. Kamata, and R. Uehara. Dudeney's dissection is optimal, 12 2024.

[9] E. D. Demaine, M. L. Demaine, D. Eppstein, G. N. Frederickson, and E. Friedman. Hinged dissection of polyominoes and polyforms. *Computational Geometry Theory and Applications*, 31(3):237–262, 2005.

[10] H. E. Dudeney. Puzzles and prizes. *Weekly Dispatch*, 1902. April 6, April 20, May 4.

[11] D. Eppstein. Hinged kite mirror dissection, 2001.

[12] D. Eppstein. Orthogonal dissection into few rectangles. *Discrete & Computational Geometry*, Dec 12:1–20, 2023.

[13] S. Frederick. Inside out dissections. https://fred-stock.github.io/InsideOutDissections/, 2025.

[14] P. Gerwien. Zerschneidung jeder beliebigen anzahl von gleichen geradlinigen figuren in dieselben stücke. 1833.

[15] M. Goldberg and B. M. Stewart. A dissection problem for sets of polygons. *The American Mathematical Monthly*, 71(10):1077–1095, 1964.

[16] E. Kranakis, D. Krizanc, and J. Urrutia. Efficient regular polygon dissections. In *Discrete and Computational Geometry*, pages 172–187, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[17] A. Meyerowitz. Inside-out polygonal dissections. MathOverflow, 2014. (version: 2014-11-06).

[18] J. O'Rourke. Inside-out polygonal dissections. MathOverflow, 2014. (version: 2014-11-06).

[19] F. T. Wang and C.-C. Hsiung. A theorem on the tangram. *The American Mathematical Monthly*, 49(9):596–599, 1942.

[20] W. Whiteley. Realizability of polyhedra. *Structural Topology*, (1):46–58, 73, 1979.

[21] L. Yuan. Acute triangulations of polygons. *Discrete & Computational Geometry*, 34(4):697–706, 2005.

Quasigeodesics on the Cube

MIT CompGeom Group* Hugo A. Akitaya† Erik D. Demaine‡ Adam Hesterberg§ Thomas C. Hull¶ Anna Lubiw‖ Jayson Lynch** Klara Mundilova†† Chie Nara‡‡ Joseph O'Rourke§§ Frederick Stock¶¶ Josef Tkadlec‖‖ Ryuhei Uehara***

Abstract

A quasigeodesic is a curve on the surface of a convex polyhedron that has $\leq \pi$ surface angle to each side at every point. In contrast, a geodesic has exactly π angle to each side and so can never pass through a vertex, whereas quasigeodesics can. Although it is known that every convex polyhedron has at least three simple closed quasigeodesics, little else is known. Only tetrahedra have been thoroughly studied.

In this paper we explore the quasigeodesics on a cube, which have not been previously enumerated. We prove that the cube has exactly 15 simple closed quasigeodesics (beyond the three known simple closed geodesics). For the lower bound we detail 15 simple closed quasigeodesics. Our main contribution is establishing a matching upper bound. For general convex polyhedra, there is no known upper bound.

1 Introduction

1.1 Quasigeodesics: Examples and Intuition

A ***quasigeodesic*** is a curve on the surface of a convex polyhedron that has $\leq \pi$ surface angle to each side at every point. In contrast, a ***geodesic*** has exactly π angle to each side. Because a vertex is a point with less than 2π surface, a geodesic can never pass through a vertex. Quasigeodesics can pass through vertices; a quasigeodesic could be unrolled to a straight line on a plane, as we will explain shortly.

To describe geodesics and quasigeodesics explicitly, we adopt the notation for faces and vertices displayed in Fig. 1(a). Note that we label vertices in figures by their index i, but refer to them in the text as v_i.

Examples of quasigeodesics are shown in Fig. 1(b). Let α_i be the surface angle to the left of a quasigeodesic q passing through v_i, and β_i the angle to the right. The ***curvature*** at v_i is $2\pi - (\alpha_i + \beta_i)$, i.e., it is the ***angle gap*** with respect to 2π. Each vertex of the cube has curvature $\pi/2$.

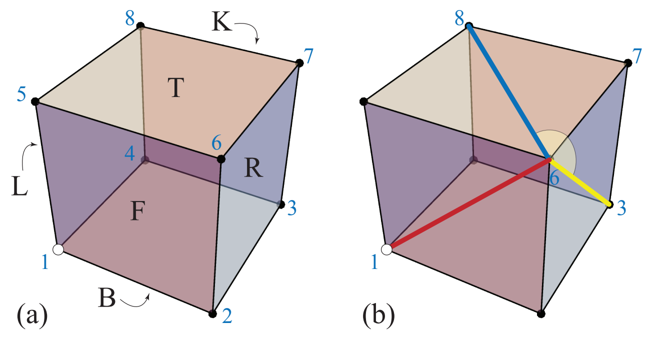


Figure 1: (a) F,R,T,K,L,B = Front, Right, Top, backK, Left, Bottom. B vertices indexed $1, 2, 3, 4$; T vertices indexed $5, 6, 7, 8$. (b) All the paths counterclockwise between $v_1v_6v_3$ to $v_1v_6v_8$ are quasigeodesics. For example, $v_1v_6v_3$ (red/blue) has $\alpha_6 = \pi/2$ to the left and $\beta_6 = \pi$ to the right.

The Gauss-Bonnet theorem implies that the total curvature (the sum of all vertex curvatures) is 4π. A simple closed geodesic partitions this curvature exactly into $2\pi : 2\pi$, as one can verify in Fig. 2. But most convex polyhedra have no such geodesic [9]. Nevertheless, there is a sense in which a simple closed quasigeodesic also partitions the curvature into equal halves. Let a quasigeodesic q have angles α and β to the left and right of a vertex v. Then one can consider q distributing $\pi - \alpha$ of its curvature to the left of q and $\pi - \beta$ curvature to the right. Then a simple closed quasigeodesic partitions this curvature into $2\pi : 2\pi$.

For example, consider the path bounding face T: v_5, v_6, v_7, v_8. This is a simple closed quasigeodesic Q, with $\alpha = \pi/2$ and $\beta = \pi$ at each of the four vertices. So, under the curvature distribution viewpoint, Q contributes $4 \cdot \pi/2$ to the left and none to the right, verifying the $2\pi : 2\pi$ partition.

*Artificial first author to highlight that the other authors (in alphabetical order) worked as an equal group.
†U. Mass. Lowell, hugo_akitaya@uml.edu
‡MIT, edemaine@mit.edu
§Harvard U., ahesterberg@seas.harvard.edu
¶thomas.hull@fandm.edu
‖U. Waterloo, alubiw@uwaterloo.ca
**MIT, jaysonl@mit.edu
††EPFL, klara.mundilova@epfl.ch
‡‡cnara@jeans.ocn.ne.jp
§§Smith College, jorourke@smith.edu
¶¶U. Mass. Lowell, frederick_stock@student.uml.edu
‖‖josef.tkadlec@iuuk.mff.cuni.cz
***uehara@jaist.ac.jp

It is then possible to view a simple closed quasigeodesic Q as partitioning convex polyhedron $\mathcal{P}$ into two ***half-surfaces***, each containing (after distribution) 2π of curvature. This partition is used in [10] to define a transformation achieved by twisting one half-surface with respect to the other, and used in [14] to find a nonoverlapping unfolding of $\mathcal{P}$ (among other results).

Returning to the rolling remark, because both $\alpha_i \leq \pi$ and $\beta_i \leq \pi$ at vertex v_i, it is possible to roll a polyhedron on a plane along a quasigeodesic straight through v_i by imagining "filling out" α_i and β_i to π each.

1.2 Historical Remarks

Ever since Poincaré's investigations more than a century ago, closed geodesics have played an important role in the topology of Riemannian manifolds [2, p. 433]. It is a famous 1929 theorem of Lyusternik-Schnirelmann that every smooth genus-0 surface has at least three simple (non-self-intersecting) closed geodesics [11]. Pogorelov proved in 1949 a natural analog: Every convex surface has at least three simple closed quasigeodesics [15]. Pogorelov's existence proof does not suggest a way to identify the three quasigeodesics, and it is only recently that finite algorithms have been proposed [5] [3].

Aside from these algorithms, simple closed quasigeodesics have only been systematically studied on tetrahedra [13], and, postdating this CCCG paper, on the regular icosahedron [12]. Two results in [13] are: (1) There is an open set in the space of all tetrahedra, each element of which has at least 34 simple closed quasigeodesics. (2) On any tetrahedron whose faces are not all congruent, there is at least one 1-vertex, one 2-vertex, and one 3-vertex simple closed quasigeodesic. In contrast to (2), it is known from [4] that the cube does not have a 1-vertex simple closed quasigeodesic, and from [7] that the regular icosahedron does not either.

Simple closed quasigeodesics play central roles in [10] and [14], and are of interest in their own right. But beyond their existence, much remains unknown. There is no known upper bound on the number of simple closed quasigeodesics on a given polyhedron, and there is an n-vertex polyhedron with $2^{\Omega(n)}$ distinct simple closed quasigeodesics [6, Sec. 24.4]. In contrast, it is known that isosceles tetrahedra[1] have arbitrarily long "spiraling" simple closed geodesics [16] [1].

1.3 Our Contribution

In this paper we make a complete inventory of simple closed quasigeodesics on a cube. It was known that there are precisely three simple closed geodesics on the cube. We identify a further 15 simple closed quasigeodesics (up to symmetries), and prove that this list is complete. We consider this proof to be our most significant contribution.

[1]Also called disphenoids, tetramonohedra, isotetrahedra, and several other names. All faces are congruent acute triangles.

1.4 Three simple closed geodesics

It has long been known that there are precisely three simple closed geodesics on the cube [8], displayed in Fig. 2.[2] Note that each of the three geodesics can slide

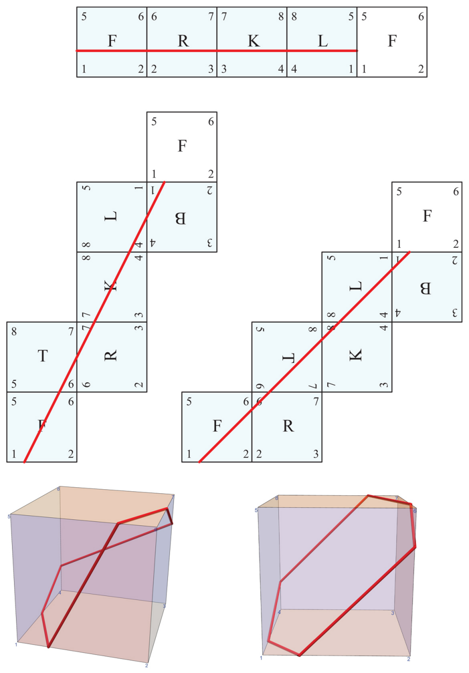


Figure 2: The three simple closed geodesics on a cube. The first is an equatorial band. The other two are as depicted.

within a range, maintaining parallelism. This is because each geodesic lies on a cylinder, with 2π curvature (four vertices, each with $\pi/2$ curvature) to each side.

2 Outline of Argument

We mentioned that simple closed geodesics can spiral around isosceles tetrahedra. A simple closed quasigeodesic also may spiral around other convex polyhedra, as shown in Fig. 3. A central aspect of our proof is to show that quasigeodesics cannot spiral on a cube.

Define a ***geodesic segment*** as a non-self-intersecting vertex-to-vertex geodesic (a geodesic whose enpoints are vertices).[3] A simple closed quasigeodesic is composed of a sequence of geodesic segments, satisfying the $\leq \pi$ condition to both sides at each vertex.

An instructive example was identified in [5]: a long box with a spiraling simple closed quasigeodesic. See

[2]Note these three are not the three from Pogorelov's theorem.

[3]In some literature, a geodesic segment is a shortest path between its endpoints. In this paper, our geodesic segments may or may not be shortest.

A_i:	4 quasigeos using only 0/1 segments.
B_i:	5 quasigeos using at least one 1/1 segment.
C_i:	4 quasigeos using at least one 1/2 segment.
D_1:	1 quasigeo using a single 1/3 segment.
E_1:	1 quasigeo using a single 2/3 segment.

Table 1: Description of the five categories of quasigeos.

Fig. 3. Each of the four marked vertices has π angle to one side and $\pi/2$ to the other side. Since there is freedom to partition the $3\pi/2$ surface angle differently (while maintaining $\leq \pi$ to each side), the number of spiraling simple closed quasigeodesics of a long box grows with the length of the long side of the box. A crucial property of spiraling is that some geodesic segment re-enters its initial face. For example, the blue geodesic segment from v_1 to v_2 in the figure starts on the long front-side face and later re-enters that face. We will prove that this cannot happen on a cube: a geodesic segment cannot return to its initial face, and in fact, cannot cross any face more than once.

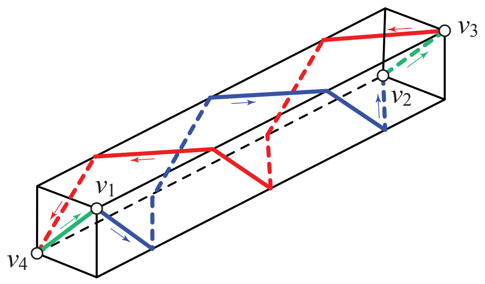


Figure 3: $(v_1, v_2, v_3, v_4, v_1)$ is a simple closed quasigeodesic. Based on Fig. 2 in [5].

3 Fifteen Simple Closed Quasigeodesics

Here is our main result:

Theorem 1 *There are exactly* 15 *simple closed quasigeodesics on the cube (beyond the three simple closed geodesics noted above). These are displayed in Fig. 4 and described in Table 1.*

As our sole focus in the remainder is on "simple closed quasigeodesics," we often simplify that term to ***quasigeos***.

The quasigeos are listed in order of the length of the geodesic segments comprising them, as described in Table 1. We identify a geodesic segment by its slope y/x, i.e., vertically up y units and rightward horizontally x units within the natural coordinate system of its starting face.

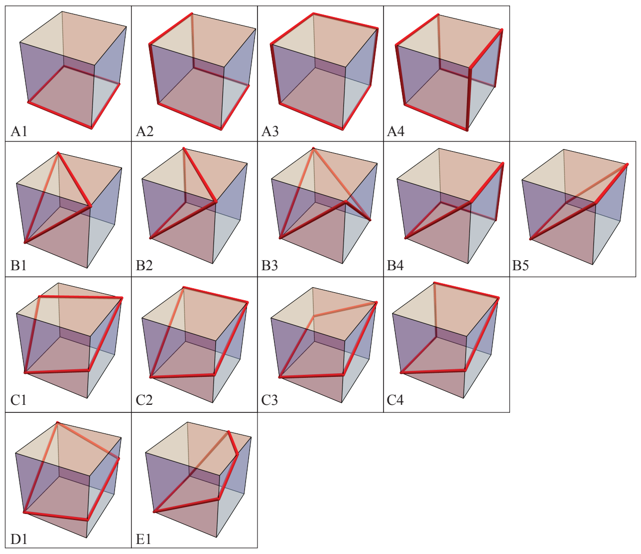


Figure 4: The 15 simple closed quasigeodesics.

4 Five Slopes

Our approach is to analyze a geodesic segment based on the angle α it makes in its starting face. Consider a geodesic segment that does not follow an edge of the cube. Then it enters the interior of a face and makes an angle in the range $(0, \pi/4]$ with one edge of the face. We express this as a slope in the range $(0, 1]$. We first rule out some slopes in this range because the geodesic segment revisits the first face and intersects itself there. We rule out further slopes by finding intersections between two geodesic segments. This reduces the possible slopes to a finite set, which allows a combinatorial enumeration of all simple closed quasigeodesics.

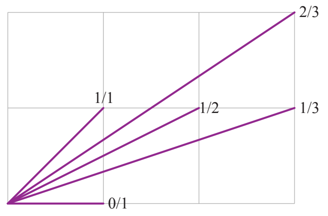


Figure 5: The five possible distinct slopes.

Lemma 2 *A geodesic segment that is a component of a simple closed quasigeodesic on the cube can only have one of the five slopes shown in Fig. 5:* $0/1, 1/3, 1/2, 2/3, 1/1$.

Corollary 3 *A geodesic segment that is a component of a simple closed quasigeodesic on the cube does not cross any face more than once.*

We prove the lemma by partitioning the rest of the slope range $(0, 1]$ into the following seven ranges:

- Case 1. $(0/1, 1/4]$
- Case 2. $(1/4, 1/3)$
- Case 3. $(1/3, 2/5)$
- Case 4. $[2/5, 1/2)$
- Case 5. $(1/2, 2/3)$
- Case 6. $(2/3, 3/4)$
- Case 7. $[3/4, 1/1)$

Fig. 6 shows the seven cases, and Fig. 7 shows how each range progresses on the unfolded surface of the cube. Each case has a (pink) ***F-cone*** with angle θ at v_1. From Fig. 7 we immediately obtain:

Claim 1 *No geodesic segment is possible in Cases 2, 3, and 6 because the segment revisits the starting face and intersects itself there. (We note that the crossing is at right angles, a known constraint [8].)*

The remaining four cases are possible for a single geodesic segment, but not for a geodesic segment that is part of a quasigeo.

Claim 2 *Consider a geodesic segment g that is a component of a simple closed quasigeodesic on the cube, and that falls into Case 1, 4, 5, or 7. Then g intersects another segment of the quasigeodesic.*

Proof. We find an intersection point by following the quasigeo backwards from v_1, the starting vertex of g. Let g' be the geodesic segment before g. We trace g' backwards from its terminus at v_1.

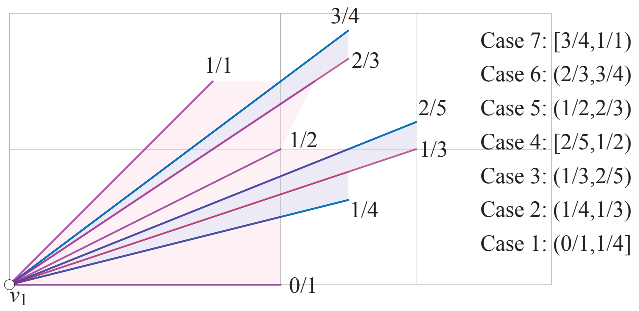


Figure 6: The seven slope ranges. Cases 2, 3, and 6 (in blue) are ruled out in Claim 1, and Cases 1, 4, 5, and 7 (in pink) are ruled out in Claim 2, leaving only the five slopes (in purple) allowed in Lemma 2.

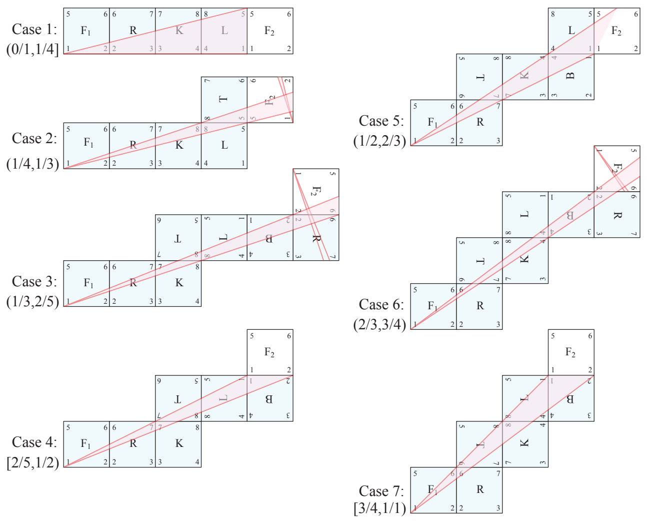


Figure 7: The seven slope cases, showing the range of slopes (in pink) progressing across the faces of the cube. The geodesic segment starts in face F_1 and revisits the starting face, marked F_2 (in white).

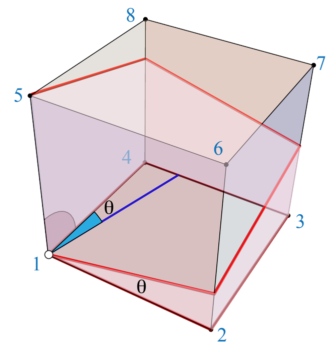


Figure 8: The Case 1 F-cone in 3D. $\theta = \arctan(1/4)$. Cf. Fig. 9.

Case 1. We focus on Case 1 in Fig. 9. The 2D unfolding of that case is shown on the 3D cube in Fig. 8. Cases 4, 5 and 7 will follow the same general scheme as Case 1.

View g as directed crossing faces F_1, R, K, L in order. In Case 1, g has slope in $(0/1, 1/4]$ and lies within the pink F-cone of angle θ as illustrated. We now show that g cannot be part of a simple closed quasigeodesic, by analyzing the possibilities for the previous geodesic segment g', i.e., the geodesic segment that closes the quasigeo as it revisits v_1.

Because the angle between g and g' at v_1 must be $\leq \pi$, g' must leave v_1 ("backwards") in a $\theta + \pi/2$ cone that extends counterclockwise $\pi/2$ from edge v_1v_5, and a further θ from edge v_1v_4. This cone is open along edge v_1v_5 and closed on its other boundary. We partition the cone into three possibilities:

(1) g' lies strictly within the quarter-circle on face L at v_1 (counterclockwise between edges v_1v_5 and v_1v_4). Then g' crosses g no matter where g and g' lie in their respective cones.

(2) g' lies in the cone of angle θ counterclockwise of edge v_1v_4. This cone (colored blue in Fig. 8) is open along the edge v_1v_4 and closed on its other boundary. Then g' wraps clockwise around v_4 by $\pi/2$, and crosses g in face K.

(3) g' follows the edge v_1v_4. Then g' hits vertex v_4 and ends there. Let g'' be the next geodesic segment. Then g'' leaves v_4 in face K in the closed quarter-circle bounded by edges v_4v_8 and v_4v_3. Any g'' in this cone intersects g unless g'' follows the edge v_4v_3. Repeating this argument, we either find an intersection with g, or we eventually follow the edge v_2v_1—but then the angle with g at v_1 is too sharp for a quasigeodesic.

So we obtain a quasigeo violation for every g inside or on the upper boundary of the F-cone in Case 1.

The argument for the remaining cases proceeds similarly, presented below somewhat more concisely.

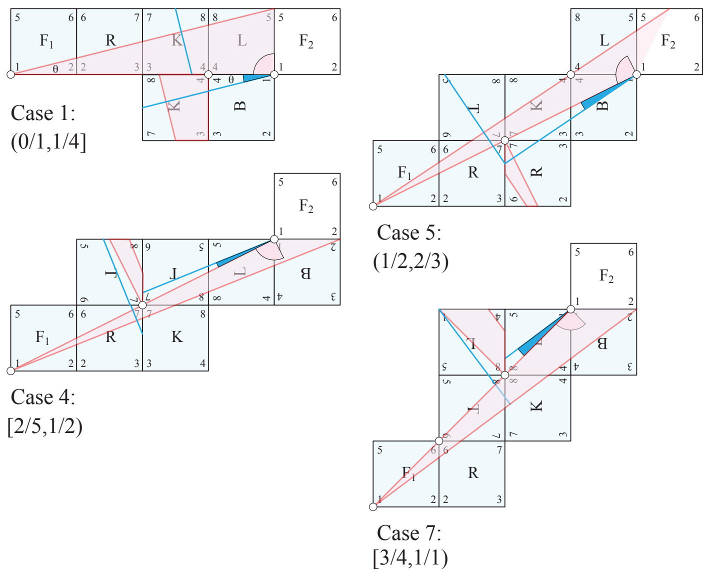


Figure 9: The geodesic segment g in the F-cone is crossed by g', either if starting backwards in the quarter-circle, or starting as much as θ beyond (blue angle and segment.), where θ is the F-cone angle at v_1.

The argument for the remaining cases proceeds similarly, presented below somewhat more concisely. In the three remaining cases, we likewise consider F-cones with angle θ at v_1 in F_1 and preceding geodesics g' that leave v_1 in a $\theta + \pi/2$ cone in F_2.

Case 4. There are three options for g':

(1) g' lies strictly within the quarter-circle on faces B and L. Then g' crosses g no matter where they lie in their cones.

(2) g' lies in the cone of angle θ strictly clockwise of the upper boundary of the F-cone. Then g' wraps counterclockwise about v_7 and crosses g in face R.

(3) g' follows the upper F-cone edge (slope 1/2). Then g' hits v_7. As in Case 1, repeating the argument, the next geodesic segment g'' leaves the quarter-circle similarly anchored on v_7 and either crosses g in the F-cone, or hits v_1 at an angle too sharp for a quasigeodesic.

Case 5.

(1) g' lies strictly within the quarter-circle on faces B and L. Then g' crosses g no matter where they lie in their cones.

(2) g' lies in the cone of angle θ strictly counterclockwise of the lower boundary of the F-cone. Then g' wraps clockwise about v_7 and crosses g in face R or T.

(3) g' follows the lower F-cone edge (slope 1/2). Then g' hits v_7. Repeating the arguments of the previous cases, the next geodesic segment g'' leaves the quarter-circle anchored on v_7 and either crosses g in the F-cone, or hits v_1 at an angle too sharp for a quasigeodesic.

Case 7.

(1) g' lies strictly within the quarter-circle on faces B and L. Then g' crosses g no matter where they lie in their cones.

(2) g' lies in the cone of angle θ strictly clockwise of the upper boundary of the F-cone. Then g' wraps counterclockwise about v_8 and crosses g in face L or K.

(3) g' follows the upper F-cone edge (slope 1/1). Then g' hits v_8. We repeat the previous arguments. The next geodesic segment g'' leaves the quarter-circle anchored on v_8 and either crosses g in the F-cone, or hits v_6. Applying the argument again, the next geodesic segment g''' either crosses g or or hits v_1 at an angle too sharp for a quasigeodesic.

This completes the proof of Claim 2. □

Claims 1 and 2 establish that, of the seven cases filling the entire range of slopes (Fig. 6), all but the five identified slopes are impossible, and so prove Lemma 2 and Corollary 3.

5 Search for Quasigeos

We initially found the 15 quasigeos in Fig. 4 "by hand." To establish that there are no other possibilities, we programmed an exhaustive search based on Lemma 2. We chose to use a DFS search, starting with the longest geodesic segments first, because they maximize pruning. Ordered by lengths, the slopes are $2/3 > 1/3 > 1/2 > 1/1 > 0/1$: $\sqrt{13}, \sqrt{10}, \sqrt{5}, \sqrt{2}, 1$. With this ordering, the first quasigeo found by the DFS was E1 in Fig. 4, employing one geodesic segment of slope 2/3 and one cube edge. And the last quasigeo found was A3, composed of six cube edges.

We implemented the DFS to first eliminate a possible next geodesic segment based on the lengths ordering (which resulted in 1070 eliminations), then on illegal turn angle (225 eliminations out of 582 checks), and finally crossing geodesic segments (174 eliminations out of 353 checks). Examples of pruning geodesic segments based on crossing are shown in Fig. 10.

The DFS found 29 quasigeos, and after eliminating the duplicates congruent by a symmetry, exactly the 15 in Fig. 4 remain.[4]

Recall Corollary 3 established that no single geodesic segment part of a cube quasigeo can cross a face more than once. This contrasts with the long box example, Fig. 3. A consequence of the inventory of the 15 quasigeos is that no cube quasigeo can cross a face more than once.

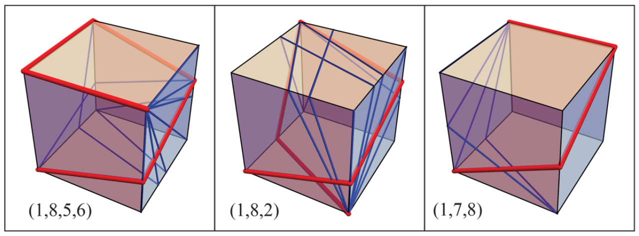


Figure 10: Red: Partial quasigeo, through vertices listed. Blue: All possible extension geodesic segments that pass the lengths ordering and the angle-turn eliminations, leaving only self-crossing to check.

6 Discussion and Open Problems

We have proved Theorem 1 by verifying that the list in Fig. 4 is exhaustive. Below we list several open questions.

(1) It would be useful to avoid the exhaustive search with a geometric proof that could be applied to convex polyhedra beyond the cube.

(2) Is there a finite upper bound to the number of simple closed quasigeodesics (that are not geodesics) on a given nondegenerate polyhedron of n vertices? There is no such bound for simple closed geodesics. Nor is there a bound for (degenerate) doubly-covered squares: see Fig. 11.

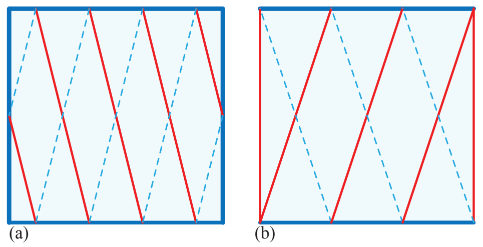


Figure 11: Doubly-covered square. Red segments: front. Blue segments: back. (a) Simple closed geodesic. (b) Simple closed quasigeodesic.

(3) It was proved in [13] that every tetrahedron has a simple closed geodesic or a 1-vertex simple closed quasigeodesic. That the same holds for any convex polyhedron was conjectured in [14]. As mentioned, it is known from [4] that the cube does not have a 1-vertex simple closed quasigeodesic, but it does have simple closed geodesics, so the cube accords with the conjecture. Settling the conjecture either way seems currently out of reach.

(4) Matters become more complicated for non-cubical boxes. Fig. 12(a) shows a twisted quasigeo on a $1 \times 1 \times 3$ box. The slightly non-cubical box in (b) of the figure has a "diamond" 1-vertex simple closed quasigeodesic. Characterizing all simple closed quasigeodesics on boxes is a natural next step.

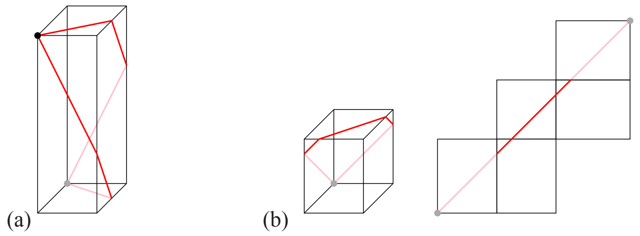


Figure 12: (a) A twisted quasigeo on a $1 \times 1 \times 3$ box. (b) A 1-vertex quasigeo on a $1 \times 1 \times 1\frac{1}{4}$ box.

Acknowledgements. We appreciate the insightful reviews of three referees. H. Akitaya and F. Stock were supported by NSF grant CCF-2348067.

[4]We have not made our code available, but it is an easy programming exercise to verify our exhaustive search.

References

[1] A. Akopyan and A. Petrunin. Long geodesics on convex surfaces. *Mathematical Intelligencer*, 40:26–31, 2018.

[2] M. Berger. *A Panoramic View of Riemannian Geometry.* Springer-Verlag, 2003.

[3] J. Chartier and A. de Mesmay. Finding weakly simple closed quasigeodesics on polyhedral spheres. *Discr. Comput. Geom.*, 71:95–120, 2024.

[4] D. Davis, V. Dods, C. Traub, and J. Yang. Geodesics on the regular tetrahedron and the cube. *Discrete Mathematics*, 340(1):3183–3196, 2017.

[5] E. D. Demaine, A. C. Hesterberg, and J. S. Ku. Finding closed quasigeodesics on convex polyhedra. In *Proc. 36th Annu. ACM Sympos. Comput. Geom.*, pages 33:1–33:13, 2020.

[6] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge University Press, 2007.

[7] D. Fuchs. Geodesics on regular polyhedra with endpoints at the vertices. *Arnold Math. J.*, 2(2):201–211, 2016.

[8] D. B. Fuchs and E. Fuchs. Closed geodesics on regular polyhedra. *Moscow Math. J.*, 7(2):265–279, 2007.

[9] P. Gruber. A typical convex surface contains no closed geodesic. *J. Reine Angew. Math.*, 416:195–205, 1991.

[10] T. C. Hull, A. Lubiw, K. Mundilova, C. Nara, J. O'Rourke, J. Tkadlec, and R. Uehara. Quasi-twisting convex polyhedra. In *34th Canad. Conf. Comput. Geom.*, pages 270–276, Aug. 2022.

[11] L. Lyusternik and L. Schnirelmann. Sur le problém de trois géodésiques fermées sur les surfaces de genre 0. *C. R. Acad. Sci. Paris*, 189:269–271, 1929.

[12] C. Nara and J. O'Rourke. Quasigeodesics on the regular icosahedron. 2-page abstract submitted to 27th China-Japan Conf. Discr. Comput. Geom., Graphs, and Games, Jun 2025.

[13] J. O'Rourke and C. Vîlcu. Simple closed quasi-geodesics on tetrahedra. *Information*, 13:238–258, May 2022.

[14] J. O'Rourke and C. Vîlcu. *Reshaping Convex Polyhedra.* Springer, 2024.

[15] A. V. Pogorelov. Quasi-geodesic lines on a convex surface. *Mat. Sb.*, 25(62):275–306, 1949. English transl., *Amer. Math. Soc. Transl.* 74, 1952.

[16] V. Y. Protasov. Closed geodesics on the surface of a simplex. *Sbornik: Mathematics*, 198(2):243, 2007.

Decremental Greedy Polygons and Polyhedra Without Sharp Angles

David Eppstein[*]

Abstract

We show that the max-min-angle polygon in a planar point set can be found in time $O(n \log n)$ and a max-min-solid-angle convex polyhedron in a three-dimensional point set can be found in time $O(n^2)$. We also study the maxmin-angle polygonal curve in 3d, which we show to be NP-hard to find if repetitions are forbidden but can be found in near-cubic time if repeated vertices or line segments are allowed, by reducing the problem to finding a bottleneck cycle in a graph. We formalize a class of problems on which a decremental greedy algorithm can be guaranteed to find an optimal solution, generalizing our max-min-angle and bottleneck cycle algorithms, together with a known algorithm for graph degeneracy.

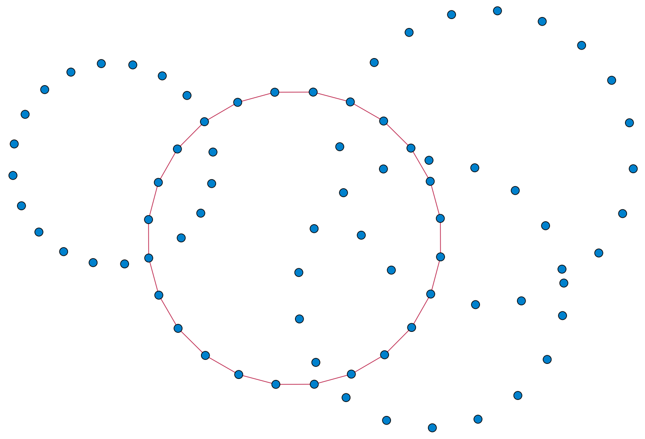


Figure 1: The maxmin-angle simple polygon in a given set of points

1 Introduction

In this work, we study the natural problem of finding a polygon with vertices drawn from n given points, maximizing its minimum (sharpest) angle (Fig. 1). As we show, there exists an optimal polygon that is convex. To find it, we define the quality of a point, in a given subset, to be 2π if it is interior to the convex hull of the subset, or its interior angle if it is a vertex of the convex hull. This quality is monotonic: as we delete points, the quality of any remaining point can only decrease, as it becomes a hull vertex or as it loses hull neighbors. Therefore, it is safe to delete the point of minimum quality: any better polygon than the convex hull of the current subset cannot include the deleted point. A greedy algorithm that repeatedly deletes the sharpest hull vertex, and then returns the best polygon found throughout this deletion process, finds the maxmin-angle polygon in time $O(n \log n)$. After detailing this method we extend it to analogous problems of finding the max-min-solid angle convex polyhedron in 3d We reduce max-min-angle polygons in 3d to finding bottleneck cycles in graphs, to which we apply related decremental greedy algorithms.

The decremental greedy nature of our algorithms both for geometry problems (max-min angle polygons and polyhedra) and graph problems (bottleneck cycles) suggests that they share a common generalization. We formalize a class of bottleneck optimization problems that includes these problems and can be solved optimally by decremental greedy algorithms. Our formalization also encompasses the known problem of computing graph degeneracy, the minimum degree of a vertex in a subgraph chosen to maximize this degree. A classical linear-time algorithm for degeneracy [19] repeatedly removes a minimum-degree vertex until a given graph becomes empty; the degeneracy equals the maximum of the degrees of the vertices at the times of their removal. Generalizing convex hull angles and vertex degrees to other measures of element quality, we define the *bottleneck subset problem*, in which we seek a (nonempty) subset of a given set of elements whose worst element is as good as possible, according to a quality measure that can only worsen as other elements are removed. As we show, these problems can be solved by a decremental greedy algorithm that repeatedly removes the lowest-quality element.

Although greedy algorithms are commonly associated with matroids, our formalization does not apply to decremental greedy algorithms for a max-min matroid base such as a maximum spanning tree, nor to greedy algorithms such as Dijkstra's algorithm. Intuitively, this difference comes from the direction of monotonicity. In decremental greedy matroid algorithms, elements become more valuable as other elements are removed and they become needed to complete a base, and in Dijkstra's algorithm, vertices in the priority queue become more valuable as better paths are found to reach them. In contrast, in the algorithms we study, elements become

[*]Department of Computer Science, University of California, Irvine. Research supported in part by NSF grant CCF-2212129.

less valuable as other elements are removed. Our formalization is related to a different type of greedoid, an *antimatroid* [17]; see discussion following Theorem 1.

1.1 New results

Section 2 formalizes the bottleneck subset problem for monotonic quality measures, and describes the general decremental greedy algorithm for solving this problem. We provide in Section 3 the following applications to geometric optimization:

- We prove that a maxmin-angle simple polygon in a planar point set (Fig. 1) can be chosen to be a convex polygon, and we show how to find it in time $O(n \log n)$.
- We prove that a maxmin-solid angle polyhedral surface in a 3d point set can be chosen to be a convex polyhedron, and we show how to find it in time $O(n^2)$.
- For a 3d point set, the maxmin-angle closed polygonal curve (by which we mean a cyclic sequence of line segments meeting end to end) may intersect at points or even entire line segments, so we do not call it a polygon. Even if it is a polygon, it may use points interior to its convex hull, or may be knotted. We show how to find a maxmin-angle closed polygonal curve in time $O(n^3 \log^* n)$, or in time $O\big(n^3(\log n \log\log n)^2\big)$ if repeated points are allowed but repeated line segments are forbidden.

For space reasons we relegate additional results to the full version of this paper, including details on the antimatroid nature of the removals performed by the decremental greedy algorithm, on the NP-hardness of finding 3d maxmin-angle non-self-intersecting curves, and on algorithms for bottleneck cycles in graphs.

Those graph algorithms are used for our 3d polygonal curve algorithms, so for completeness we state them here. For undirected graphs the bottleneck cycle problem has an easy non-decremental linear time algorithm. For other types of graphs, it becomes more complicated:

- In a directed graph with m edges, or a mixed graph with m directed and undirected edges, we show how to find a bottleneck cycle in time $O(m \log^* m)$.
- We also consider *polar graphs* or *switch graphs*, in which each vertex has two poles at which edges attach, and a *regular cycle* must pass through both poles of each of its vertices (Fig. 2). We show how to find a bottleneck regular cycle in such a graph in time $O\big(m(\log m \log\log m)^2\big)$.

1.2 Related work

As far as we know the graph bottleneck cycle problems that we study are novel, but bottleneck path and bottleneck spanning tree problems were already studied by Pollack in 1960. The maximum spanning tree follows undirected bottleneck paths, and a variant of Dijkstra's algorithm constructs directed bottleneck paths [23]. Additional algorithms for these problems are known [10,14], and they have several applications [12,24,26].

Connecting given points into curves or surfaces has been studied with the goal of reconstructing an unknown shape from sparse samples [1,3,21], in some cases assuming that the curve is sampled densely enough to cause all angles to be close to π [1]. The problems that we study have a similar flavor, but for curves through a subset of points rather than requiring a curve to pass through or near all points. We are unaware of previous work using the max-min-angle optimization criterion for curves and surfaces, but this criterion is well known in computational geometry in the context of Delaunay triangulations, which maximize the minimum angle among triangulations of given planar points [25], and has also been applied to other forms of triangulation [2,16,20]. Maximizing the minimum angle is also important in graph drawing, where the minimum angle at any vertex of a graph drawing is its *angular resolution* [13,18].

Dynamic programming can find convex polygons with vertices in a planar point set, optimizing a broad range of criteria [4,9,11]. However, we do not require the maxmin-angle polygon to be convex (instead this is an emergent property of the result) and our algorithm is simpler and faster than the known dynamic programming methods. It is not obvious how to generalize the dynamic programs for optimal convex polygons to curves or surfaces in higher dimensions.

2 Formalization

We define a *monotone bottleneck subset problem* to consist of a set U of elements, and a function $q(x, S)$ that takes as input a pair (x, S) where $x \in S$ and $S \subset U$, and produces as output a real number, the quality of x as a member of S. We require q to be monotone: whenever $x \in S \subset T \subset U$, we have $q(x, S) \leq q(x, T)$. Intuitively, removing other elements from a subset containing x causes the quality of x to decrease or stay the same. We define the quality $Q(S)$ of a nonempty subset S to be $Q(S) = \min_{x \in S} q(x, S)$, the least quality of a member of S, with $Q(\emptyset) = -\infty$, a flag value preventing the empty set from being optimal. Our goal is to find a nonempty subset whose quality is maximum, which we call the *bottleneck subset*. If x is a lowest-quality element of a subset S we call x a *bottleneck element* of S.

The *decremental greedy algorithm* for a monotone bottleneck subset problem performs the following steps:

```
1. Initialize two sets $S$ and $T$ to both equal $U$.
2. While $S$ is not empty, repeat the following steps:
    • If $Q(S) > Q(T)$, set $T = S$.
    • Find any $x$ with $q(x, S) \le Q(T)$.
    • Remove $x$ from $S$.
3. Return $T$.
```

Any bottleneck element x of S satisfies $q(x, S) \le Q(T)$, so the algorithm always has an element that it can remove, although we do not require it to remove a bottleneck element at each step. When the bottleneck quality β is already known, a simpler decremental algorithm can find the bottleneck subset: repeatedly remove any element of quality less than β until all remaining elements have greater value. Call this the *known-β algorithm.*

Theorem 1. *Every monotone bottleneck subset problem has a unique maximal bottleneck subset. Regardless of the choices they make at each step, the decremental greedy algorithm and known-β algorithm both always find and return the maximal bottleneck subset.*

Proof. The union M of all bottleneck subsets is a bottleneck subset: each $x \in M$ belongs to a bottleneck subset $X \subset M$, so $q(x, M) \ge q(x, X) \ge \beta$. Because this is true for all elements of M, $Q(M) \ge \beta$, the optimal value. M is the unique maximal bottleneck subset, because it is a superset of all other bottleneck subsets.

Because each proper superset $V \supset M$ is not a bottleneck subset, some element $y \in V$ has $q(y, V) < \beta$, and any such y cannot be in M by monotonicity of its quality. Therefore, until M is reached, the known-β algorithm can and will remove an element of the complement of M. And until M is reached, the decremental greedy algorithm will have $Q(T) < \beta$ and the element x that it removes will have $q(x, S) < \beta$; again, $x \notin M$ by monotonicity of its quality. Therefore, until M is reached, the decremental greedy algorithm can and will remove an element of the complement of M.

Once M is reached, the known-β algorithm terminates and returns it. The decremental greedy algorithm records M as the set T that it will eventually return; it can never subsequently change T to another set because no other set has better quality. □

Both algorithms make arbitrary choices that can cause them to produce different removal sequences before reaching U. Their families of allowed removal sequences form *antimatroids*, structures that formalize the familiar introductory programming concept of a ready list. Antimatroids can be defined as families of sequences generated by a process that repeatedly appends an arbitrary "available" element from a given set, under the constraint that availability is determined by a monotonic function of the elements that have already been appended. Less formally, once an element becomes available, it remains available until it is appended itself, and availability depends only on the set of elements that have been chosen, not on their order. For the known-β algorithm, the antimatroid property is straightforward (the property of having quality at most β is determined by a monotonic function, as required) but for the decremental greedy algorithm, it requires a proof; see the full version of this paper.

According to the monotonicity that we require our quality measures to satisfy, each element gets worse (prioritized for earlier removal) as other elements around it are removed. This behavior should be contrasted with the quality of elements in max-min matroid problems such as the maximum spanning tree, which can also be solved by a decremental algorithm (remove the minimum-weight non-bridge edge until all remaining edges are bridges). In the decremental maximum spanning tree algorithm, an element (an edge) either keeps its priority (its weight) or gets a better priority (it becomes an unremovable bridge) as the algorithm progresses, rather than getting worse, so the decremental greedy maximum spanning tree algorithm does not fit into our framework.

Finding graph degeneracy is a bottleneck subset problem where U is the set of vertices of a given graph G and $q(x, S) = \deg_{G[S]} x$ is the degree of vertex x in the subgraph induced by S. The bottleneck elements of any induced subgraph are its minimum-degree vertices. The bottleneck subset is a set of vertices that induces a subgraph maximizing its minimum degree. The linear-time degeneracy algorithm of Matula and Beck [19] repeatedly removes a minimum-degree vertex, as a special case of the decremental greedy algorithm for this quality. However, even for graph degeneracy, the antimatroid nature of the decremental greedy algorithm appears to be novel. To achieve this antimatroid property, we require a more general algorithm, allowing the removal of any vertex whose degree is at most the current quality bound, rather than the special case that only removes minimum-degree vertices.

3 Geometric applications

3.1 Polygons in 2d

Lemma 2. *Let S be a finite set of points in the plane. Then there exists a convex polygon P with vertices in S that maximizes the minimum angle among all closed polygonal curves (allowing repeated vertices and edges) with vertices in S.*

Proof. Because S determines finitely many angles, the max-min angle among closed polygonal curves exists. Let W be any closed polygonal curve through S attaining this angle, and let P be the convex hull of W. Then, compared to W, P may omit some vertices and may increase the angle of the others that remain; both of these changes can only increase the sharpest angle in P, relative to the sharpest angle in W. Therefore, P is also a max-min angle closed polygonal curves, as was stated to exist in the lemma. □

Theorem 3. *Let S be a finite set of points in the plane. Then in time $O(n \log n)$ we can find a convex polygon P with vertices in S that maximizes the minimum angle among all closed polygonal curves in S.*

Proof. We perform the following steps:

1. Initialize a dynamic convex hull data structure to contain all points of S.
2. Initialize parameters θ to the sharpest angle of the hull, and s to zero. These parameters will store the best angle found so far, and the number of removed points corresponding to that best angle.
3. Initialize an empty list L of removed points.
4. While the current convex hull is non-degenerate (it has more than two vertices), repeat the following steps:
   - Find and remove from the dynamic hull the vertex with the sharpest angle (choosing arbitrarily any vertex of sharpest angle in case of ties) and append this vertex to L.
   - Set θ to the maximum of its previous value and the sharpest angle of the current hull, and if the result is an increase in θ then set s to the current length of L.
5. Return the convex hull of the point set obtained from S by removing the first s points of L.

This is a decremental greedy algorithm where the quality of a point is its interior angle, if it is a convex hull vertex, or 2π otherwise. This quality is monotonic and the algorithm finds a max-min angle convex polygon by Theorem 1. This polygon is also a max-min angle closed polygonal curve by Theorem 2.

The sharpest angle in the current hull can be maintained during this algorithm using a priority queue of the current convex hull vertices and their angles, updated whenever the dynamic convex hull structure adds a vertex to the hull or changes the neighbor of an existing vertex. This structure requires $O(n)$ updates over the course of the algorithm and therefore takes $O(n \log n)$ time. Decremental or fully dynamic convex hull data structures that take $O(\log n)$ time per update are also known [5, 15], leading to an $O(n \log n)$ time bound for that part of the algorithm as well. □

3.2 Polyhedra in 3d

In this section we seek a polyhedral surface, with vertices at a subset of a given point sets, maximizing the solid angle interior to the surface as viewed from any point (or vertex) of the surface. To avoid definitional issues we consider only non-self-intersecting surfaces. Analogously to the results of the previous section, we have:

Lemma 4. *Among non-self-intersecting polyhedral surfaces having vertices at a subset of given points in $\mathbb{R}^3$, there exists a convex polyhedron that maximizes the minimum solid angle.*

Proof. As in Theorem 2, consider any polyhedral surface that maximizes the minimum solid angle, and take its convex hull. This can only remove vertices and improve the solid angle at the remaining vertices, so it must also maximize the minimum solid angle. □

As in the two-dimensional case, we will need a data structure for decremental convex hulls. Chan has studied this problem [7, 8], but his algorithms do not represent the hull explicitly, instead using an implicit representation that allows only extreme point queries. In our case, we need to find the solid angles of the vertices of the hull, not possible with Chan's structure. Another data structure, of Buchin and Mulzer [6] allows hulls of any subset of an input point set to be computed in randomized expected time $O\big(n(\log \log n)^2\big)$ on a word RAM. Instead, we use a simpler method for maintaining three-dimensional convex hulls explicitly, in time $O(n)$ per point deletion. Such a data structure was described by Overmars [22] (Theorem 6.4.1.6, p. 90), but to keep the presentation self-contained we outline a simplified version. The simplified data structure consists of a balanced binary search tree for the x-coordinate order of the given points, together with explicitly represented hulls of the sets of not-yet-deleted points in each subtree. After each deletion, each changed hull can be computed by merging two child hulls in time linear in its subtree. The total time for all merges adds in a geometric series to $O(n)$ per deletion. The space for this data structure is $O(n \log n)$. The version of Overmars improves the space to $O(n \log \log n)$ by storing only the hulls of large subsets of points, and makes the data structure fully dynamic rather than decremental using weight-balanced trees rather than static balanced trees, but we do not need those advances.

Theorem 5. *Let S be a finite set of points in $\mathbb{R}^3$. Then in time $O(n^2)$ we can find a convex polyhedron P with vertices in S that maximizes the minimum solid angle among all non-self-intersecting polyhedral surfaces having vertices at a subset of S.*

Proof. We perform the following steps:

1. Initialize a dynamic convex hull data structure for the points of S.
2. Initialize parameters θ to the sharpest solid angle of the hull, and s to zero. These parameters will store the best angle found so far, and the number of removed points corresponding to that best angle.
3. Initialize a list L of removed points to an empty list.
4. While the current convex hull is non-degenerate (it has nonzero volume), repeat the following steps:
   - Find and remove the vertex with the sharpest solid angle (choosing arbitrarily any vertex of sharpest angle in case of ties) and append this vertex to L.
   - Set θ to the maximum of its previous value and the sharpest angle of the current hull, and if the result is an increase in θ then set s to the current length of L.
   - Update the hull of the remaining points.
5. Return the convex hull of the point set obtained from S by removing the first s points of L.

Using the dynamic hull data structure outlined above, each iteration of the main loop takes time $O(n)$, and the whole algorithm takes time $O(n^2)$. □

3.3 Closed polygonal curves in 3d

To model the search for a closed polygonal curves through a given system of line segments in 3d, allowing repeated vertices but without allowing any line segment to be repeated, we use *polar graphs*. These are graphs in which the edges are undirected, but are attached to one of two *poles* of each vertex. A *regular cycle* in such a graph is a simple cycle for which the two edges incident to each vertex are attached to different poles: if the cycle enters a vertex via one pole, it must exit via the other pole. In the full version of this paper we describe a decremental greedy algorithm for finding a minmax-weight regular cycle in a polar graph with m edges in time $O(m(\log m \log\log m)^2)$. For a given set S of points in $\mathbb{R}^3$, we may define a weighted polar graph $G(S)$, as follows (Fig. 2):

- The vertices of $G(S)$ are the line segments determined by pairs of points in S.
- The two poles of each vertex of $G(S)$ are the two endpoints of the corresponding line segment.
- The edges of $G(S)$ connect pairs of line segments that form a three-point polygonal chain and are weighted by the angle at the middle point of the chain.

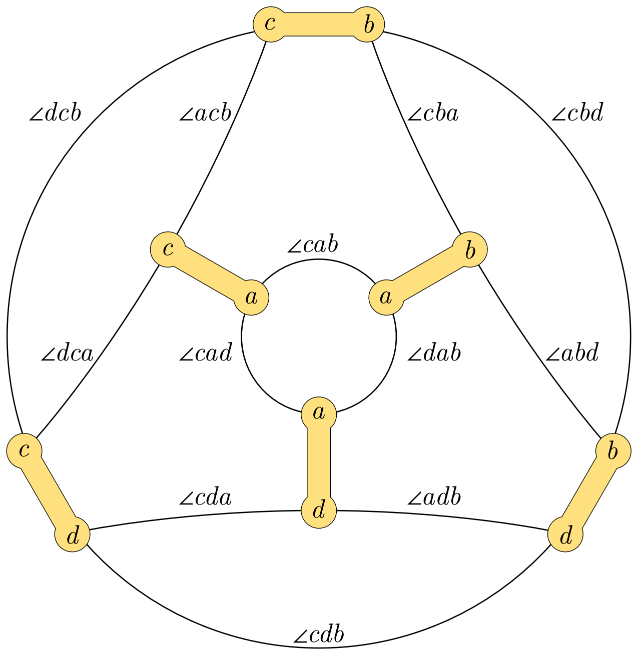


Figure 2: A polar graph with six vertices representing the line segments determined by pairs of four points a, b, c, and d, and 12 edges labeled by the angles determined by triples of points. The two poles of each vertex are labeled by the points. A regular cycle must enter and exit the vertices it visits via opposite poles.

- For each vertex and incident edge in $G(S)$, the pole of the vertex to which the edge is attached is the middle point of the three-point polygonal chain that defines the edge.

Then, a regular cycle in this graph corresponds to a closed polygonal curve in 3d, allowing repeated points but not allowing repeated line segments. The minimum edge weight in this cycle is the sharpest angle of the curve. If repeated line segments are to be allowed, we can instead replace the polar graph by an ordinary graph, its *double cover*. This has two copies of each vertex of the polar graph, and two directed edges for each edge of the polar graph, in each direction. Each copy of a vertex is incident to the incoming directed edges for one pole of the vertex and to the outgoing directed edges for the other pole. The resulting directed graph is *skew-symmetric* (the vertex bijection that swaps the two copies of each polar vertex also reverses all the directed edges) and has been used in algorithms for polar graphs. When a directed cycle in the double cover of $G(S)$ uses both copies of a vertex, it corresponds to a closed polygonal curve that uses a line segment twice, in both directions. In the full version of this paper we describe a decremental greedy algorithm for finding a bottleneck directed cycle in a directed graph in time $O(m \log^* m)$.

Theorem 6. *We consider maxmin-angle closed polygo-*

nal curves in a 3d point set that may use points interior to its convex hull, may be knotted, and may self-intersect. If both repeated points and repeated line segments are allowed, we can find such a curve in time $O(n^3 \log^* n)$. If repeated points but not repeated line segments are allowed, we can find it in time $O\big(n^3(\log n \log\log n)^2\big)$.

Proof. We construct $G(S)$, with $O(n^3)$ edges, and then find either a bottleneck directed cycle in the double cover of $G(S)$, or a bottleneck regular cycle in $G(S)$, according to whether we allow or disallow repeated segments, respectively. □

For completeness, we state:

Theorem 7. *It is* NP*-complete to find the maxmin-angle simple polygon for points in $\mathbb{R}^3$, for points on a unit sphere given by rational Euler angles.*

We defer the proof to the full version of this paper.

4 Conclusions

We have formalized a broad family of decremental greedy problems for monotone bottleneck subset problems, generalizing existing graph degeneracy algorithms, and shown its applicability in developing new algorithms in computational geometry and graphs. We have found simple greedy decremental algorithms for several maxmin-angle problems in geometric optimization: finding a planar polygon through given points maximizing the minimum angle, finding a 3d polyhedral surface through given pints maximizing the minimum solid angle, and finding a (self-intersecting) 3d closed polygonal curve through given points maximizing the minimum angle. On the other hand, constraining the 3d polygonal curve to be non-self-intersecting appears to make the problem computationally infeasible.

There appears to be room for improvement in our polyhedral surface and 3d curve time bounds. It would also be of interest to find other problems to which the same greedy decremental approach applies.

References

[1] N. Amenta, M. W. Bern, and D. Eppstein. The crust and the β-skeleton: combinatorial curve reconstruction. *Graphical Models & Image Processing* 60/2(2):125–135, 1998, doi:10.1006/gmip.1998.0465.

[2] N. Amenta, M. W. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. *J. Algorithms* 30(2):302–322, 1999, doi:10.1006/jagm.1998.0984.

[3] D. Attali. r-regular shape reconstruction from unorganized points. *Comput. Geom.* 10(4):239–247, 1998, doi:10.1016/S0925-7721(98)00013-3.

[4] D. Avis and D. Rappaport. Computing the largest empty convex subset of a set of points. *Proc. 1st Symp. on Computational Geometry (SOCG '85)*, pp. 161–167. ACM, 1985, doi:10.1145/323233.323255.

[5] G. S. Brodal and R. Jacob. Dynamic planar convex hull. *Proc. 43rd Symp. Foundations of Computer Science (FOCS 2002)*, pp. 617–626. IEEE Computer Society, 2002, doi:10.1109/SFCS.2002.1181985.

[6] K. Buchin and W. Mulzer. Delaunay triangulations in $O\big(\mathrm{sort}(n)\big)$ time and more. *J. ACM* 58(2):1–27, 2011, doi:10.1145/1944345.1944347.

[7] T. M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM* 57(3):1–15, 2010, doi:10.1145/1706591.1706596.

[8] T. M. Chan and K. Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete Comput. Geom.* 56(4):866–881, 2016, doi:10.1007/s00454-016-9784-4.

[9] V. Chvátal and G. Klincsek. Finding largest convex subsets. *Congressus Numerantium* 29:453–460, 1980, MR608447.

[10] R. Duan and S. Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. *Proc. 20th ACM-SIAM Symp. on Discrete Algorithms (SODA '09)*, pp. 384–391, 2009, https://portal.acm.org/citation.cfm?id=1496813.

[11] D. Eppstein, M. Overmars, G. Rote, and G. J. Woeginger. Finding minimum area k-gons. *Discrete Comput. Geom.* 7(1):45–58, 1992, doi:10.1007/BF02187823, MR1134451.

[12] E. Fernández, R. Garfinkel, and R. Arbiol. Mosaicking of aerial photographic maps via seams defined by bottleneck shortest paths. *Operations Research* 46(3):293–304, 1998, doi:10.1287/opre.46.3.293.

[13] M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F. T. Leighton, A. Symvonis, E. Welzl, and G. Woeginger. Drawing graphs in the plane with high resolution. *SIAM J. Comput.* 22(5):1035–1052, 1993, doi:10.1137/0222063, MR1237161.

[14] H. N. Gabow and R. E. Tarjan. Algorithms for two bottleneck optimization problems. *J. Algorithms* 9(3):411–417, 1988, doi:10.1016/0196-6774(88)90031-4, MR955149.

[15] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT* 32(2):249–267, June 1992, doi:10.1007/bf01994880.

[16] B. Joe. Delaunay versus max-min solid angle triangulations for three-dimensional mesh generation. *Int. J. Numer. Methods Eng.* 31(5):987–997, 1991, doi:10.1002/nme.1620310511.

[17] B. Korte, L. Lovász, and R. Schrader. Chapter III: Abstract Convexity – Antimatroids. *Greedoids*, pp. 19–43. Springer-Verlag, 1991, doi:10.1007/978-3-642-58191-5_3.

[18] S. Malitz and A. Papakostas. On the angular resolution of planar graphs. *SIAM J. Discrete Math.* 7(2):172–183, 1994, doi:10.1137/S0895480193242931, MR1271989.

[19] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM* 30(3):417–427, 1983, doi:10.1145/2402.322385, MR0709826.

[20] S. A. Mitchell. Approximating the maxmin-angle covering triangulation. *Comput. Geom.* 7(1–2):93–111, 1997, doi:10.1016/0925-7721(95)00046-1.

[21] S. Ohrhallinger, J. Peethambaran, A. D. Parakkat, T. K. Dey, and R. Muthuganapathy. 2D points curve reconstruction survey and benchmark. *Computer Graphics Forum* 40(2):611–632, 2021, doi:10.1111/cgf.142659.

[22] M. H. Overmars. *The Design of Dynamic Data Structures*. Lecture Notes in Computer Science 156. Springer, 1983, doi:10.1007/BFB0014927.

[23] M. Pollack. The maximum capacity through a network. *Operations Research]* 8(5):733–736, 1960, doi:10.1287/opre.8.5.733.

[24] M. Schulze. A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method. *Social Choice and Welfare* 36(2):267–303, 2011, doi:10.1007/s00355-010-0475-4.

[25] R. Sibson. Locally equiangular triangulations. *The Computer J.* 21(3):243–245, 1978, doi:10.1093/comjnl/21.3.243.

[26] E. Ullah, K. Lee, and S. Hassoun. An algorithm for identifying dominant-edge metabolic pathways. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2009)*, pp. 144–150, 2009, https://ieeexplore.ieee.org/document/5361299.

On t-fold Totally-Concave Polyominoes

Gill Barequet[*] Neal Madras[†] Johann Peters[‡]

Abstract

A *t-fold totally concave polyomino* (t-TCP) is an edge-wise connected collection of square cells with t or more gaps in every row and column. We prove that the minimum area of the smallest possible t-TCP is 21 for $t = 1$, 50 for $t = 2$, and $6(t+1)^2 - 1$ for $t > 2$. Answering a previous conjecture on the affirmative, we prove that the t-TCP counting sequence has the same leading exponential order as all polyominoes, from which we prove that the ratio of successive terms converges.

1 Introduction

A polyomino is an edge-wise connected collection of unit squares in the plane. That is, given a connected subgraph G of the square lattice (with nodes at integer coordinates), the polyomino determined by G is $P_G := \bigcup_{(x,y)\in G} [x, x+1] \times [y, y+1]$. To consider only one translate of each polyomino, we use the convention that every polyomino P satisfies $P \subset [0, \infty) \times [0, \infty)$, $P \cap (\{0\} \times \mathbb{R}) \neq \emptyset$, and $P \cap (\mathbb{R} \times \{0\}) \neq \emptyset$. A row or a column ξ of a polyomino has a *gap* if ξ contains at least two maximal sequences of consecutive cells; likewise, ξ has t gaps if it consists of at least $t+1$ maximal sequences of consecutive cells. *Totally Concave Polyominoes (TCPs)* are those polyominoes in which every row and every column of cells has at least one "gap." Figure 1 shows a non-TCP, while Figure 2(a) shows a similar TCP. The difference between the two polyominoes is that the former one is missing the bottom left cell. Note that it is sometimes useful to consider a polyomino just as a subgraph of the square lattice, and sometimes equivalently as a collection of square cells. Throughout the paper, our drawings will show both, when convenient. TCPs were introduced in the Handbook of Discrete and Computational Geometry [6] as an extremal opposite of *convex polyominoes*, a much more extensively studied set for which an asymptotic formula is known [7].

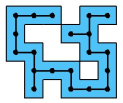


Figure 1: Not TCP.

[*]Dept. of Computer Science, Techion—Israel Inst. of Technology, Haifa 3200003, Israel, `barequet@cs.technion.ac.il`

[†]Dept. of Mathematics and Statistics, York University, Toronto, ON M3J 1P3, Canada, `madras@yorku.ca`. Supported in part by a Discovery Grant from NSERC Canada.

[‡]Dept. of Mathematics, Univ. of Waterloo, ON N2L 3G1, Canada, `j8peters@uwaterloo.ca`. Supported in part by an Undergraduate Student Research Award from NSERC Canada.

TCPs were first investigated in-depth recently in reference [2]. There,

i. The minimum possible area for a TCP was proved to be 21;

ii. The number of area-n TCPs, $\kappa(n)$, was evaluated for $21 \leq n \leq 35$; and

iii. The TCP growth constant, $\lambda_\kappa := \lim_{n\to\infty} \sqrt[n]{\kappa(n)}$ was shown to exist, bounded from below, and conjectured to be equal to $\lambda := \lim_{n\to\infty} \sqrt[n]{A(n)}$, the growth constant of all polyominoes.

In this paper, for each positive t, we consider t-fold TCPs (t-TCPs): those with at least t gaps in every row and column. Figure 2 shows a few examples which are in fact of the minimum possible sizes (see Section 3). We generalize the result on minimal examples to t-TCPs, answer the mentioned conjecture, and prove the existence of another important limit by strengthening an established technique.

The symbols A_n, κ_n, and $\kappa_{t,n}$ will denote the sets of area-n polyominoes, TCPs, and t-TCPs, respectively, while $A(n)$, $\kappa(n)$, and $\kappa_t(n)$ will denote the number of these objects. In addition, we use the following lexicographic order of cells on the square lattice.

Definition 1 *(Lexicographic Order) Given two cells on the square lattice, $c_1 = [x_1, x_1+1] \times [y_1, y_1+1]$ and $c_2 = [x_2, x_2+1] \times [y_2, y_2+1]$, we say that c_1 is lexicographically smaller than c_2 if $x_1 < x_2$, or if $x_1 = x_2$ and $y_1 < y_2$.*

2 A Physical Context

In statistical physics, it is generally believed that typical polyominoes (and, more generally, lattice trees and lattice animals, which are models of branched polymers) display a fractal geometry in the limit as their size gets large. This is supported by non-rigorous scaling theory as well as by simulations. However, almost nothing has been proven rigorously about the asymptotic geometry of these objects (except in high dimensional space, which here means above eight dimensions). Fractal behavior in two dimensions would imply in particular that a vertical or horizontal line intersecting a large polyomino would typically have gaps on all length scales. This seems hard to prove, but a simpler task would be to show that, for each positive integer t, there is a reasonable probability that most lines intersecting a sufficiently-large polyomino would have at least t gaps. Our result upgrades "most" to "every," but replaces "reasonable probability" with "probability that is not exponentially small". (In fact, it is only polynomially small under the standard belief that $A(n)$ is asymptotically proportional to $n^{-\theta}\lambda^n$ for some constant $\theta > 0$.) The probability for a given size, of course, is the ratio of t-TCPs to all polyominoes of that size. We conjecture that this probability is bounded away from 0 as the size tends to infinity. (A simple argument proves that it is bounded away from 1.) This conjecture seems to be a difficult problem to resolve rigorously, but we view our theorem as a first rigorous corroboration (albeit a mild one) of the behavior that physicists expect.

3 Minimum Area of t-TCPs

We will prove a lower bound on the area of t-TCPs for all t, and construct examples to prove it is tight, which proves the following theorem, our first main result.

Let m_t be the minimum possible area of a t-TCP.

Theorem 1 *$m_1 = 21$, $m_2 = 50$, and $m_t = 6(t+1)^2 - 1$ for all $t > 2$.* □

The theorem above is the combination of lemmata 3 and 4 below.

3.1 Lower Bound

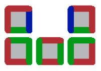
We use the notion of the *minimum bounding box* of a polyomino.

Definition 2 *The* minimum bounding box *of a polyomino P is the least pair of integers (k, ℓ), such that $P \subset [0, k] \times [0, \ell]$. That is, the minimum bounding box of P is contained in any other bounding box of P.*

The relations in Lemma 2 for the $t = 1$ case were first described in reference [2]. What follows is to our knowledge the first treatment of them in the general-t case.

Lemma 2 *For a t-fold TCP with n cells in a (k, ℓ)-bounding box,*

$$(t+1)(k+\ell) - 1 \le n \le k\ell - t \cdot \max\{k, \ell\} - 2t.$$

Proof. By rotating if necessary, we may assume $k \ge \ell$. For the lower bound, partition the edges of the polyomino's cells into *outside*, *inside*, and *hidden* edges, which we will say number o, i, and h, respectively. Outside edges face away from the polyomino, inside edges back into it, and hidden edges are those in between two cells. For example, the U-pentomino  has ten outside edges (red), two inside edges (blue), and eight hidden edges (green). Being the perimeter, $o = 2k + 2\ell$. By the t-TC property, $i \ge 2tk + 2t\ell$. By connectedness, every polyomino has a spanning tree with at least $n - 1$ edges, and so $h \ge 2n - 2$. The lower bound follows from this and the fact that we counted exactly $4n = o + i + h$ edges. For the upper bound, notice that we must remove at least tk cells from $[0, k] \times [1, \ell - 1]$ in order to have k-many t-fold concave columns, and a further t cells from the top and the bottom rows to guarantee their t-fold concavity. Finally, we take in the statement of the lemma the maximum of k and ℓ since their roles can be exchanged. □

These relations restrict the possible areas of t-TCPs in (k, ℓ) bounding boxes rather significantly. We see this by solving an integer non-linear program (NLP) in the general-t case using duality. Guenin *et al.* [8] provide a friendly reference for the techniques used.

Lemma 3 *$m_1 \ge 21$, $m_2 \ge 50$, and $m_t \ge 6(t+1)^2 - 1$ for $t > 2$.*

Proof. Assume, without loss of generality, that $k \ge \ell$. For a t-TCP to exist in a (k, ℓ) bounding box, the lower and upper bounds of Lemma 2 must both hold, *i.e.*, their difference $H(k, \ell) := k\ell - (2t+1)k - (t+1)\ell - 2t + 1$ must be non-negative. Minimizing the lower bound of Lemma 2, we therefore have the integer NLP (1). To solve (1), we will consider two auxiliary NLPs, (2) and (3).

$$\begin{aligned} \min\ & k+\ell \\ \text{s.t.}\ & H(k,\ell) \ge 0, \\ & k-\ell \ge 0, \text{ and} \\ & k,\ell \in \mathbb{Z}^+. \end{aligned} \tag{1}$$

$$\begin{aligned} \min\ & k+\ell \\ \text{s.t.}\ & H(k,\ell) \ge 0, \\ & k-\ell \ge 0, \text{ and} \\ & k,\ell \ge 0. \end{aligned} \tag{2}$$

$$\begin{aligned} \min\ & k+\ell \\ \text{s.t.}\ & H(k,\ell) \ge 0, \\ & k-\ell \ge 1, \text{ and} \\ & k,\ell \ge 0 \end{aligned} \tag{3}$$

First, we solve the NLP (2). Noting the region $\{(k, \ell) : H(k, \ell) \ge 0, k, \ell > 0\}$ is convex, we may define a linear relaxation by a gradient, the LP (4). We also write its dual, the LP (5),

$$\begin{aligned} \min\ & [1\ 1]\,[k\ \ell]^T \\ \text{s.t.}\ & \begin{bmatrix} 1 & -1 \\ 1 & -\alpha_t \end{bmatrix} \begin{bmatrix} k \\ \ell \end{bmatrix} \ge \begin{bmatrix} 0 \\ \beta_t - \alpha_t\beta_t \end{bmatrix} \\ \text{where}\ & k,\ell \ge 0 \end{aligned} \tag{4}$$

$$\begin{aligned} \max\ & [0\ (\beta_t - \alpha_t\beta_t)]\,[x\ y]^T \\ \text{s.t.}\ & \begin{bmatrix} 1 & 1 \\ -1 & -\alpha_t \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \le \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \text{where}\ & x,y \ge 0 \end{aligned} \tag{5}$$

where the number β_t is such that (β_t, β_t) is the point of intersection of the line $k = \ell$ and the hyperbola $H(k, \ell) = 0$, and α_t is the derivative of k with respect to ℓ of the hyperbola at the point $(k, \ell) = (\beta_t, \beta_t)$. Explicitly,

$$\beta_t = \frac{3}{2}t + 1 + \sqrt{\frac{9}{4}t^2 + 5t} \quad \text{and} \quad \alpha_t = \frac{t + 1 - \beta_t}{\beta_t - 2t - 1}.$$

To solve the primal-dual pair (4)-(5), notice that $(\bar{k}, \bar{\ell}) = (\beta_t, \beta_t)$ and $(\bar{x}, \bar{y}) = (\frac{1+\alpha_t}{\alpha_t - 1}, \frac{2}{1-\alpha_t})$ are feasible in (4) and (5), respectively, both with the objective value $2\beta_t$. Thus, it follows by weak duality that $(\bar{k}, \bar{\ell})$ is optimal in the LP (4). Since $(\bar{k}, \bar{\ell})$ is also feasible in the NLP (2), it is optimal there too.

We will also solve the auxiliary NLP (3), just as we solved (2). First, we find an LP relaxation of NLP (3), the LP (6), and write its dual, the LP (7),

$$\begin{aligned} &\min \quad [1\ 1]\,[k\ \ell]^T \\ &\text{s.t.} \quad \begin{bmatrix} 1 & -1 \\ 1 & -\delta_t \end{bmatrix} \begin{bmatrix} k \\ \ell \end{bmatrix} \geq \begin{bmatrix} 1 \\ \gamma_t - \delta_t\gamma_t + 1 \end{bmatrix} \\ &\text{where} \quad k, \ell \geq 0 \end{aligned} \tag{6}$$

$$\begin{aligned} &\max \quad [1\ (\gamma_t - \delta_t\gamma_t + 1)]\,[x\ y]^T \\ &\text{s.t.} \quad \begin{bmatrix} 1 & 1 \\ -1 & -\delta_t \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ &\text{where} \quad x, y \geq 0 \end{aligned} \tag{7}$$

where γ_t and δ_t are defined analogously to α_t and β_t. The number γ_t is such that $(\gamma_t + 1, \gamma_t)$ is the point of intersection between the line $k = \ell + 1$ and the hyperbola $H(k, \ell) = 0$, and δ_t is the derivative of k with respect to ℓ of the hyperbola at the point $(k, \ell) = (\gamma_t + 1, \gamma_t)$. Explicitly,

$$\gamma_t = \frac{3}{2}t + \frac{1}{2} + \sqrt{\frac{9}{4}t^2 + \frac{11}{2}t + \frac{1}{4}} \quad \text{and} \quad \delta_t = \frac{t - \gamma_t}{\gamma_t - 2t - 1}.$$

Observe that $(\bar{k}, \bar{\ell}) = (\gamma_t + 1, \gamma_t)$ and $(\bar{x}, \bar{y}) = (\frac{1+\delta_t}{\delta_t - 1}, \frac{2}{1-\delta_t})$ are feasible in (6) and (7), respectively, both with objective value $2\gamma_t + 1$. Thus, it follows by weak duality that $(\bar{k}, \bar{\ell})$ is optimal in (6). Since $(\bar{k}, \bar{\ell})$ is feasible also in the NLP (3), it is optimal there too.

We are now ready to solve the original integer NLP (1). In the $t = 1$ case, the optimal value of (1) is at least the ceiling of the minimum of the optimal values of (4) and (6), which is 11. Since $(k, \ell) = (\gamma_1 + 1, \gamma_1) = (6, 5)$ realizes this bound, it is optimal in (1). For the $t = 2$ case, $\gamma_2 = 8$ is an integer. Hence, $(k, \ell) = (9, 8)$ is an optimal integer solution to (3). Because the only integer points feasible in (2) but not in (3) are on the line $k = \ell$, the least of which is $(k, \ell) = (\lceil\beta_2\rceil, \lceil\beta_2\rceil) = (9, 9)$, we have that $(k, \ell) = (8, 9)$ is optimal in (1). For $t > 2$, we observe $3t + 3 > \beta_t, \gamma_t > 3t + 2$. Since $(k, \ell) = (\gamma_t + 1, \gamma_t)$ is optimal in (3), all feasible integers $k > \ell$ have $k + \ell \geq \lceil 2\gamma_t + 1\rceil \geq 6t + 6$. Since the least feasible integer $k = \ell$ is $\lceil\beta_t\rceil = 3t + 3$, we have that $(k, \ell) = (3t + 3, 3t + 3)$ is optimal in (1). The result follows from these solutions to (1) and Bound (1) of Lemma 2. □

We remark that one could alternatively perform the above proof by finding points satisfying the Karush-Kuhn-Tucker conditions in NLPs (2) and (3).

3.2 Upper Bound

To bound the minimum area of a t-TCP from above by $n \in \mathbb{N}$ inclusive, it suffices to find a t-TCP of area n. For $t = 1, 2$, the examples given in Figures 2(a,b) are enough. For $t > 2$, we require a general construction.

Lemma 4 *For $t \geq 3$, $m_t \leq 6(t + 1)^2 - 1$.*

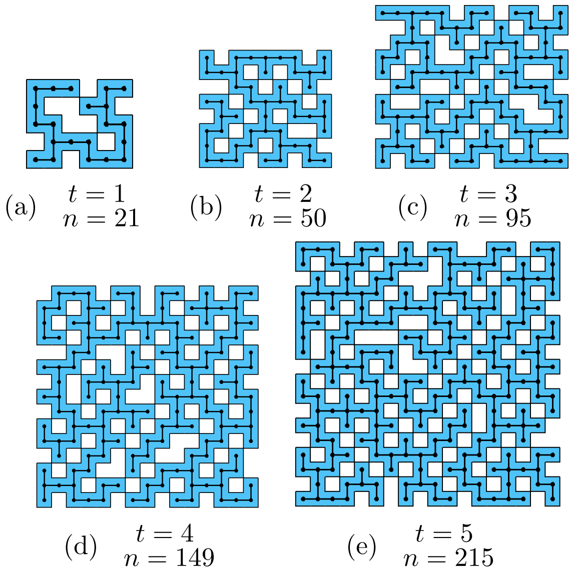


Figure 2: Minimum-area TCPs for $1 \leq t \leq 5$
.

Proof. Consider the following construction, in four steps. It is illustrated in Figure 3 in the $t = 3$ case.

1. Create a collection of cells, placing one cell about a point $(x, y) \in \{0, 1, 2, \cdots, 3t+2\} \times \{0, 1, 2, \cdots, 3t+2\}$ if and only if the sum $(x + y)$ is not congruent to 2 modulo 3. This collection has $6(t + 1)^2$ cells, and $2(t + 1)$ connected components. Each row and column has at least t gaps, and some have more.

2. There are $(t+1)$ columns (resp., rows) with $(t+1)$ gaps, at $x \equiv 2 \mod 3$ (resp., $y \equiv 2 \mod 3$). Place more cells about points of the form $(3i+1, 3i+1)$, for $0 \leq i \leq t$. The resulting collection of cells has $6(t + 1)^2 + (t + 1)$ cells, and $(t + 1)$ connected components. Each row and column retains at least t gaps.

3. Step 2 created multiply connected components. Therefore, we may remove the $2t + 2$ cells about points of the form $(3i + 1 \pm 1, 3i + 1 \pm 1)$ for $0 \leq i \leq t$ without creating more connected components. There are still at least t gaps in each column and row, since $(3i, 3i)$ is adjacent to $(3i+1, 3i)$ and $(3i, 3i + 1)$, while $(3i + 2, 3i + 2)$ is adjacent to $(3i + 2, 3i + 1)$ and $(3i + 1, 3i + 2)$. The result is a collection of $6(t + 1)^2 - (t + 1)$ cells.

4. We can connect the remaining $(t + 1)$ connected components to each other with t additional cells, centered about points of the form $(3i, 3i - 1)$ for $1 \leq i \leq t$. No such addition breaks t-fold concavity, since $(3i, 3i-1)$ is not adjacent to either $(3i+1, 3i+1)$ or $(3i, 3i)$, each removed in Step 3. We are left with a t-TCP of area $6(t + 1)^2 - 1$.

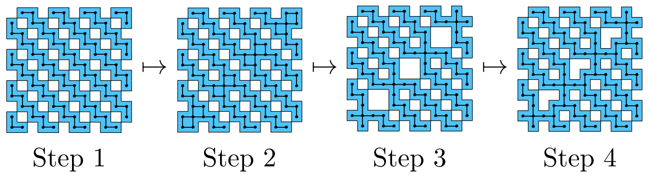


Figure 3: Constructing a t-TCP of area $6(t+1)^2 - 1$.

Since this construction works for all $t > 2$, the claim is proved. □

Note that the examples produced by the above construction are different from the ones given in Figure 2.

Corollary 5 *For $t \geq 1$, there are at least 2^t-many t-TCPs of area $6(t+1)^2 - 1$.*

Proof. In Step 4, we could have just as well placed a cell about the point $(3t, 3t+1)$ whenever we placed one about $(3t+1, 3t)$, giving t binary choices. □

3.3 Structure of Minimum-Area TCPs

Figure 2 shows some minimum-area t-TCPs for $1 \leq t \leq 5$. Lemma 6 below characterizes the bounding boxes of minimal t-TCPs. The existence of non-square minimal t-TCP for $t > 3$ is an open question. The construction in the proof of Theorem 4 shows that there exists a minimal t-TCP in a $(3t+3, 3t+3)$ bounding box for all $t > 2$. However, the 3-TCP with a (13,11) bounding box shown in Figure 2(c) is currently the only example of a minimal t-TCP that does not have a square bounding box for $t > 2$. The following results also relate the dimension of a t-TCPs bounding box to its connectivity and its concavity.

Lemma 6 *If a t-TC polyomino has a (k, ℓ) bounding box and its area is $(t+1)(k+\ell) - 1$, then*
(i) it is a tree; and
(ii) it has exactly t gaps in every row and every column.

Proof. We prove the contrapositive. If P is an area-n t-TCP that is not a tree, the bound for the number of hidden edges (see the proof of Lemma 2) becomes $h \geq 2n$. If P has more than t gaps in some row or column, the bound on the number of inside edges becomes $i \geq 2t(k+\ell) + 2$. In either case, we get $n \geq (t+1)(k+\ell)$ given that $i + o + h = 4n$, hence $n \neq (t+1)(k+\ell) - 1$. The claim follows. □

We now present our second main result.

Theorem 7 *Suppose that P is a minimum-area t-TCP of area n whose bounding box is $B = (k, \ell)$ (for $k \geq \ell$). If $t = 1$, then $B = (6, 5)$. If $t = 2$, then $B = (9, 8)$. Otherwise, if $t \geq 3$, then B is either $(3t+3, 3t+3)$ or $(3t+4, 3t+2)$. Moreover, P is a tree and it has exactly t gaps in every row and every column.*

Proof. Let P be a t-TCP with area m_t in a (k, ℓ) bounding box. We claim that

$$(t+1)(k+\ell) - 1 = m_t. \tag{8}$$

By Lemma 2, we have that $m_t \geq (t+1)(k+\ell) - 1$. The pair (k, ℓ) is feasible in the integer NLP (1) because P is a t-TCP. Since m_t is equal to the lower bound given by Lemma 3, m_t is the minimum of $(t+1)(k+\ell) - 1$ for feasible (k, ℓ) pairs, that is, $m_t \leq (t+1)(k+\ell) - 1$. Hence, Equation (8) holds, and by Lemma 6 we have that all minimal t-TCPs are trees and have exactly t gaps everywhere.

It is easy to check that the unique solutions that satisfy Equation (8) and the relations in Lemma 2 are $(k, \ell) = (6, 5)$ in the $t = 1$ case and $(k, \ell) = (9, 8)$ in the $t = 2$ case. A manual inspection of all 1-TCPs (provided in reference [2]) is also available for $t = 1$.

For $t \geq 3$, notice that the solutions to Equation (8) with $k \geq \ell$ take the form $(k, \ell) = (3t+3+\Delta, 3t+3-\Delta)$ for some non-negative integer $\Delta \geq 0$. Expanding and rearranging the relations in Lemma 2 in this case give $t(1-\Delta) + 4 - \Delta^2 \geq 0$, which is possible only if $\Delta = 0$ or 1. □

4 Equality of the t-TCP Growth Constants to λ

It is straightforward to prove the existence of the t-TCP growth constants via a concatenation argument and supermultiplicativity, as was done previously for 1-TCPs [2] and is common for other families of lattice animals [1, 5, 10].

Theorem 8 *For all $t > 0$, $\lambda_{\kappa_t} := \lim_{n\to\infty} \sqrt[n]{\kappa_t(n)}$ exists. Moreover, $\lambda_{\kappa_t} := \sup_n \sqrt[n]{\kappa_t(n)}$.*

Proof. Since every t-TCP is a polyomino, $\sqrt[n]{\kappa_t(n)} \leq \sqrt[n]{A(n)}$. Additionally, $\sqrt[n]{A(n)} \to \lambda$ as $n \to \infty$. We conclude that the sequence $\{\sqrt[n]{\kappa_t(n)}\}_{n>0}$ is bounded. We will also show that for all $t, n, m > 0$, $\kappa_t(n) \cdot \kappa_t(m) \leq \kappa_t(n+m)$, *i.e.*, $\{\kappa_t(n)\}$ is supermultiplicative. The result then follows from Lemma 1 of Ref. [10] which states precisely the existence of the hypothesized limits for supermultiplicative sequences that are bounded as above.

To see the supermultiplicative relation, we concatenate two t-TCPs of sizes n and m. Given $P_1 \in \kappa_{t,n}$ and $P_2 \in \kappa_{t,m}$, let P_3 be the union of P_1 and the translation of P_2 such that its lexicographically smallest cell lies immediately to the right of P_1's greatest cell. Then, P_3 uniquely determines an element of $\kappa_{t,n+m}$ since all rows and columns still have at least t gaps, and the original pair P_1, P_2 can be determined uniquely from P_3 by separating the n lexicographically-smallest from the m lexicographically-biggest cells of P_3.

To prove $\lambda_{\kappa_t} := \sup_n \sqrt[n]{\kappa_t(n)}$, observe that for each n, the supermultiplicative relation implies that the subsequence $\{\sqrt[n2^m]{\kappa_t(n2^m)} : m \geq 0\}$ is increasing. Indeed, its limit must be λ_{κ_t}, so $\sqrt[n]{\kappa_t(n)} \leq \lambda_{\kappa_t}$ for all $n > 0$ $\square$

Theorem 8 can be used for obtaining lower bounds on λ_{κ_t}. If $\kappa_t(n) \geq x$, then $\lambda_{\kappa_t} \geq \sqrt[n]{x}$. That is how the best known lower bound on λ_{κ_1}, which is 2.4474, was found (see reference [2]). However, the following construction does better. Compare the previous known bound to the one given in Corollary 10 below.

Theorem 9 *The growth constant for t-TCPs, λ_{κ_t}, equals λ for all $t > 0$.*

Proof. We partition the set of all n-ominoes into a polynomial number of subsets. Given any polyomino P, we define the following quantities.

$$\begin{aligned}
X_{span}(P) &= \max\{x : (x,y) \in P \text{ for some } y\},\\
Y_{span}(P) &= \max\{y : (x,y) \in P \text{ for some } x\},\\
X_0^-(P) &= \min\{x : (x,0) \in P\},\\
X_0^+(P) &= \min\{x : (x, Y_{span}(P)) \in P\},\\
Y_0^-(P) &= \min\{y : (0,y) \in P\},\\
Y_0^+(P) &= \min\{y : (X_{span}(P), y) \in P\}.
\end{aligned}$$

Note that if the span of a polyomino P in either of the axes is d, then the coordinates of cells of P along that axis are in the range $[0, d-1]$. Then, the set $P_n[a,b,c,d,e,f]$ is defined as

$$\begin{aligned}
P_n[a,b,c,d,e,f] = \{P \in A_n :\ & X_{span}(P) = a,\\
& Y_{span}(P) = b, X_0^-(P) = c, X_0^+(P) = d,\\
& Y_0^-(P) = e, Y_0^+(P) = f\}.
\end{aligned}$$

See Figure 4 for an illustration of a typical member of $P_n[a,b,c,d,e,f]$. It follows from the connectedness of P that $P_n[a,b,c,d,e,f] = \emptyset$ if any of a,b,c,d,e,f are greater than n, hence,

$$A_n = \bigcup_{\substack{0<a,b\leq n\\ 0\leq c,d,e,f<n}} P_n[a,b,c,d,e,f]. \tag{9}$$

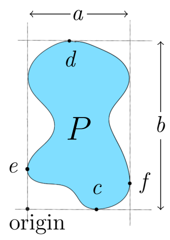


Figure 4: A typical member of $P_n[a,b,c,d,e,f]$.

Therefore, there exist $a^\circ, b^\circ, c^\circ, d^\circ, e^\circ, f^\circ$ for which $|P_n[a^\circ, b^\circ, c^\circ, d^\circ, e^\circ, f^\circ]| \geq \left(\frac{A(n)}{n^6}\right)$. For each $i,j \in \{0,1,\ldots,t\}$, let $\pi_{i,j}$ be any element of $P_n[a^\circ, b^\circ, c^\circ, d^\circ, e^\circ, f^\circ]$. There are at least $\left(\frac{A(n)}{n^6}\right)^{(t+1)^2}$ choices for $\{\pi_{i,j}\}_{0\leq i,j\leq t}$.

We now construct to each choice of $\{\pi_{i,j}\}$ a unique t-TCP. First we define the polyomino B ( ), and its 90° clockwise-rotated version, $B^{\circlearrowright}$. We use the notation P_G as defined in the introduction. We set $B := P_G$, where $G := \{(0,0),(1,0),(1,1),(2,1),(3,1),(3,0),(4,0)\}$, and $B^{\circlearrowright} := P_{G^{\circlearrowright}}$, where $G^{\circlearrowright} := \{(0,0),(0,1),(1,1),(1,2),(1,3),(0,3),(0,4)\}$.

Let $P + \vec{v}$ denote the translation of a polyomino P by a vector $\vec{v} \in \mathbb{Z}^2$. We are now ready to define our constructed t-TC polyomino, $\mathrm{TCP}(\{\pi_{i,j}\}_{0\leq i,j\leq t})$. Define $\{\pi'_{i,j}\}_{0\leq i,j\leq t}$, $\{\hat{\pi}_{i,j}\}_{0\leq i,j\leq t}$, $\{B_{i,j}\}_{0\leq i<t,\,0\leq j\leq t}$, and $\{B^{\circlearrowright}_{i,j}\}_{0\leq i\leq t,\,0\leq j<t}$, by the following rules.

$$\pi'_{i,j} = \begin{cases} \pi_{i,j} & i,j \text{ even}\\ \text{reflection of } \pi_{i,j} \text{ through the line } y = b^\circ/2 & i \text{ even, } j \text{ odd}\\ \text{reflection of } \pi_{i,j} \text{ through the line } x = a^\circ/2 & i \text{ odd, } j \text{ even}\\ \text{reflection of } \pi_{i,j} \text{ through } y = \frac{b^\circ}{2} \text{ and } x = \frac{a^\circ}{2} & i,j \text{ odd}\end{cases}$$

$$\hat{\pi}_{i,j} = \pi'_{i,j} + (i(a^\circ+5), j(b^\circ+5))$$

$$B_{i,j} = \begin{cases} B + (i(a^\circ+5)+a^\circ,\ j(b^\circ+5)+f^\circ) & i,j \text{ even}\\ B + (i(a^\circ+5)+a^\circ,\ j(b^\circ+5)+b^\circ-f^\circ-1) & i \text{ even, } j \text{ odd}\\ B + (i(a^\circ+5)+a^\circ,\ j(b^\circ+5)+e^\circ) & i \text{ odd, } j \text{ even}\\ B + (i(a^\circ+5)+a^\circ,\ j(b^\circ+5)+b^\circ-e^\circ-1) & i,j \text{ odd}\end{cases}$$

$$B^{\circlearrowright}_{i,j} = \begin{cases} B^{\circlearrowright} + (i(a^\circ+5)+d^\circ,\ j(b^\circ+5)+b^\circ) & i,j \text{ even}\\ B^{\circlearrowright} + (i(a^\circ+5)+c^\circ,\ j(b^\circ+5)+b^\circ) & i \text{ even, } j \text{ odd}\\ B^{\circlearrowright} + (i(a^\circ+5)+a^\circ-d^\circ-1,\ j(b^\circ+5)+b^\circ) & i \text{ odd, } j \text{ even}\\ B^{\circlearrowright} + (i(a^\circ+5)+a^\circ-c^\circ-1,\ j(b^\circ+5)+b^\circ) & i,j \text{ odd}\end{cases}$$

Thus, $B_{i,j}$ intersects each of $\hat{\pi}_{i,j}$ and $\hat{\pi}_{i+1,j}$ in one edge, and $B^{\circlearrowright}_{i,j}$ intersects each of $\hat{\pi}_{i,j}$ and $\hat{\pi}_{i,j+1}$ in one edge. Finally, define

$$TCP(\{\pi_{i,j}\}_{0\leq i,j\leq t}) = \left(\bigcup_{0\leq i,j\leq t} \hat{\pi}_{i,j}\right) \cup \left(\bigcup_{i=0}^{t-1}\bigcup_{j=0}^{t} B_{i,j}\right) \cup \left(\bigcup_{i=0}^{t}\bigcup_{j=0}^{t-1} B^{\circlearrowright}_{i,j}\right).$$

See Figure 5 for an illustration of this construction. It is evident that $\mathrm{TCP}(\{\pi_{i,j}\})$ uniquely determines $\{\pi_{i,j}\}$. It is easily verified by inspecting rows and columns that $\mathrm{TCP}(\{\pi_{i,j}\})$ is indeed t-TC, although we must be careful with the following special case. If e° or f° is $b^\circ-1$ or 0, then we must redefine $B_{i,t}$ for $0 \leq i < t$, replacing B ( ) in the above with its vertical reflection ( ) in the line $y = \frac{1}{2}$. Without this hack, the top row of $\mathrm{TCP}(\{\pi_{i,j}\})$ may not be t-TC, consisting only of the $B_{i,t}$ kinks. The same applies to $B^{\circlearrowright}_{t,j}$ for the case in which c° or d° is $a^\circ-1$ or 0, where we must replace $B^{\circlearrowright}$ with its horizontal reflection in the line $x = \frac{1}{2}$.

The constructed t-TC polyomino $\mathrm{TCP}(\{\pi_{i,j}\})$ has $\varphi := (t+1)^2 n + 14t(t+1)$ cells. The $14t(t+1)$ term comes from the 7 cells in each of $B_{i,j}$ and $B^{\circlearrowright}_{i,j}$. Therefore, we have that $\kappa_t(\varphi) \geq \left(\frac{A(n)}{n^6}\right)^{(t+1)^2}$, and hence we have that $\sqrt[\varphi]{\kappa_t(\varphi)} \geq \sqrt[\varphi]{\left(\frac{A(n)}{n^6}\right)^{(t+1)^2}}$.

We now let $n \to \infty$. Note that the indices φ define a subsequence of the sequence $\kappa_t(n)$. Since the sequence $\sqrt[n]{\kappa_t(n)}$ converges, so does the subsequence, and to the same limit. The right side of the final inequality shows

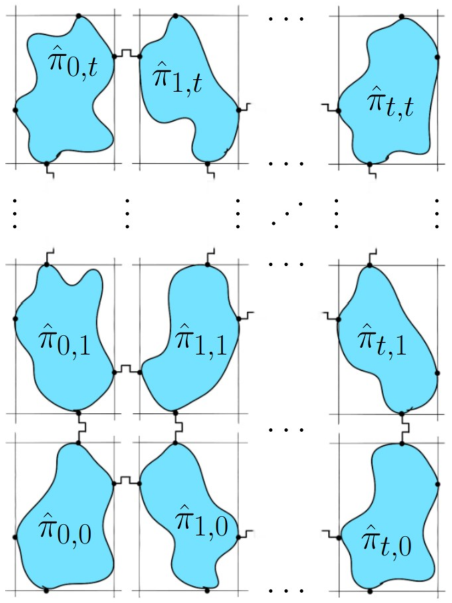


Figure 5: A "blob" representation of the polyomino TCP$(\{\hat{\pi}_{i,j}\}_{0\le i,j\le t})$. The blobs represent $\hat{\pi}_{i,j}$s, and the "squiggles" between them the $B_{i,j}$s and $B^{\circlearrowleft}_{i,j}$s. Notice how the $\hat{\pi}_{i,j}$s are flipped throughout the construction to match along their boundaries.

that the limit is at least λ. However, the limit cannot exceed λ, because t-TCPs are a proper subset of all polyominoes. Therefore, it must be that $\lambda_{\kappa_t} = \lambda$. □

Corollary 10 *For all $t > 0$, we have that $4.0025 \le \lambda_{\kappa_t} \le 4.5252$.*

Proof. These are just the best known lower [3] and upper [4] bounds on λ. □

We have just proved that t-TCPs are not exponentially rare in the polyominoes. That begs the question; how common are t-TCPs? The next theorems show that they are not overwhelmingly so: The proportion of polyominoes that are t-TCPs is bounded away from 1.

First, we need a ratio-limit theorem.

Theorem 11 *(Ratio Limit Theorem for t-TCPs)* $\lim\limits_{n\to\infty} \frac{\kappa_t(n+1)}{\kappa_t(n)}$ *exists and equals* λ_{κ_t}. □

The proof is given in the full version of the paper.

Theorem 12 $\limsup_{n\to\infty} \frac{\kappa_t(n)}{A(n)} \le \frac{\lambda}{\lambda+4(t+1)}\ \forall t > 0$.

Proof. We map every t-TCP of size $n-1$ to all elements of A_n obtained by attaching one cell immediately to the right (resp., left) of any cell in its rightmost (resp., leftmost) column, or above (resp., below) any cell in its topmost (resp., bottommost) row. This way, every polyomino in $\kappa_{t,n-1}$ is mapped to at least $4(t+1)$ polyominoes in A_n, all images are distinct, and all images are not t-TCPs. Hence, we have that $A(n) \ge \kappa_t(n) + 4(t+1)\kappa_t(n-1)$, that is, $A(n)/\kappa_t(n) \ge 1 + 4(t+1)\kappa_t(n-1)/\kappa_t(n)$. Therefore, by the Ratio Limit Theorem for t-TCPs (Theorem 11),

$$\limsup \frac{\kappa_t(n)}{A(n)} \le \lim_{n\to\infty} \frac{1}{1+\frac{4(t+1)\kappa_t(n-1)}{\kappa_t(n)}} = \frac{1}{1+\frac{4(t+1)}{\lambda}} = \frac{\lambda}{\lambda+4(t+1)}.$$

□

Note that this does *not* imply that $\kappa_t(n)/A(n)$ (t fixed) converges to a positive value (as a function of n), or converges at all, when $n \to \infty$. For example, if $A(n) \sim c_0 n^{\theta_0}\lambda^n$ (which is widely believed) and $\kappa_t(n) \sim c_t n^{\theta_t}\lambda^n$, where $\theta_t \le \theta_0$, then $\lim_{n\to\infty} \kappa_t(n)/A(n)$ would be 0 if $\theta_t < \theta_0$ but nonzero if $\theta_t = \theta_0$. However, since $\lim_{t\to\infty} \lambda/(\lambda+4(t+1)) = 0$, we conclude that the limiting fraction of t-TCPs out of all polyominoes vanishes as t tends to ∞.

5 Conclusion

In this paper, we make a natural generalization of TCPs to t-TCPs, generalized previous results on TCPs to t-TCPs, and proved new results on TCPs in the more general t-TCP case. To answer the minimum-area problem for t-TCPs, we find that m_t grows quadratically with t (Theorem 1).

We also prove that for a fixed t, t-TCPs are not exponentially rare in the regular (those not necessarily TC) polyominoes (Theorem 9). This begs the question, exactly how common are t-TCPs in the regular polyominoes? From the bound $\kappa_t(\varphi) \ge (A(n)/n^6)^{(t+1)^2}$ in the proof of Theorem 9, we find that the widely believed relation $c_0 n^{\theta_0}\lambda^n \le A(n)$ (for some constants c_0, θ_0), together with the known relation $A(n) \le \lambda^n$, would imply that $\kappa_t(n)/A(n) = \Omega\left(n^{(\theta_0-6)(t+1)^2}\right)$. We also have an asymptotic upper bound on $\kappa_t(n)/A(n)$ in Theorem 12. Our question in this regard is: For a fixed t, do t-TCPs form an asymptotically positive fraction of all polyominoes? Looking at the known values of $\kappa_t(n)/A(n)$, it seems plausible that it is indeed a positive fraction, but the data are insufficient to make confident claims. In the case that rigorous proof eludes, we may still gain insight by measuring the prevalence of t-TCPs in Monte Carlo samples of polyominoes, along the lines of reference [9].

We also notice that the definitions of *minimality* (not minimum area), *primitivity*, and *saturation* given in reference [2] have clear t-fold generalizations, and questions 3–6 therein can be posed in the t-fold case too.

References

[1] G. Barequet, G. Ben-Shachar, and M.C. Osegueda, Concatenation arguments and their applications to polyominoes and polycubes, *Computational Geometry: Theory and Applications*, 98 (2021), 12 pp.

[2] G. Barequet, N. Keren, N. Madras, J. Peters, and A. Rivkin, On totally-concave polyominoes, *Proc. 36th Canadian Conf. on Computational Geometry*, St. Catharines, Ontario, Canada, 17–24, July 2024.

[3] G. Barequet, G. Rote, and M. Shalah, $\lambda > 4$: An improved lower bound on the growth constant of polyominoes, *Comm. of the ACM*, 59 (2016), 88–95.

[4] G. Barequet and M. Shalah, Improved upper bounds on the growth constants of polyominoes and polycubes, *Algorithmica*, 84 (2022), 3559–3586.

[5] G. Barequet, M. Shalah, and Y. Zheng, An improved lower bound on the growth constant of polyiamonds, *J. of Combinatorial Optimization*, 37 (2019), 424–438.

[6] G. Barequet, S.W. Solomon, and D.A. Klarner, Polyominoes, *Handbook of Discrete and Computational Geometry*, 3rd ed. (E. Goodman, J. O'Rourke, and C.D. Tóth, eds.), 359–380. Chapman and Hall/CRC Press, 2017.

[7] E.A. Bender, Convex n-ominoes, *Discrete Mathematics*, 8 (1974), 219–226.

[8] B. Guenin, J. Könemann, and L. Tunçel, *A Gentle Introduction to Optimization*, Cambridge University Press, 2014.

[9] E.J. Janse van Rensburg and N. Madras, Metropolis Monte Carlo simulation of lattice animals, *J. Phys. A: Math. Gen.*, 30 (1997), 8035–8066.

[10] D.A. Klarner, Cell growth problems, *Canadian J. of Mathematics*, 19 (1967), 851–863.

[11] N. Madras, A pattern theorem for lattice clusters, *Annals of Combinatorics*, 3 (1999), 357–384.

Computational Geometry from a Non-Euclidean Perspective

Godfried Toussaint Memorial Lecture

David Mount[*]

The field of discrete and computational geometry has been immensely successful in enhancing the understanding of efficient algorithms and data structures for problems that involve geometric inputs. The vast majority of work in this field has focused on Euclidean geometry and its close relatives in normed spaces, such as L_1 and L_∞. Recently, there has been growing interest in non-Euclidean geometries, including hyperbolic geometry, Hilbert geometry, and alternative notions of distance, such as Bregman divergences. In this talk, we will explore the reasons behind the increased interest in these geometries, survey recent developments, and demystify the somewhat arcane mathematics that underlies these systems. Finally, we will present many open problems that are inspired by this non-Euclidean viewpoint.

[*]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, mount@umd.edu

Guarding Polygons With Mutually Visible π-Guards[*]

Arash Ahadi[†] Ahmad Biniaz[‡] Mohammad Hashemi[§] Ali Nakhaeisharif[¶]

Abstract

We study a version of the polygon guarding problem in which we want to guard the polygon with a minimum number of vertex guards with 180° field of view such that for each guard $\mathbf{g}$ there is a guard $\mathbf{g}'$ where $\mathbf{g}$ and $\mathbf{g}'$ are mutually visible. Let $g(n)$ be the minimum number of such guards over all polygons of size n. We show that $\frac{2n-2}{3} \leqslant g(n) \leqslant \frac{4n}{5}$. We define $\bar{g}(n)$ analogously for orthogonal polygons, and show that $\frac{3n-4}{7} \leqslant \bar{g}(n) \leqslant \frac{n}{2}$. The lower bounds are existential in the sense that there are polygons that need these many guards.

1 Introduction

The classic art gallery problem, posed by Victor Klee in 1973 [8], asks for "*the minimum number of guards required to monitor every point within an art gallery with n walls*". In this problem, each guard is considered to be a point g in the polygon (including boundary) that can see any point p, where the segment gp lies inside the polygon. In 1975, Chvátal [5] showed that $\lfloor \frac{n}{3} \rfloor$ guards are always sufficient and sometimes necessary to cover a polygon with n vertices. Chvátal's proof is by induction on a triangulation graph of the polygon. In 1978, Fisk [6] provided a shorter proof using a 3-coloring of the triangulation graph. The initial appearance of other variations of the art gallery problem followed these foundational results, including scenarios with mobile guards [14], guards with limited visibility [3, 18, 19] or mobility [1, 4], and guarding orthogonal polygons [9, 13]. For more related results, we refer the interested readers to the book by O'rourke [15].

One of the variations of the art gallery problem is the concept of *cooperative guards* [10], which is also known as *connected guards* [17]. The goal of this problem is to find the minimum number of guards such that their visibility graph is connected. In another variation [11, 17, 20], known as *guarded guards* (*weakly cooperative guards*, or *watched guards*), the goal is to find the minimum number of guards such that each guard is visible from some other guard. For example, in Figure 1(a) three guards are sufficient for guarding the polygon, while in Figure 1(b), the guards must guard each other mutually, resulting in a total number of 4 guards, each of which is being guarded by at least one other guard. The main motivation of these variations is that if something goes wrong with one guard, at least one other guard can be notified.

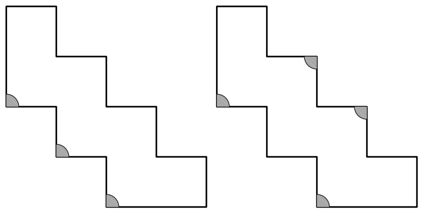


Figure 1: A difference between (a) classic guards, and (b) guarded guards.

The guarded guards problem was introduced by Liaw et. al. [11] in 1994, where they presented an optimal linear-time algorithm for 1-spiral polygons.[1] In 1995, Hernandez [16] proved an upper bound of $\lfloor \frac{n}{3} \rfloor$ for orthogonal polygons. In 2002, Zylinski [20, 21] provided tight bounds of $\lfloor \frac{2n}{5} \rfloor$ for monotone and spiral polygons, and $\lfloor \frac{3n-1}{7} \rfloor$ for star-shaped polygons. In the same year, independently, Michael and Pinciu [17] showed that $\lfloor \frac{3n-1}{7} \rfloor$ guards for simple polygons are always sufficient and sometimes necessary. They also provided a shorter proof on the upper bound of orthogonal polygons. Regarding computational complexity, Liaw et. al. proved that the guarded guards problem is $\mathcal{NP}$-hard [11].

In the above problems the guards are assumed to have 360° field of view. In recent years, there has been an increased interest in guards with bounded field of view that only see an angle $\alpha < 360°$ [3, 18, 19]. For 360° field of view, if a guard g_1 sees g_2, then g_2 also sees g_1, which means that they are *mutually visible*. However, this property does not hold for $\alpha < 360°$. Florentino et al. [7] studied a version of the connected guard problem with $\alpha = 180°$, where the mutual visibility graph must be connected.

[*]This research is supported in part by NSERC.

[†]The work is done while the author was a research associate in the Department of Computer Science at University of Windsor, aarash.ahadi.academic@gmail.com

[‡]Department of Computer Science, University of Windsor, abiniaz@uwindsor.ca

[§]The work is done while the author was a research assistant in the Department of Computer Science at University of Windsor, hashem62@uwindsor.ca

[¶]Department of Computer Science, University of Windsor, nakhaeia@uwindsor.ca

[1]A polygon is 1-spiral if its boundary has exactly one chain of reflex vertices.

1.1 Preliminaries

A *triangulation* of a polygon P refers to the division of P into a set of triangles, which are non-overlapping and cover the entire area of the polygon. The corners of triangles are at vertices of P. The *dual graph* of a triangulation is constructed by defining a vertex for each triangle in the triangulation; edges are drawn between vertices if the corresponding triangles share a common edge [15]. The dual of triangulation is a tree with vertices of degree ≤ 3.

An *orthogonal polygon* is defined as a polygon whose edges meet at right angles, resulting in edges that are either horizontal or vertical. An orthogonal polygon admits a *convex quadrangulation* which is a partitioning of the polygon into convex quadrilaterals [9]. The *dual of convex quadrangulation* is derived analogously to the dual of triangulation. The dual of quadrangulation is a tree with exactly $\frac{n-2}{2}$ vertices of degree $\leqslant 4$ [15].

1.2 Our Contributions

We study the guarded guards problem with guards of 180° field of view. In this problem, we are given a simple polygon P with n vertices. The goal is to place the minimum number of guards with 180° field of view at vertices of P to cover the entire polygon P, such that for each guard g, there is a guard g' such that g and g' see each other, in other words, g and g' are mutually visible. In this problem:

1. The guards must be placed at the vertices.
2. Each guard has 180° field of view.
3. Each guard must be mutually visible by some other guard (Figure 2).
4. The guards cannot be placed outwards (i.e. towards the exterior of the polygon as in Figure 11).
5. If two guards are placed at the same vertex, then they cannot be mutually visible to each other.

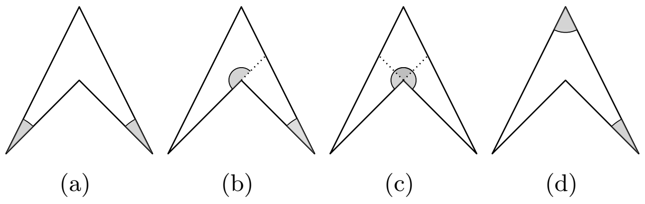


Figure 2: In (a), (b), (c) the guards cover the polygon but do not see each other mutually. (d) is a valid guarded guarding.

The assumption that, two guards placed at the same vertex are not considered visible to each other, is to avoid collusion among the guards.

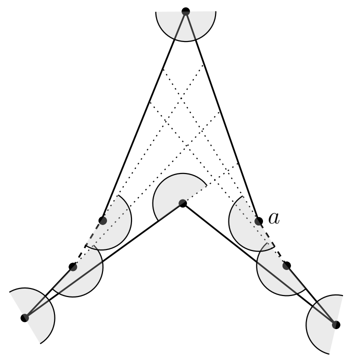


Figure 3: The necessity of two guards on some vertex.

Table 1: Bounds for different variants of the problem.

Polygon	Field of View	Bound
Simple	360°	$\lfloor \frac{3n-1}{7} \rfloor$ [12,20]
Orthogonal	360°	$\lfloor \frac{n}{3} \rfloor$ [12,16]
Simple	180°	$[\frac{2n-2}{3}, \frac{4n}{5}]$ (Thm. 2)
Orthogonal	180°	$[\frac{3n}{7}, \frac{n}{2}]$ (Thm. 9)

We define $g(n)$ to be the minimum number such that any polygon with n vertices can be guarded by at most $g(n)$ such guards. We define $\bar{g}(n)$ analogously for orthogonal polygons, where the edges are axis-aligned.

Placement of two guards at one vertex is sometimes necessary. Figure 3 shows an arbitrary large polygon, such that if we place one guard at each vertex there is still a guard that is not mutually visible by any other guard (vertex a in the figure).

In Section 2 we study simple polygons and prove that $\frac{2n-2}{3} \leqslant g(n) \leqslant \frac{4n}{5}$. In Section 3 we study the orthogonal polygons and prove that $\frac{3n-4}{7} \leqslant \bar{g}(n) \leqslant \frac{n}{2}$. Table 1 summarizes existing bounds for angle 360° and our bounds for angle 180°. In the rest of the paper the term "guard" refers to a guard of 180° field of view.

2 Simple Polygons

In this Section we prove that any n-vertex polygon can be mutually guarded by at most $\frac{4n}{5}$ guards. Moreover, there are n-vertex polygons that require $\frac{2n-2}{3}$ guards. Therefore, $\frac{2n-2}{3} \leqslant g(n) \leqslant \frac{4n}{5}$. In section 2.1 we present a lower bound example, and in section 2.2 we prove the upper bound. Our proof is by induction. It cuts a small portion of the polygon, guard it with a few mutually visible guards, and apply induction on the rest of the polygon. This is a standard technique used in many visibility guarding problems [14,17].

2.1 The Lower Bound

For every $m \geqslant 1$ we construct a polygon P with $n = 3m+1$ vertices that requires $2m$ guards, which implies the lower bound. Our polygon is illustrated in Figure 4. For any $i \leqslant m$, we identify a part in P that consists of vertices a_i, b_i, c_i, and a_{i+1}. In each part i, we consider two points p_i and q_i such that they are only visible from the vertices of its corresponding part. Placing one guard g_i on either one of a_i, b_i, or a_{i+1} is not sufficient for guarding both p_i and q_i; if g_i is placed on c_i, then it covers part i, but it is not covered by any other guard. Thus, we need two guards in part i to mutually guard p_i and q_i. Since there are m such parts in P, and each part requires two guards, polygon P requires a total amount of $2m = \frac{2(n-1)}{3}$ guards. Therefore, $g(n) \geqslant \frac{2n-2}{3}$.

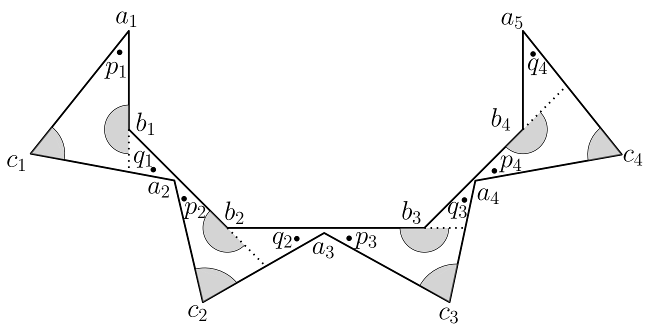


Figure 4: A polygon that requires at least $\frac{2n-2}{3}$ guards.

2.2 The Upper Bound

We prove that any n-vertex simple polygon P can be mutually guarded by at most $\frac{4n}{5}$ guards. Hence, $g(n) \leqslant \frac{4n}{5}$. Our proof is by induction on the number of vertices of P. We cut a small part of P and guard it, and recur on the remaining part of the polygon. We continue the induction until the remaining polygon is one of our base cases ($n \leqslant 7$).

The following lemma is implied from a result of [2]:

Lemma 1 *Let P be a polygon with $n \geqslant 8$ vertices. There exists a diagonal d that divides P into two polygons P' and P'' with m and $n-m+2$ vertices, respectively, for some $m \in \{5,6,7\}$.*

Observation 1 *Let P be a polygon with $n \geqslant 8$ vertices. If P can be partitioned into two polygons P' and P'' by a diagonal, with $n' \geqslant 3$ and $n'' \geqslant 3$ vertices, respectively, then $g(n) \leqslant g(n') + g(n'')$.*

Theorem 2 *For every $n \geqslant 3$:*

$$g(n) = \begin{cases} 2 & \text{if } n = 3,4,5 \\ 3 & \text{if } n = 6 \\ 4 & \text{if } n = 7 \\ \frac{4n}{5} & \text{if } n \geqslant 8. \end{cases}$$

Proof. The proof is by induction on n. The cases were $3 \leqslant n \leqslant 7$ serve as base cases, and are proved later in Section 2.2.1.

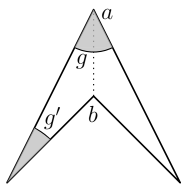


Figure 5: Guarding of P_4.

We proceed the induction step for $n \geqslant 8$. By Lemma 1, there exists a diagonal that partitions P into two polygons P' and P'' of sizes m and $n-m+2$, respectively, such that $m \in \{5,6,7\}$. By Observation 1, $g(n) \leqslant g(n-m+2) + g(m)$. Substituting the possible values of m into the formula confirms the induction step, as shown below:

$$g(n) \le g(n-3) + g(5) \le \frac{4(n-3)}{5} + 2 \le \frac{4n}{5},$$
$$g(n) \le g(n-4) + g(6) \le \frac{4(n-4)}{5} + 3 \le \frac{4n}{5},$$
$$g(n) \le g(n-5) + g(7) \le \frac{4(n-5)}{5} + 4 \le \frac{4n}{5}.$$

□

2.2.1 Base Cases

The base cases happen when $n \leqslant 8$. For $n = 3$, we need exactly two mutual guards to guard the entire polygon. As every polygon with $n = 3$ vertices is convex, one guard is sufficient to guard the entire area of the polygon. We require another guard to guard the first guard, resulting in a total of 2 guards. In Lemmas 3, 4, 5, and 6 we show the base cases where $n = 4$, $n = 5$, $n = 6$, and $n = 7$, respectively. Note that in the proofs, the term P_n presents a polygon with n vertices.

Since the sum of a quadrilateral's internal angles is 360°, the following observation holds:

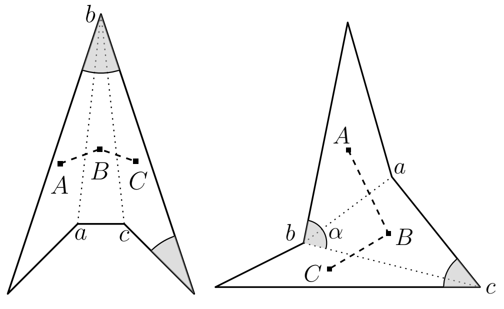
Observation 2 *Let Q be a quadrilateral. Among each pair of opposite angles in Q, at least one is convex.*

Lemma 3 $g(4) = 2$.

Proof. Let P_4 be a polygon of size 4, triangulated by adding a diagonal ab, as shown in Figure 5. By Observation 2, at least one of the vertices a or b is convex. Putting a guard g on the convex vertex is sufficient to guard P_4, as it covers both of the triangular faces. To ensure mutual visibility, we put another guard g' on a vertex that is visible from g. Therefore, $g(4) = 2$. □

Let X and Y be two interior disjoint triangles in the plane that share a side. We refer to the union of X and Y, which is a quadrilateral, by XY. Additionally, the shared side of X and Y is a diagonal in XY. In what follows, we refer to this as 'the diagonal' of XY.

Lemma 4 $g(5) = 2$.



(a) b is convex (b) b is reflex

Figure 6: Two cases for guarding P_5.

Proof. Let P_5 be a polygon of size 5. P_5 can have at most two reflex vertices, as the sum of the internal angles cannot exceed 540°. We triangulate P_5 by adding two non-intersecting diagonals and observe that its dual tree is a path of three nodes. Let B be the triangle corresponding to the middle node of the path, while triangles A and C correspond to its adjacent nodes, as shown in Figure 6. Let a, b, and c be the vertices of triangle B, such that b is shared between the two diagonals, a is shared between A and B, and c is shared between B and C. We have two cases:

- b is a convex vertex of P_5: We place a guard on b to cover the entire polygon, and another guard on another vertex to mutually see b, as in Figure 6(a).
- b is a reflex vertex of P_5: Then, one of a and c is convex. Without loss of generality, let c be the convex vertex. We place a guard on c. Let α be the portion of the angle of vertex b that lies in quadrilateral AB, as presented in Figure 6(b). By Observation 2, either α or the angle at vertex a is convex. We put one guard to cover this convex angle. This guard and c are mutually visible. □

A proof of the following lemma is given in the full version of the paper.

Lemma 5 *$g(6) = 3$.*

Lemma 6 *$g(7) = 4$.*

Proof. This statement is verified as there is a diagonal in every 7-gon that splits the polygon into a pentagon and a quadrilateral. By Lemmas 3 and 4, any quadrilateral and any pentagon can be guarded by 2 guards. Therefore, we divide the 7-gon into a pentagon and a quadrilateral, and we guard each with 2 guards separately, resulting in a total number of 4 guards. □

3 Orthogonal Polygons

In this section, we study the guarded guard problem for orthogonal polygons. In Section 3.1 we uncover some properties of orthogonal polygons. We will use these properties in our proof of the $\frac{n}{2}$ upper bound in Section 3.2. In the full version of the paper, we present our lower bound by exhibiting a family of polygons that require $\frac{3n-4}{7}$ guards for an arbitrary large n.

3.1 Preliminaries

We prove some properties of quadrangulated orthogonal polygons that will be used later in Section 3.2. However, these properties are of independent interest.

Observation 3 *Any simple polygon that admits a quadrangulation has an even number of vertices.*

Lemma 7 *Let P be an orthogonal polygon and Q be a quadrangulation of P. Any diagonal d of Q partitions P into two quadrangulated polygons P_1 and P_2 such that in each of P_1 and P_2 the edges that are adjacent to d are parallel to each other.*

Proof. Due to symmetry, we prove the statement only for P_1. Since P is orthogonal, its edges alternate between horizontal and vertical. Consequently, all edges of P_1 except d are orthogonal. The polygon P_1 admits a quadrangulation, which is inherited from Q. Thus, by observation 3, it has an even number of vertices and edges. Therefore, the two edges that are incident to d are either horizontal or vertical, and thus parallel to each other. □

The following observation is implied from Lemma 7.

Observation 4 *Let P be an orthogonal polygon and Q be a quadrangulation of P. Then every diagonal of Q is incident to at least one reflex vertex of P.*

Lemma 8 *Let $d = (a, b)$ be any diagonal in Q that divides P into two polygons P_1 and P_2. Let α_1, β_1 be the angles of P_1 at a and b, respectively, and let α_2, β_2 be the angles of P_2 at a and b, respectively. Then the following statements hold:*

1. *$\min\{\alpha_1, \beta_1\} \leqslant \pi$ and $\min\{\alpha_2, \beta_2\} \leqslant \pi$*
2. *$\alpha_1 + \beta_1 = \pi$ or $\alpha_2 + \beta_2 = \pi$.*

Proof. By Observation 4 one endpoint of d, say b, is a reflex vertex. We consider two cases:

1. *a is a convex vertex.* In this case the angle at a is $\frac{\pi}{2}$; see Figure 7(a). Statement 1 holds because α_1 and α_2 are both smaller than $\frac{\pi}{2}$. To verify statement 2, observe that the two edges of P_1 that are incident to d are on the same side of the line through d. This and the fact that these edges are parallel (by Lemma 7) imply that $\alpha_1 + \beta_1 = \pi$. We get $\alpha_2 + \beta_2 = \pi$ by a similar argument.

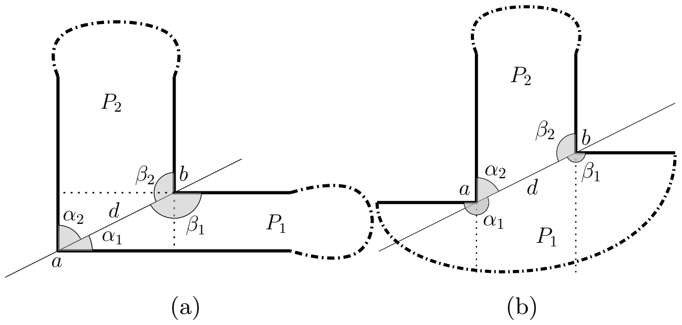


Figure 7: Possible configurations of a diagonal $d = (a, b)$ of P in Q.

2. *a is a reflex vertex.* In this case the two edges of, say P_2, that are incident to d are on the same side of the line through d, and the corresponding two edges in P_1 are on opposite sides of the line. See Figure 7(b). Therefore $\alpha_2 + \beta_2 = \pi$—this proves statement 2 and the second part of statement 1. It remains to show that $\min\{\alpha_1, \beta_1\} \leqslant \pi$. This statement is also true because $\alpha_1 + \beta_1 = 2\pi$ as the two edges of P_1, incident to d, lie on different sides of the line through d. □

3.2 The Upper Bound

In this section, we prove that any n-vertex orthogonal polygon P can be mutually guarded by at most $\frac{n}{2}$ guards. Hence, $\bar{g}(n) \leqslant \frac{n}{2}$. Note that n must be an even number, as P is orthogonal.

Theorem 9 *$\bar{g}(n) \leqslant \frac{n}{2}$, for every even $n \geqslant 4$.*

Proof. Let P be an orthogonal polygon, Q be a quadrangulation of P, and T be the dual tree of Q. Note that T has $\frac{n-2}{2}$ vertices of degree at most 4. We root T by taking an arbitrary leaf as the root. Thus, each node has at most 3 children (See Figure 8).

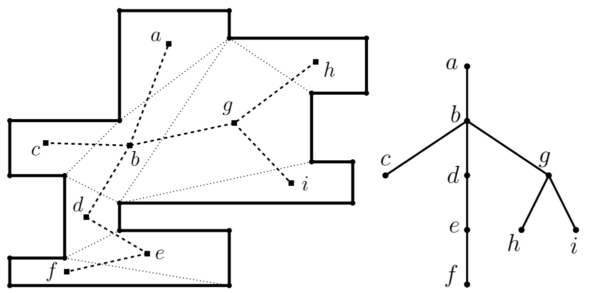


Figure 8: A quadrangulation and its dual tree.

Our proof is by partitioning T into connected subtrees with at least 2 and at most 4 nodes, and possibly one subtree of size 1. We partition T as follows:

Let l be a deepest leaf in T, i.e. a node with maximum distance from the root, and p be the parent of l. Let $T(p)$ be the subtree of T rooted at p, and note that $2 \leqslant |T(p)| \leqslant 4$ (assuming T has at least two nodes). We remove the vertices of $T(p)$ from T and repeat the above process. In the last iteration, the tree T may contain only one vertex which is the root, and we take it as a subtree with one node.

Consider any subtree $T(p)$ obtained after the above partitioning of T. Let polygon $\mathcal{P}$ be the union of quadrilaterals in Q corresponding to the nodes of $T(p)$. We guard $\mathcal{P}$ for each subtree separately. This would give a guarding of P. Assuming $2 \leqslant |T(p)| \leqslant 4$, we have $|\mathcal{P}| \in \{6, 8, 10\}$, as the number of vertices of $\mathcal{P}$ is given by $|\mathcal{P}| = 2|T(p)| + 2$.

In Lemmas 11, 12, and 13 we will show how to guard the corresponding polygon $\mathcal{P}$ of a subtree $T(p)$ with $|T(p)|$ guards. In case $T(p)$ is a 1-node tree, we guard $\mathcal{P}$ with 2 guards. This would give a total of $|T| + 1$ guards, which is $\frac{n-2}{2} + 1 = \frac{n}{2}$. □

Observe that if $|T(p)| = 1$, then $|\mathcal{P}| = 4$, in which case $\mathcal{P}$ can be guarded by 2 guards. In Lemmas 11, 12, and 13 we show how to guard a polygon $\mathcal{P}$ for a subtree $T(p)$, where $|T(p)| \in \{2, 3, 4\}$ and consequently $|\mathcal{P}| \in \{6, 8, 10\}$. In Figure 9, all possible configurations of $T(p)$ alongside a sample representation of its corresponding $\mathcal{P}$ are shown. Throughout the proofs, we use $\mathcal{P}$ to represent the union of the quadrilaterals that correspond to nodes of $T(p)$. We use $|\mathcal{P}|$ to denote the number of vertices of $\mathcal{P}$.

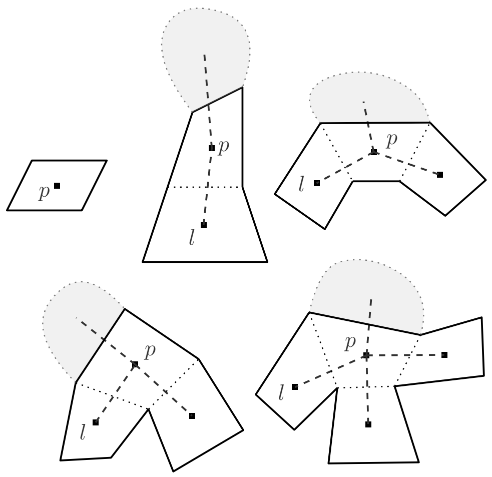


Figure 9: Possible polygonal patterns for $T(p)$.

Notice that the angles α_i and β_i in Lemma 8 are defined by the diagonals of Q and the boundary of P (not necessarily orthogonal). Also, notice that $\mathcal{P}$ is a simple polygon formed by some quadrilaterals in Q. Therefore, if we define α'_i and β'_i analogous to α_i and β_i but with respect to the diagonals of Q and the boundary of $\mathcal{P}$, then $\alpha'_i \leq \alpha_i$ and $\beta'_i \leq \beta_i$. Therefore, the following Corollary is valid.

Corollary 10 *Let $\mathcal{P}$ be a simple polygon formed by some quadrilaterals in Q. Let $d = (a, b)$ be any diagonal in Q that divides $\mathcal{P}$ into two polygons $\mathcal{P}_1$ and $\mathcal{P}_2$. Let α'_1, β'_1 be the angles of $\mathcal{P}_1$ at a and b, respectively, and let α'_2, β'_2 be the angles of $\mathcal{P}_2$ at a and b, respectively. Then the following statements hold:*

1. $\min\{\alpha'_1, \beta'_1\} \leqslant \pi$ *and* $\min\{\alpha'_2, \beta'_2\} \leqslant \pi$
2. $\alpha'_1 + \beta'_1 \leqslant \pi$ *or* $\alpha'_2 + \beta'_2 \leqslant \pi$.

While we only use the first statement in our proofs, we keep the second statement due to its independent interest and potential applications.

Lemma 11 *Two guards are sufficient for every $\mathcal{P}$ where $|\mathcal{P}| = 6$.*

Proof. A polygon $\mathcal{P}$ with 6 vertices consists of two quadrilaterals of Q, say A and B. Let a and b be the shared vertices between A and B—ab is the shared diagonal. We place one guard on a to cover A and one guard on b to cover B. Such a placement exists because A and B are convex quadrilaterals. The two guards are visible to each other as both cover the diagonal ab. □

Lemma 12 *Three guards are sufficient for every $\mathcal{P}$ where $|\mathcal{P}| = 8$.*

Proof. A polygon $\mathcal{P}$ with 8 vertices contains three quadrilaterals of Q. Let A be the quadrilateral corresponding to the child node l, B be the quadrilateral corresponding to the parent node p, and C be the quadrilateral corresponding to the sibling of l in $T(p)$. We consider two cases:

1. Quadrilaterals A and C share a vertex:

   Let b be the vertex shared between A and C, and let ab be the diagonal that separates A and B, and let bc be the diagonal that separates B and C, as in Figure 10(a). We place one guard on a to cover A, and one guard on c to cover C. As the quadrilaterals are convex, both guards cover vertex b. We place a third guard on b to cover B. Thus, the entire $\mathcal{P}$ is guarded, and the guards on a and c are mutually visible from the guard on b.

2. Quadrilaterals A and C has no common vertex:

   Let ab_1 be the diagonal that separates A and B and let cb_2 be the diagonal that separates B and C, such that ac corresponds to the edge of T that connects p to its parent, as in Figure 10(b). By statement 1 of Corollary 10, the angle at one of the endpoints of ac in polygon $\mathcal{P}$ is at most π. Due to symmetry, let a be this endpoint. By placing a guard on a we cover both A and B. We place one guard on b_2 to cover C. We place a third guard on c to cover B. The guards on a and c see each other through ac, and the guards on c and b_2 see each other through cb_2. Hence, $\mathcal{P}$ is mutually covered by three guards.

□

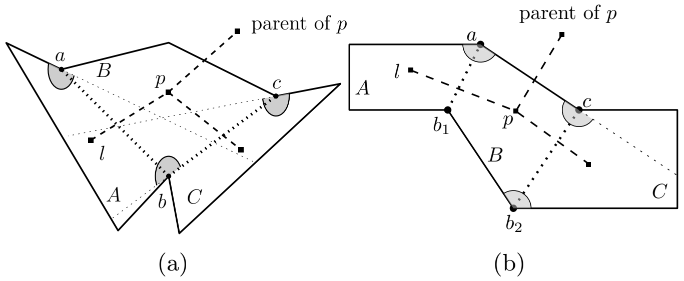


Figure 10: A and C share (a) one vertex (b) no vertex.

A proof of the following lemma is given in the full version of the paper.

Lemma 13 *Four guards are sufficient for every $\mathcal{P}$ where $|\mathcal{P}| = 10$.*

4 Open Problems

One natural problem is to improve any of the bounds given for $g(n)$ and $\bar{g}(n)$. We believe the true bounds are closer to our lower bounds.

Another research direction is to relax any of the constraints that is imposed on a feasible guard set in Section 1.2. For example one may allow outward guards, because in some cases they lead to smaller number of guards, as in Figure 11. Alternatively one may allow the two guards that are placed at the same vertex to be mutually visible to each other.

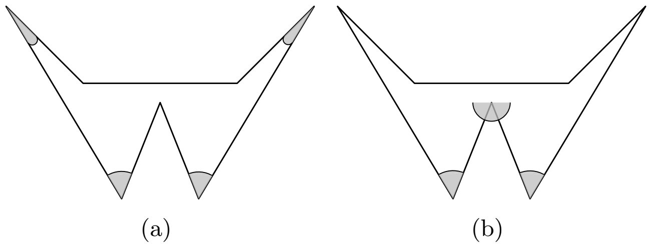


Figure 11: Outward guarding can result in less guards

References

[1] Y. Amit, J. S. B. Mitchell, and E. Packer. Locating guards for visibility coverage of polygons. *International Journal of Computational Geometry & Applications*, 20(5):601–632, 2010.

[2] A. Biniaz. Art galleries and mobile guards: Revisiting O'rourke's proof. In *32nd Annual European Symposium on Algorithms, ESA*, pages 27:1–27:4, 2024.

[3] A. Biniaz and M. Hashemi. City guarding with cameras of bounded field of view. In *35th Canadian Conference on Computational Geometry*, pages 71–75, 2023.

[4] P. Bose and T. C. Shermer. Guarding polygons with two mobile guards. *Computational Geometry*, 22(1-3):123–136, 2002.

[5] V. Chvátal. A combinatorial theorem in plane geometry. *J. Comb. Theory B*, 18:39–41, 1975.

[6] S. Fisk. A short proof of Chvátal's watchman theorem. *J. Comb. Theory B*, 24:374, 1978.

[7] D. Florentino, E. Moy, and R. Muth. Cooperative half-guards in art galleries, 2020.

[8] R. Honsberger. *Mathematical Gems II*. The Mathematical Association of America, Washington, 1976.

[9] J. Kahn, M. Klawe, and D. Kleitman. Traditional galleries require fewer watchmen. *SIAM Journal on Algebraic Discrete Methods*, 4(2):194–206, 1983.

[10] B. C. Liaw, N. F. Huang, and R. C. T. Lee. The minimum cooperative guards problem on k-spiral polygons. In *5th Canadian Conference on Computational Geometry*, pages 97–101, 1993.

[11] B.-C. Liaw and R. Lee. An optimal algorithm to solve the minimum weakly cooperative guards problem for 1-spiral polygons. *Information Processing Letters*, 52(2):69–75, 1994.

[12] T. Michael and V. Pinciu. Art gallery theorems for guarded guards. *Computational Geometry*, 26(3):247–258, 2003.

[13] J. O'Rourke. An alternate proof of the rectilinear art gallery theorem. Technical Report 82-15, Johns Hopkins University, December 1982.

[14] J. O'Rourke. Galleries need fewer mobile guards: a variation on chvátal's theorem. *Geometriae Dedicata*, 14:273–283, 1983.

[15] J. O'rourke. *Art gallery theorems and algorithms*, volume 57. Oxford University Press Oxford, 1987.

[16] G. H. Peñalver. Vigilancia vigilada de polígonos ortogonales. In *VI Encuentros de Geometría Computacional*, pages 198–205, 1995.

[17] V. Pinciu. A coloring algorithm for finding connected guards in art galleries. In *Discrete Mathematics and Theoretical Computer Science, DMTCS*, 2003.

[18] B. Speckmann and C. D. Tóth. Allocating vertex π-guards in simple polygons via pseudo-triangulations. *Discrete & Computational Geometry*, 33:345–364, 2005.

[19] C. Tóth. Art gallery problem with guards whose range of vision is 180°. *Computational Geometry*, 17:121–134, 2000.

[20] P. Żyliński. Watched guards in art galleries. *Journal of Geometry*, 84(1-2):164–185, 2006.

[21] P. Żyliński. Some results on cooperative guards. Technical Report 30/02 PG, Gdańsk University of Technology, ETI Faculty, 2002.

Multiple Watchman Routes in Staircase Polygons[*]

Anna Brötzner[†] Bengt J. Nilsson[*] Christiane Schmidt[‡]

Abstract

We consider the watchman route problem for multiple watchmen in staircase polygons, which are rectilinear x- and y-monotone polygons. For two watchmen, we propose an algorithm to find an optimal solution that takes quadratic time, improving on the cubic time of a trivial solution. For $m \geq 3$ watchmen, we explain where this approach fails, and present an approximation algorithm for the min-max criterion with only an additive error.

1 Introduction

The watchman route problem asks for a shortest route inside a polygon, such that every point in the polygon is visible to some point on the route. It was first introduced by Chin and Ntafos [3], who showed that the problem is NP-hard for polygons with holes, but may be solved efficiently for simple polygons. Given a starting point, an optimal route can be computed in $O(n^3)$ time [9], and finding a solution without a fixed starting point takes a linear factor longer [8].

The Watchman Route Problem has also been considered for multiple watchmen (a problem introduced in [2]). For histograms, efficient algorithms have been proposed for minimizing the total route length (min-sum) [2] and the length of the longest route (min-max) [7]. For two watchmen, Mitchell and Wynters [5] proved NP-hardness for the min-max objective in simple polygons. Recently, Nilsson and Packer presented a polynomial-time 5.969-approximation algorithm for the same objective in simple polygons [6].

In this paper, we consider a quite restricted class of polygons, staircase polygons, that for two watchmen allows us to assign the responsibility for guarding any edge solely to one of the two watchmen (and for two watchmen, seeing the complete polygon boundary is sufficient to see the whole polygon, as shown in [6]). Additionally, we show that the two routes can be separated by a diagonal between two reflex vertices. This enables a polynomial-time algorithm to compute the optimal two watchman routes (for both the min-max and the min-sum objective). Despite staircase polygons being so restricted, some of the observations we make do not hold for three or more watchman routes. This indicates a discrepancy in the computational complexity between the watchman route problem for one or two watchmen and for multiple watchmen. We therefore propose an approximation algorithm for m watchmen for the min-max criterion with only additive error that depends on the polygon.

[*]A. B., B. J. N. and C. S. are supported by grant 2021-03810 (Illuminate: provably good algorithms for guarding problems) from the Swedish Research Council (Vetenskapsrådet).

[†]Department of Computer Science and Media Technology, Malmö University, Sweden, {anna.brotzner,bengt.nilsson.TS}@mau.se

[‡]Department of Science and Technology, Linköping University, Sweden, christiane.schmidt@liu.se

2 Notation and Preliminaries

A polygon is called *rectilinear* if all its edges are parallel to the x- or the y-axis of a given coordinate system, and *x-monotone* (*y-monotone*) if every line that is orthogonal to the x-axis (y-axis) intersects the polygon in exactly one connected interval. A *staircase polygon* is a rectilinear polygon that consists of two x-y-monotone chains that intersect only at their endpoints. The vertices along a chain then alternatingly have angles of 90 and 270 degrees; we refer to the former ones as *convex vertices* and to the latter ones as *reflex vertices*. We call the polygonal chain of boundary edges that lie above and below the interior the *ceiling* and the *floor* of P, respectively. We consider the watchman route problem for multiple watchmen in staircase polygons, where every route is a closed route.

Multiple Watchman Route Problem (m-WRP). Given a polygon P, and a number of watchmen m, find a shortest set of m closed routes, with respect to the min-sum or min-max criterion, such that every point in P is seen from at least one of the routes.

We denote the length of a route w by $\|w\|$. In the following, if we refer to a watchman w we mean that w is the route that the watchman walks. We refer to a solution of the m-WRP as a set of m watchman routes in P. In the following, we consider the m-WRP for the min-sum and the min-max criterion. Any statement on optimal watchman routes holds for either objective, unless stated otherwise.

Let P be a staircase polygon that is not guardable with two point guards. As P is x- and y-monotone, we make the following claim.

Observation 1 *A watchman walking along a route w with leftmost coordinate $x_{\min}$ and rightmost coordinate $x_{\max}$ sees all points $p \in P$ with $x(p) \in [x_{\min}, x_{\max}]$.*

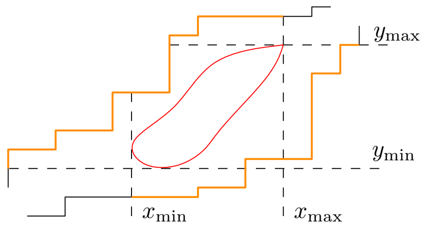


Figure 1: The orange parts of the polygon boundary are seen from the watchman route (red).

The analogous statement holds for the lowest coordinate $y_{\min}$ and the uppermost coordinate $y_{\max}$ of watchman route w. Watchman w thus sees the contiguous part of the ceiling between $y_{\min}$ and $x_{\max}$, and the contiguous part of the floor between $x_{\min}$ and $y_{\max}$, see Figure 1.

A *cut* is a directed line segment in P with both end points on boundary of P and where each interior point is an interior point of P. A cut always separates P into exactly two sub-polygons of nonzero area. If a cut is represented by the segment $[p, q]$ we say that the cut is directed from p to q. For a cut c in P, we define the *left polygon*, $L(c)$, to be the set of points in P locally to the left of c according to c's direction.

Assume a counterclockwise walk of the boundary of P. Such a walk imposes a direction on each of the edges of P in the direction of the walk. Consider a reflex vertex of P. The two edges incident to the vertex can each be extended inside P until the extensions reach a boundary point. These extended segments form cuts given the same direction as the edge they are collinear to. We call these cuts *extensions.*

Given a set of cuts, we say that a cut c *dominates* a cut c' if the subpolygon $L(c)$ is a subset of $L(c')$. The non-dominated cuts are called *essential cuts.* A cut c is visited by a watchman if its route has a point on c or in $L(c)$. The essential cuts are exactly those cuts that need to be visited if the polygon is to be seen by a single watchman. Clearly, visiting all essential cuts is also a necessary condition for a set of m watchman routes.

A staircase polygon has at most four essential cuts: the leftmost vertical extension of the floor v_{left}, the lowest horizontal extension of the ceiling h_{bot}, the rightmost vertical extension of the ceiling v_{right}, and the topmost horizontal extension of the floor h_{top}. Note that not necessarily all of these four extensions are essential cuts, but at least one of v_{left} and h_{bot}, and one of v_{right} and h_{top} is essential (if only one cut of a pair is essential, it dominates the other cut of the pair, and visiting the essential cut of such a pair guarantees that also the other cut of the pair is visited). For the sake of simplicity, we still refer to all four of them as essential cuts.

For one watchman, an optimal solution is given by the shortest route that visits all essential extensions. An example is shown in Figure 2(a). Chin and Ntafos [3], prove that such a route can be computed in linear time.

Theorem 1 *(Theorem 2, Chin, Ntafos [3]) A shortest watchman route in simple rectilinear polygons can be found in $O(n)$ time.*

For multiple watchman routes, the watchmen share the responsibility of seeing P. Thus, we aim to find a "good" distribution of responsibilities among the watchmen. For two watchmen, we prove that the polygon may be split into two subpolygons such that an optimal solution to the 2-WRP corresponds to an optimal solution to the WRP in each subpolygon.

3 An Algorithm for Optimal Two Watchmen

In this section, we investigate the 2-WRP. Let us first state some properties of two optimal watchman routes in staircase polygons.

Lemma 2 *There exists an optimal solution (w_1^*, w_2^*) to the 2-WRP in a staircase polygon P that satisfies the following properties:*

1. *w_1^* and w_2^* do not have any common x- and y-coordinate.*
2. *w_1^* visits the essential cuts $h_{\text{bot}}, v_{\text{left}}$, and w_2^* visits the essential cuts $h_{\text{top}}, v_{\text{right}}$.*
3. *There exists a pair of reflex vertices (r, r') with r on the floor and r' on the ceiling, such that $\overline{rr'}$ separates w_1^* and w_2^*, see Figure 2(b).*

Proof. First, we prove Properties 1 and 2. Let (w_1, w_2) be an optimal solution to the 2-WRP in P. Since P is seen, each of the four essential cuts is visited by some watchman. This can be done in four combinatorially different ways. We distinguish these four cases and prove that in an optimal solution only Case 4 given by Property 2 occurs. If one essential cut is dominated by another one, then both of them are visited by the same watchman (that is, not all four cases are possible).

Case 1: w_1 visits all 4 extensions (Figure 3(a)). Then, w_1 is a watchman route in P, and w_2 is redundant. Therefore, assume that $\|w_2\| \leq \|w_1\|$, and assume that h_{top} is non-dominated. Cut P into two subpolygons along h_{top}, and denote the subpolygon below h_{top} by P_1, the one above h_{top} by P_2. Then, P_1 is seen by w_1. P_2 is star-shaped and may therefore be guarded by a single watchman w_2' with route length 0. Hence, replacing w_1 with an optimal watchman route w_1' in P_1 yields a solution (w_1', w_2') for P that is shorter than (w_1, w_2), both for the min-sum and for the min-max criterion: w_1' will never visit h_{top}, but the essential cuts of P_1, and is thus strictly shorter than w_1; $0 = \|w_2'\| \leq \|w_2\|$, with equality in case w_2 was point-shaped. This contradicts the optimality of (w_1, w_2).

Case 2: w_1 visits three essential cuts, w_2 visits the fourth one (Figure 3(b)). Assume w.l.o.g. that w_1 visits $h_{\text{bot}}, v_{\text{left}}$, and h_{top}. We shorten w_1 and w_2 as in Case 1.

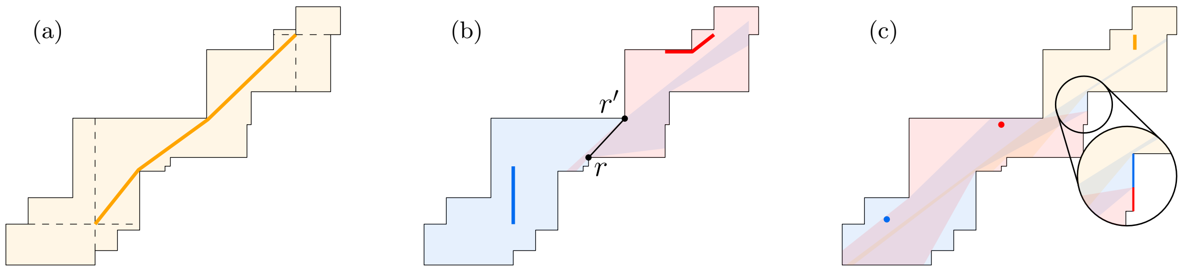


Figure 2: Optimal solutions for (a) one watchman, (b) two watchmen, (c) three watchmen.

Case 3: w_1 visits v_{left} and h_{top}, and w_2 visits h_{bot} and v_{right} (Figure 3(c)). If the two routes have neither x- nor y-overlap, then observe that w_2 lies to the right of the rightmost point of w_1. We translate w_1 such that its lowest intersection with v_{left} lies in the point of intersection of h_{bot} with v_{left}. Analogously, we move w_2 such that its uppermost intersection with v_{right} lies in the point of intersection of h_{top} with v_{right}. By construction, the subpolygon below h_{bot} (above h_{top}), and the subpolygon left of v_{left} (right of v_{right}), are star-shaped and w_1 (w_2) visits its kernel. Let P' be the subpolygon between the essential cuts. Watchman w_1 (w_2) sees the ceiling (floor) of P', and moving it vertically does not affect this except possibly losing sight of an interval on h_{top} (h_{bot}), which will be seen by the translated w_2 (w_1). Hence, by Lemma 3.1 in [6] (which states that a simple polygon is seen by two watchmen if its boundary is seen), P' is seen as well.

Assume now w.l.o.g. that there is y-overlap between the two routes. Let q_1 be the point of intersection of v_{left} with h_{bot}, and let q_2 be the point of intersection of v_{right} with h_{top}. We distinguish between the min-sum and the min-max criterion.

For the min-max case, consider the vertical segment s_1 between q_1 and the intersection of v_{left} with w_1, and the vertical segment s_2 between q_2 and the intersection of v_{right} with w_2, and assume w.l.o.g. that $\|s_1\| \geq \|s_2\|$. Consider the route $w_2 \cup s_2$, and shorten it by removing the connected part of length $\|s_2\|$ that starts at h_{bot} to obtain a new route w_2' of the same length as w_2. Then we substitute w_1 by a vertical segment w_1' of length $\|s_2\|$ with lowest point q_1. The new route w_1' is shorter than w_2'. Moreover, P is seen from (w_1', w_2') because the routes touch q_1 and q_2, thereby covering $P \setminus P'$, and P' is covered since w_1' and w_2' have y-overlap.

For the min-sum case, observe that $\|s_1\| \leq \|w_2\|$. We construct a new route $w_1' = w_1 \cup s_1$, and replace w_2 by a route w_2' of length 0 at its endpoint on v_{right}. Then, the sum of the route lengths does not increase, and the routes satisfy the conditions of Case 2.

Case 4: w_1 visits h_{bot} and v_{left}, and w_2 visits h_{top} and v_{right} (Figure 3(d)). Assume w.l.o.g. that w_1 and w_2 have some x-overlap and let ℓ be the leftmost vertical line that intersects both routes. Then, ℓ cuts P into

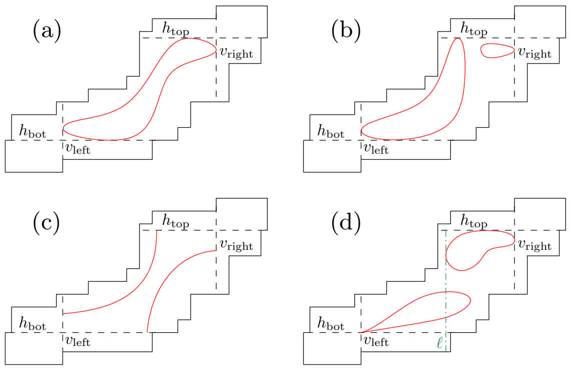


Figure 3: Four possibilities for w_1 and w_2 to visit the up to four essential cuts.

subpolygons which both are staircase polygons. Denote the subpolygon to the left of ℓ with P_1, and the one to the right of ℓ with P_2. Let w_1' be the watchman route that consists of the part of w_1 that lies in P_1, together with the straight-line segment between the points of intersection of w_1 with ℓ (this may also be just a single point). By Observation 1, P_1 is seen by w_1'. Similarly, P_2 is seen by w_2. Thus, (w_1', w_2) is a shorter solution for P than (w_1, w_2). Furthermore, these routes may still be improved: If both P_1 and P_2 are seen, then P is seen. We may thus replace w_1' with an optimal watchman route in P_1, and w_2 with an optimal watchman route in P_2. None of these optimal routes touches ℓ, and therefore they together yield a solution for P that is shorter than (w_1, w_2) where both routes do not share a common x-coordinate.

Let us now show Property 3 to finish the proof. For this, assume that w_1 and w_2 are two optimal watchman routes that satisfy Properties 1 and 2. Consider all vertical extensions of reflex vertices ordered from left to right. Assume that the interior of every strip between two consecutive such extensions is entered by w_1 and w_2. Then, we may shorten both w_1 and w_2 as in Case 4 (with the two vertical extensions of the entered strip playing the role of ℓ), without losing visibility, as both tours will see the complete rectangle spanned by the two extensions, leading to a contradiction. Hence, there exists an empty strip. The diagonal connecting the two opposite reflex vertices of the corresponding extensions does not intersect any of w_1 or w_2. □

In the following, we always assume that an optimal solution (w_1^*, w_2^*) obeys Properties 1–3 of Lemma 2. In particular, w_1^* lies below and to the left of w_2^*.

Lemma 3 *In an optimal solution to the 2-WRP in a staircase polygon, for every polygon edge there exists a watchman that sees the edge completely.*

Proof. Let (w_1^*, w_2^*) be an optimal solution, and consider w_1^*. As soon as it crosses the extension of a horizontal floor edge e, it sees e completely since nothing blocks the visibility between w_1^* and e along e's extension. Similarly, w_1^* sees a vertical edge on the ceiling completely as soon as it crosses the edge's extension. Before crossing the extension, w_1^* does not see the respective edge at all. Hence, for any horizontal floor edge (vertical ceiling edge) e, if w_1^* sees any point on e, then it sees all points of e. Similarly, for any horizontal ceiling edge (vertical floor edge) e, if w_2^* sees any point on e, then it sees all points of e. Assume w.l.o.g. that there is a horizontal floor edge e such that no point on e is seen by w_1^*. Then, w_2^* sees e completely as otherwise there are points on e that are not seen at all. □

We can therefore separate the two optimal routes.

Lemma 4 *Let (w_1^*, w_2^*) be an optimal solution in a staircase polygon P. There exists a unique diagonal between a vertex on the floor and a vertex on the ceiling that cuts P into two subpolygons P_1 and P_2 such that w_1^* sees P_1, and w_2^* sees P_2.*

Proof. By Lemma 3, every edge is completely seen by a watchman. For a chain of consecutive edges on the floor or ceiling, there cannot be an alteration in the responsibility of the watchmen: Let e_i, e_{i+1}, and e_{i+2} be three consecutive edges (on the floor or ceiling). If one watchman sees e_i and e_{i+2} completely, then it also sees e_{i+1}. Hence, there exist vertices on the floor and the ceiling such that w_1^* sees all edges that lie below and to the left of them completely, and w_2^* sees all edges that lie above and to the right of them completely. We call such vertices *breaking points* and show that there exist two breaking points, one on the floor and one on the ceiling, that see each other—these define the unique diagonal. Assume that this is not the case. Let b_f be the lowest-leftmost breaking point on the floor, and b_c be the upper-rightmost breaking point on the ceiling. W.l.o.g., assume that all breaking points on the floor lie to the upper-right of the breaking points on the ceiling (in particular, b_f lies to the upper-right of b_c).

Since b_f and b_c do not see each other, there exist some edges incident to a reflex vertex r that block the visibility. Assume that these edges lie on the ceiling. Then, the horizontal edge incident to r lies above b_c

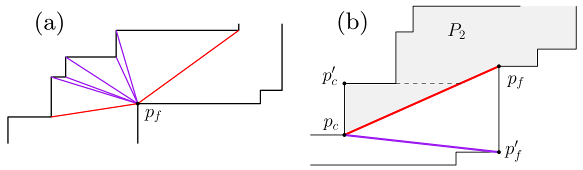


Figure 4: (a) The candidate diagonals of a reflex vertex p_f: two with positive slope (red), and several with negative slope (purple). (b) A diagonal with positive slope (red) that is not a candidate. An optimal route in P_2 (gray) needs to visit the same essential cut (dashed) as an optimal watchman route in the subpolygon induced by $\overline{p_c p'_f}$ (purple).

and below b_f, and is seen by w_2^* (by definition of b_c). Hence, w_2^* sees the vertical floor edge v that is hit by the horizontal extension through r (as described in the proof of Lemma 3), and thereby also the convex vertex on the lower end of v, contradicting the choice of b_f (being the lowest-leftmost breaking point on the floor). □

Observe that Lemma 3 only holds for two watchmen. For three or more watchmen, some edges may only be seen partially by each watchman in an optimal solution. An example is shown in Figure 2(c). The blue watchman is in charge of monitoring a part of a vertical floor edge above the red watchman's visibility region. The yellow watchman does not see this edge at all and would have to walk very far to reach the vertical extension of this edge. Therefore, an optimal solution for $m \geq 3$ watchmen may induce a split of the polygon's floor and ceiling into more than m parts each, such that every part is seen by a single watchman. This means that a watchman may be "in charge of" more than one contiguous part of the boundary on the floor and ceiling.

We present an algorithm that finds an optimal split, and thus computes an optimal solution for two watchmen in $O(n^2)$ time. By Lemma 4, we consider all diagonals between vertices on the floor and on the ceiling. Any such diagonal splits P into two subpolygons. For each of them, we compute an optimal watchman route using a modified version of the linear-time algorithm by Chin and Ntafos [3], and then combine the two routes to a solution for the 2-WRP in P.

As there are at most quadratically many diagonals to consider, this procedure trivially yields a cubic-time algorithm. However, maintaining a similar structure of the subpolygons by dealing with the diagonals in a certain order allows us to compute many of the watchman routes in amortized constant time.

To this end, we iterate over the vertices on the floor. For each floor vertex p_f, we compute all its diagonals to points on the ceiling, in clockwise order around p_f. If p_f is a convex vertex, then all diagonals have a negative slope. If p_f is a reflex vertex, some diagonals have positive slope. However, we do not need to consider all diagonals with positive slope, but only those two that are followed or preceded by a negative-slope diagonal in clockwise order. We call those, and the diagonals with negative slopes, *candidate diagonals*; see Figure 4(a). Every candidate diagonal splits P into two subpolygons, P_1 below and P_2 above the diagonal.

Lemma 5 *Any diagonal that is not a candidate diagonal induces a solution that is at least as long as the solution induced by some candidate diagonal.*

Proof. First, note that a diagonal of positive slope is spanned between two reflex vertices. Consider w.l.o.g. a non-candidate diagonal $\overline{p_f p_c}$, as seen in Figure 4(b). Then there is a convex vertex p'_c above p_c that does not yield a diagonal of p_f because $y(p_c) < y(p_f)$. The subpolygon P_2 above $\overline{p_f p_c}$ has the horizontal line through p'_c as an essential cut. Hence, the watchman route in P_2 has points below this cut. There exists a subpolygon induced by a candidate diagonal (incident to p_c and with the other endpoint p'_f below p_f) that also has the horizontal line through p'_c as an essential cut. The watchman route in the subpolygon above $\overline{p_c p'_f}$ remains the same, and the watchman route in the subpolygon below is not longer than the one induced by $\overline{p_f p_c}$. □

Now we compute a solution for each candidate diagonal in the following manner.

Step 1: Consider a diagonal with negative slope. Cutting along this diagonal creates only convex vertices in each subpolygon, hence all four essential cuts per subpolygon are rectilinear. The watchman routes touch these extensions, but do not cross them [3]. We compute the optimal solutions for the subpolygons induced by the first diagonal in clockwise order in linear time by Theorem 1. In addition, we compute two shortest-path-tree data structures [4]. One is rooted at the first reflex vertex on the floor and stores the shortest paths to all other floor vertices, the other one is rooted at the first reflex vertex on the ceiling and stores the shortest paths to all other ceiling vertices.

Then, for each diagonal in order, we update the solution in the following way. Moving from one diagonal to the next (i.e., moving from one vertex on the ceiling to the next) alters either the essential cut $v_{\text{right}}(P_1)$ of P_1, or the essential cut $h_{\text{bot}}(P_2)$ of P_2. During this movement, any reflex vertex on the ceiling touched by the route can only be released once per vertex p_f, and they are released from right to left. Similarly, any reflex vertex on the floor can be added as an anchor point only once per vertex p_f, and they get added from left to right. Hence, the number of updates per vertex p_f is at most linear.

When updating the route w_1 in P_1, we move from one vertical extension $v_{\text{right}}(P_1)$ to the next one $v'_{\text{right}}(P_1)$. We use the shortest-path-tree of the floor to check whether vertices on the floor get added to, and the shortest-path-tree of the ceiling to check whether vertices on the ceiling get released from the route. This can be done in amortized constant time [4].

Step 2: If p_f is a reflex vertex, we need to consider also the two candidate diagonals with positive slope. Here, the subpolygons' essential cuts differ from those of a staircase polygon: There is exactly one non-rectilinear essential cut, namely the extension of the diagonal. We may nevertheless compute an optimal solution, using the algorithm by Chin and Ntafos [3]. This algorithm defines a set of essential cuts, along which the polygon is reflected. Computing the shortest path from one of these essential cuts to its copy yields the shortest watchman route in the original polygon. Since there are at most five essential cuts, we can try all combinations of subsegments of these essential cuts and apply the Chin-and-Ntafos reduction which takes linear time in each of these constant number of cases.

Thus, the computations for each vertex p_f take amortized linear time. As we do this for every vertex on the floor, there are linearly many vertices to consider. With this, we get an optimal solution to the 2-watchman route problem in staircase polygons.

Theorem 6 *An optimal solution to the 2-WRP in staircase polygons can be computed in $O(n^2)$ time.*

4 An Approximation Algorithm for Min-Max Multiple Watchman Routes

For the general case of m watchmen, we propose an approximation algorithm for the min-max criterion. We consider a canonical set of watchman routes that always span the distance between the floor and the ceiling, thus avoiding the situation depicted in Figure 2(c). Using dynamic programming, we can find a set of min-max canonical routes. We formalize this in Definitions 1 and 2.

Definition 1 *Let p_c be a convex vertex on the ceiling, and p_f be a convex vertex on the floor. Let v_c be the maximal vertical line segment in P incident to p_c, and let h_f be the maximal horizontal line segment in P incident to p_f. If v_c and h_f intersect in a point s, we define the* left arm *of p_c and p_f to be the segment pair $\overline{p_c s} \cup \overline{p_f s}$. Then, the* left elbow *of p_c and p_f is the closure of the part of the left arm that lies strictly in the interior of P, and s is its* (left) elbow point.

Symmetrically, we define the right arm *and the* right elbow *with its* (right) elbow point *of p_c and p_f by considering the horizontal line segment incident to p_c and the vertical line segment incident to p_f.*

Given two elbows E and E', where E is defined by the two vertices p_c and p_f on the ceiling and the floor, and E' is defined by the two vertices p'_c and p'_f on the ceiling and the floor, respectively, where p'_c does not lie

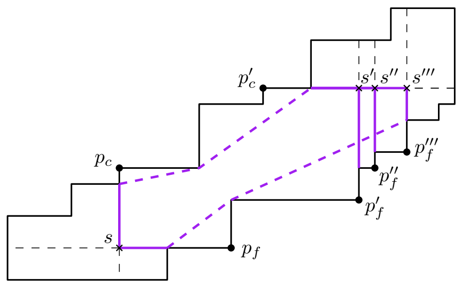


Figure 5: A left elbow defined by p_c and p_f, and right elbows defined by p'_c and p'_f, p''_f, and p'''_f (purple).

to the left of p_c and p'_f does not lie to the left of p_f, we say that E *precedes* E' and denote this by $E \prec E'$.

For our approximation algorithm, we consider watchman routes that consist of a left elbow E, a right elbow E' with $E \prec E'$, and the two shortest paths that connect the ceiling endpoints of the two elbows, and the floor endpoints of the two elbows, see Figure 5. We call such a route an *E-route*, and denote its length by $\|T(E, E')\|$.

Definition 2 *A set of watchman routes is called a* canonical solution *if every watchman route is an E-route and every point in P is seen from at least one route.*

By Observation 1, every point in P that lies between the leftmost and the rightmost point, or the lowest and the uppermost point of an E-route is seen from the route. Thus, we only need to make sure that the points between two E-routes are seen as well. Let E' be a right elbow and E be a left elbow, $E' \prec E$. We say that E' and E *partner* each other if every part in P that lies to the right or above E' and to the left or below E is seen from some point on E' or E. To find a shortest canonical solution for a polygon P, we consider all elbows in P, sorted by the coordinates of their elbow points, first from left to right and then from bottom to top.

Let E_l be a left elbow from which every point in P that lies to the left or below E_l is seen, and let E_r be a right elbow from which every point in P that lies to the right or above E_r is seen. We call such an elbow an *outer* elbow. Assume $E_l \prec E_r$. E_l and E_r define an E-route and, furthermore, every pair of convex vertices on the floor and ceiling that lie between the vertices that define E_l and E_r may define a left and a right elbow. Let $\mathcal{E}$ be the set of elbows between E_l and E_r, and let $R(\mathcal{E})$ be the set of right elbows in $\mathcal{E}$. $L^{\rightarrow}(\mathcal{E}, E')$ is the set of left elbows in $\mathcal{E}$ partnering the right elbow E'. $A(\mathcal{E}, E)$ is the set of all elbows in $\mathcal{E}$ preceding the right elbow E. The following recurrence computes the length of a min-max set of at most m E-routes.

$$\mathcal{L}\big(\mathcal{E}, E_l, E_r, m\big) =$$

$$\min_{\substack{E_i \in R(\mathcal{E}) \\ E_j \in L^{\rightarrow}(\mathcal{E}, E_i)}} \begin{cases} \max\Big\{\|T(E_j, E_r)\|, \\ \quad \mathcal{L}\big(A(\mathcal{E}, E_i), E_l, E_i, m-1\big)\Big\} & \text{if } m > 1 \\ \|T(E_l, E_r)\| & \text{if } m \geq 1 \end{cases}$$

We apply dynamic programming to the recurrence, using all possible outer left elbows E_l and outermost right elbows E_r. For this, we preprocess the polygon to obtain a data structure that allows us to test whether two elbows are partnering in constant time. For each extension of a ceiling edge, we identify the edge on the floor that it hits, and similarly for the extensions of floor edges. This takes linear time.

Let E' be a right elbow of vertices p'_c and p'_f on the ceiling and floor, respectively, let E be a left elbow of vertices p_c and p_f on the ceiling and floor, respectively, and let $E' \prec E$. We can determine whether E' and E see all points between them by considering the edges that are hit by the extensions aligned with the elbows. If there is an unseen convex vertex between them, then E' and E are not partnering. Otherwise, the relevant region between them has constant complexity and we can determine visibility in constant time.

Lemma 7 *Recurrence $\mathcal{L}\big(\mathcal{E}, E_l, E_r, m\big)$ gives the length of a min-max canonical solution of up to m E-routes using the outer elbows E_l and E_r in $O(mn^4)$ time.*

Proof. All of P is seen because of Observation 1, the choice of E_l and E_r, and the fact that for every right elbow E_i that is selected, the left elbow E_j partners E_i. The recurrence considers all possible combinations of left and right elbow pairs, and takes the minimum length canonical solution defined by these. Since there are up to $O(n^2)$ left and right elbows, there are up to $O(n^4)$ pairs to consider. Testing whether a left and a right elbow are partnering each other can be done in constant time using linear time preprocessing. Moreover, the recursion splits the polygon into at most m subpolygons, thus the total runtime is $O(mn^4)$. □

As there are $O(n^2)$ outer left elbows, and $O(n^2)$ outer right elbows, running the algorithm for all such pairs then takes $O(mn^8)$ time and computes a shortest canonical solution. This dominates the cost of preprocessing.

Lemma 8 *Let W^* be an optimal solution to the m-WRP in a staircase polygon P, and let W be the canonical solution computed by the approximation algorithm. Then, $\|W\| \leq \|W^*\| + 4(h_{\max} + v_{\max})$, where $h_{\max}$ is the length of a longest horizontal line segment in P, and $v_{\max}$ is the length of a longest vertical line segment in P.*

The proof of Lemma 8 can be found in the appendix and in the full version of this paper [1]. The algorithm, albeit having approximation factor one, since the approximation error is additive, will never compute a route of length zero. Hence, it cannot be used to obtain a polynomial time solution to minimum point guarding in staircase polygons. A polynomial time solution for this problem remains elusive.

References

[1] A. Brötzner, B. J. Nilsson, and C. Schmidt. Multiple watchman routes in staircase polygons. https://arxiv.org/abs/2507.01940, 2025.

[2] S. Carlsson, B. J. Nilsson, and S. C. Ntafos. Optimum guard covers and m-watchmen routes for restricted polygons. *Int. J. Comput. Geom. Appl.*, 3(1):85–105, 1993.

[3] W. Chin and S. C. Ntafos. Optimum watchman routes. *Inf. Process. Lett.*, 28(1):39–44, 1988.

[4] D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[5] J. S. B. Mitchell and E. L. Wynters. Watchman routes for multiple guards. In *Proc. 3rd CCCG*, volume 9, pages 293–327, 1991.

[6] B. J. Nilsson and E. Packer. Approximation algorithms for the two-watchman route in a simple polygon. *Algorithmica*, 86(9):2845–2884, Sep 2024.

[7] B. J. Nilsson and S. Schuierer. Shortest m-watchmen routes for histograms: the minmax case. In *Proceedings ICCI '92: Fourth International Conference on Computing and Information*, pages 30–33, 1992.

[8] X. Tan. Fast computation of shortest watchman routes in simple polygons. *Information Processing Letters*, 77(1):27–33, 2001.

[9] X. Tan and B. Jiang. Efficient algorithms for touring a sequence of convex polygons and related problems. In T. V. Gopal, G. Jäger, and S. Steila, editors, *Theory and Applications of Models of Computation - 14th Annual Conference, 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, volume 10185 of *LNCS*, pages 614–627, 2017.

A Proof of Lemma 8

Proof. Let $w \in W^*$, and transform w as follows: from the leftmost point p_c of w, extend a maximal segment towards the left until it hits a vertical edge of the ceiling at a point q_c. Similarly, from its lowest point p_f extend a maximal segment downwards until it hits a horizontal edge on the floor at a point q_f. Since w is polygonal, there is a (possibly empty) subpath of w that is both below p_c and to the left of p_f. We denote this subpath $w_d \subseteq w$. Let E be the left elbow defined by the convex vertices of the edges that contain q_c and q_f and let r_c and r_f denote the reflex vertices of these edges respectively.

We replace the subpath w_d of w by $\overline{p_c q_c} \cup \overline{q_c r_c} \cup E \cup \overline{r_f q_f} \cup \overline{q_f p_f}$, thereby extending it by the additive term $\leq 2(h_{\max} + v_{\max})$. Let p'_c and p'_f be the topmost and rightmost points of w, respectively. Let w_u be the (possibly empty) subpath to the right of p'_c and above p'_f. Let q'_c be the point directly above p'_c and q'_f be the point to the right of p'_f. Define r'_c and r'_f as the reflex vertices of the edges containing q'_c and q'_f and let E' be the corresponding right elbow. We make the analogous

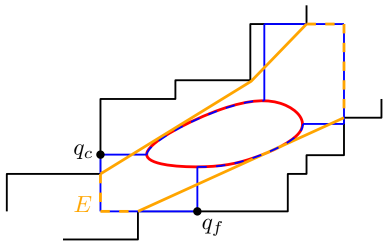


Figure 6: Transforming an optimal route w (red) into an E-route (orange).

replacement of w_u for this case; see Figure 6, giving us a route w'. The new route w' has length at most $\|w\| + 4(h_{\max} + v_{\max})$, by the previous argument. $\square$

On Super-Guarding Convex and Star-Shaped Polygons

Gabriel Aldous* Seth Barber* Alper Üngör*

Abstract

Given a simple polygon, the super-guarding problem asks to determine the number and placement of opaque guards such that every point in the polygon is guarded by at least k guards. We suggest algorithms to solve the k-guarding problem in some special simple polygons by placing guards on Bézier curves, and present some new bounds on special cases of the problem in two and three dimensions.

1 Introduction

Art gallery problems deal with determining the number and placement of guards (or cameras) needed to surveil an art gallery, which is typically modeled as a simple polygon. We refer to the book by O'Rourke [9] for comprehensive coverage of concepts on many versions of these problems. Here, we study a version of the problem discussed in [8].

Problem Statement. Given a polygon P of n points, determine the number and placement of opaque guards such that every point in P is guarded by at least k-guards. The guards may not be co-located and cannot see through each other. Designing algorithms as well as establishing bounds for various cases of the problem are of interest in this paper.

Contribution. Our contributions include (i) an alternative simpler algorithm to the one given in [8] to k-guard a convex polygon; (ii) extensions of bounds and algorithms for various other input such as star-shaped polygons and unbounded polygons; (iii) extensions to $\mathbb{R}^3$ (which are presented in the Appendix).

1.1 Previous Work

Our work is primarily motivated by a recent article by MIT CompGeom Group [8]. The article focuses on the problem statement for convex polygons, simple polygons, and wedges. That is, the authors found bounds on the minimum number of opaque, uniquely positioned guards needed to k-cover the different classes of polygons. Note that a polygon is *k-covered* if every point in the polygon is visible to at least k guards. In determining the bounds, the article introduces several important concepts such as dark rays and dark points, which will be reviewed below. The authors showed that exactly k guards are needed to k-cover a convex polygon of n vertices, or a convex n-gon, when $k \leq n$; at least $k + 1$ guards are necessary when $k > n$; and it is always possible to k-cover a convex polygon with $k + 2$ guards. Using a method similar to the standard Art Gallery problem, they prove that $(k + 2)\lfloor n/3 \rfloor$ guards are sometimes necessary and always sufficient to guard a simple n-gon. Specifically, occasional necessity is shown through a comb construction, and universal sufficiency is shown through 3-coloring and using arcs of points around the minimum size color set. However, as the authors note, the bound is not tight and cannot easily make use of the concepts used to prove bounds on convex polygons. Additionally, the article does not investigate any specialized polygons beyond convex polygons, such as star-shaped and monotone polygons. The article also presents exact bounds on the number of guards required to k-cover a wedge, noting that while the dark rays and points are useful, they cannot be forced to be external as in the case for $k \leq n$ in convex n-gons. MIT CompGeom Group does not explicitly provide a formal algorithm and time complexity for placing k guards. However, their use of line arrangements in the construction implies that the running time would be $\Omega(k^2)$. In contrast, our algorithm takes $O(k)$ time.

There are other results on k-guarding of polygons [1, 2, 4, 7, 10]. However, they differ from our problem primarily by allowing transparent guards, hence avoiding the challenges of handling opaque guards.

1.2 Preliminaries

Dark Points, and Dark Rays A point, ray, or region is said to be *dark* if it cannot be seen by one or more guards placed inside a polygon due to obstruction by other opaque guards, or obstacles. More specifically, a *d-dark point* is a point that cannot be seen by d guards due to other guards blocking their line of sight [8]. Similarly, a *dark ray* is an open half-line of dark points emanating from a guard because it blocks the view of another guard [8]. Dark points and rays are important as they dictate how many guards are required to make up for the lost view to get the full k-guarding. The following useful observation is given in [8] "k-guarding [a convex polygon] with $k + 1$ guards is possible if and only if there is no 2-dark point inside" the polygon.

Polynomial Curves A Bézier curve is a representation of a polynomial curve in the following form:

$$B(t) = \sum_{i=0}^{n} P_i B_i^n(t)$$

where P_i are control points, $B_i^n(t) = \binom{n}{i} t^i (1 - t)^{n-i}$ are Bernstein polynomials [6], and $t \in [0, 1]$. It can also be constructed using de Casteljau's algorithm [6]. De Castelau's

*University of Florida.{g.aldous,barber.seth,ungor}@ufl.edu

algorithm determines the value of the Bézier curve defined by control points $P_0, P_1, \cdots, P_n$ at a value t by computing $(1-t)P_j + tP_{j+1}$ for each pair of adjacent control points to get a new set of control points $P_0^1, P_1^1, \cdots P_{n-1}^1$ [6]. The process is then repeated with the new set of control points until there is only one control point left [6]. In this paper, we work with quadratic Bézier curves that have only three control points. So, we first calculate the points $P_0^1 = (1-t)P_0 + tP_1$ and $P_1^1 = (1-t)P_1 + tP_2$. Then, we compute $P_0^2 = (1-t)P_0^1 + tP_1^1$, which is the value of the Bézier curve defined by control points $P_0, P_1,$ and P_2 at the value t.

The use of Bézier curves was motivated by a desire to control the locations where $(k > 2)$-dark points exist (see Lemmas 2 and 3). By restricting the locations where these points exist, it is possible to extend guarding convex polygons to further domains, such as star-shaped polygons. Other placement strategies, such as perturbing guards around a single point or placement of guards along a line, do not provide the same provable restrictions. Bézier curves specifically were chosen due to the numerical stability of de Castelau's algorithm; however, other curves may also be able to provide similar guarantees.

Bézier curves importantly allow for *tangents* and *secants* with similar definitions to their counterparts on a circle. The tangents to the curve at the beginning and end point lie in the direction of the vector from the first control point to the second control point and the vector from the penultimate control point to the final control point, respectively [6]. Another useful property of Bézier curves is that the entire curve is contained within the convex hull of the control points [6].

Star-Shaped Polygons and Polyhedra A *star-shaped polygon* has a *kernel*—a convex set of points each visible to the entire polygon—which can be computed in linear time [5, 9]. Degenerate cases have a zero or one dimensional kernel; non-degenerate ones have a two dimensional kernel. A *true star polygon* is formed by extending the edges of a convex polygon until they intersect (e.g., a pentagram [3]). These definitions extend naturally to polyhedra, where a *star-shaped polyhedron* has a kernel visible to the entire shape and a true star polyhedron extends the facets of a convex polygon until they intersect.

2 Our Results

We first present a new algorithm for k-guarding convex polygons, followed by various bounds for other special domains.

2.1 A Linear-Time Algorithm for k-Guarding Convex Polygons with $k + 2$ Guards.

Our main result is Theorem 1, which is largely based on the properties of quadratic Bézier curves, De Casteljau's algorithm, and the concept of dark rays. Proof of Theorem 1 relies on the following Lemmas 2 and 3.

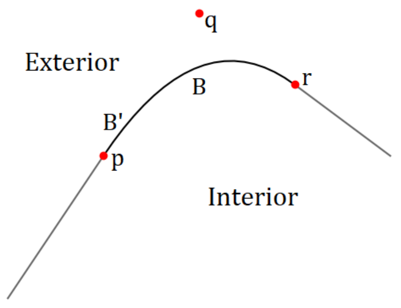


Figure 1: Extending the curve B at its endpoints with tangent lines gives a clear separation of interior and exterior of B'

In preparation for the main result, take a non-degenerate quadratic Bézier curve $B(t) : [0, 1] \mapsto \mathbb{R}^2$ defined by three non-co-linear control points. Extend B to B' by adding a ray to each endpoint of B such that B' is continuous as shown in Figure 1. The line B' divides the plane into two regions: A concave region and a convex region, respectively termed the interior and exterior of the curve. Additionally, observe that dark rays that share a guard in common do not intersect, as the intersection point of the corresponding lines wold be the common guard and the guards involved in generating dark rays are not included in those dark rays.

Theorem 1 *Every convex polygon can be k-covered with non-co-located, opaque guards in $O(k)$ time.*

Lemma 2 *The intersection of two dark rays formed by placing guards along a quadratic Bézier curve between $t = 0$ and $t = 1$, if such an intersection exists, will always be located in the region 'exterior' to the Bézier curve.*

Lemma 3 *Any three secants defined by points on a quadratic Bézier curve $B(t)$, where each point corresponds to a parameter value $t_i = 1 - \frac{1}{2^i}$, may all intersect at only one common point which must be one of the points in $B(t_i)$.*

Proof. (of Lemma 2) Consider the line connecting any pair of guards located on B' and assume B' is not a line. The dark rays associated with these two points will lie on this line. As the interior is convex, the portion of the line in the interior is visible to both guards and is not involved in any dark rays. Thus, there are no dark rays, or dark ray intersections, in the interior or on B'. Therefore, any dark ray extensions must be located in the exterior of B'. □

Proof. (of Lemma 3) Let p, q, and r be non-linear control points of B, and let B be defined by the parametric function $B(t)$, defined as:

$$B(t) = (B_x(t), B_y(t))$$
$$B_x(t) := (1-t)^2 p_x + 2(1-t)tq_x + t^2 r_x$$
$$B_y(t) := (1-t)^2 p_y + 2(1-t)tq_y + t^2 r_y$$

Consider the following sequence:

$$t_i = 1 - \frac{1}{2^{i-1}}, \quad \text{for } i \geq 1$$

Each value in the sequence t_i is in the domain of $B(t)$ and can be used to generate a point on B, $g_i = \left(B_x(t_i), B_y(t_i)\right)$ for all i. Each pair of points in g_i defines a secant to the curve.

We use induction on the number of guards generated, where subsequent guards are given the next integer value of i. For $i = 3$ (base case with three guards) the claim is trivially true as the arrangement of the three points forms a triangle. The secants formed cannot possibly intersect except at the points g_1, g_2, g_3.

Assume that the lemma is true for all $j < i + 1$. Adding a new point g_{i+1} creates i new secants. Assume, that a point s exists where 3 different secants intersect and $s \notin t$. There are three possible cases:

1. None of the 3 secants intersecting at s were created when adding point g_{i+1}, contradicting the inductive assumption.

2. 1 or 2 of the 3 secants that intersect in s were created when adding point g_{i+1}. Recall that the guards are placed incrementally at positions of the form $t_i = 1/2^i$ along a Bézier curve. Due to this method of selection and the strict concaveness of the Bézier curve, the dark rays emanating from each guard fan out in a predictable manner. Due to this fanning out, each consecutive dark-ray intersection on a certain dark ray is further along the dark ray than the previous one, or, in other words, the dark rays do not intersect. Thus, the geometry implies that there are 3 different intersection points . Therefore, all three secants never intersect at the same location, meaning s cannot exist.

3. The 3 secants that intersect at s were created when adding the point g_{i+1}. This implies that all three secants intersect at the point g_{i+1}, and since two different lines can intersect at most once, this means that point s cannot exist.

As all three cases lead to a contradiction, the inductive assumption is valid by contradiction. As the base case is also valid, the lemma holds by induction.

□

Using Lemmas 2 and 3, it is straightforward to see that with k guards placed on a Bézier curve, it is possible to k-guard the interior of the Bézier curve. However, the dark rays and at worst 2-dark points in the exterior mean that the exterior will be $(k - 2)$-covered. Algorithm 1 utilizes this strategy to find $k + 2$ guards to k-cover any convex polygon in $O(k)$ time. Figure 2 illustrates an output of the algorithm.

Algorithm 1 k-guard a convex polygon with $k + 2$ guards

```
1: procedure GuardConvexPolygon($p$, $q$, $r$ : non-collinear vertices of convex polygon $C$)
2:     Use $p, q, r$ as control points to construct Bézier curve $B(t) : [0, 1] \to \mathbb{R}^2$
3:     for $i := 0 \ldots k + 1$ do
4:         Place a guard at $B(1 - 2^{-i})$ using De Casteljau's algorithm
5:     end for
6:     return the locations of the $k + 2$ guards.
7: end procedure
```

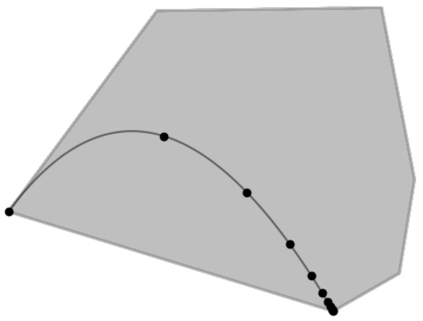


Figure 2: A hexagon 10-guarded with 12 guards on a curve.

Proof of Correctness. By Lemma 3, the $k+2$ guards will generate at worst 2-dark points located at the intersections of the dark rays generated by the guards (because the dark rays are the portions of a secant within the exterior region of the Bézier curve). As there are $k + 2$ guards, such points will be visible to k guards. Thus, all points in the polygon are visible to at least k guards, by virtue of which the shape is k-covered.

Complexity Analysis. Calculating the positions of the $k + 2$ guards placed along the Bézier curve involves one application of De Casteljau's algorithm per guard. For a quadratic Bézier curve, this operation requires $O(dn^2) = O\left(2\left(3^2\right)\right) = O(1)$ time complexity, where d is the degree of the Bézier curve. Therefore, calculating the guard positions takes $O(k + 2) = O(k)$ time.

Note that this time complexity is an improvement on the time complexity of the constructive algorithm implied in [8] discussed previously.

Use of Bézier Curves. Bézier curves were chosen because selecting control points from a convex polygon guarantees the curve remains inside it. Experimenting with other curves with similar properties could inform future work.

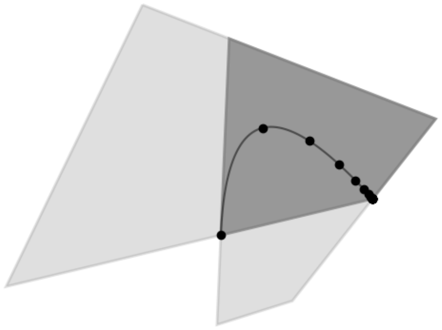


Figure 3: A star-shaped polygon 20-guarded with 22 guards on a curve inside the kernel.

2.2 Guarding Star-Shaped Polygons

Many of the concepts introduced for guarding convex polygons can be modified to apply to guarding star-shaped polygons. For example, the algorithm to place $k + 2$ guards in a convex polygon can be extended to guard non-degenerate star-shaped polygons by guarding the polygon's kernel using the previous algorithm. As the guards may see all the polygon except when blocked by another guard and there are at worst 2-dark points, the $k + 2$ guards produced k-cover the non-degenerate star-shaped polygon (see Figure 3). The total time complexity of placing $k + 2$ guards to k-cover a a star-shaped n-gon is $O(n + k)$. This time complexity is composed of the time to calculate the kernel, which can be linear in the number of sides n [5], and the time to place the $k + 2$ guards, which is linear in k.

It is possible to k-cover polygons using fewer than $k + 2$ guards in certain cases. In general:

Theorem 4 *Let P be a non-degenerate star-shaped polygon with a kernel of m vertices. Then, to k-cover P with g guards:*

1. *When $k \leq 1$, $g = k$ guards are always necessary and sufficient.*
2. *When $2 \leq k \leq 3$, $g = k + 1$ guards are sometimes necessary and always sufficient.*
3. *When $k > 3$, $g = k + 2$ guards are sometimes necessary and always sufficient.*

Proof. Sufficiency and necessity for the first case are trivial. The fact that 1 is the upper bound for $g = k$ is demonstrated by the shape in Figure 4, where all guards must be placed in the kernel to k-cover with k guards, but any two guards generate interior dark rays.

For the second case, observe that four guards may be arranged as in Figure 5 and placed in the kernel. As the guards are in the kernel, they can see all areas of the polygon except for some limited visibility in dark rays. As there are no 2-dark points, $g = 4 = k + 1$ guards are sufficient when $k \leq 3$.

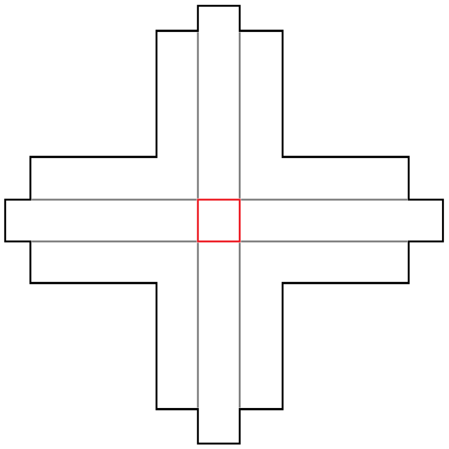


Figure 4: Kernel (red) of a polygon P not overlapping ∂P

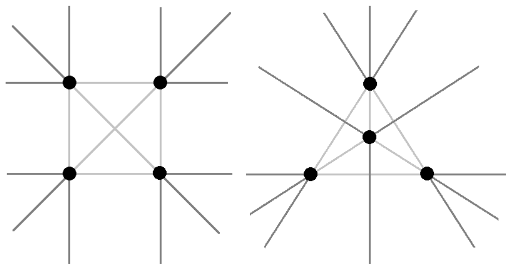


Figure 5: Two general arrangements of four points. The dark gray lines are the dark rays, while the light gray lines are the non-dark portion of the lines the dark rays lie on.

Necessity for the second case follows from the first case. To see why 4 is the upper bound in this case, note that Figure 5 shows possible arrangement classes for 4 guards. Observe that any additional guard added to either of these configurations will cause dark ray intersections. These intersections need to be exterior to the polygon. Observe also that the guard must be placed in the kernel and that P can be made arbitrarily large without increasing the kernel size by, for example, fattening the arms of the cross in Figure 5, but not the nubs sticking out. Thus, the dark ray intersections caused by the new guard cannot always be located outside P, and a fifth guard may not always be added to attain a k-guarding with $k + 1$ guards, completing the proof. (The above result is similar to the Triangle Lemma [8].)

For the last case, sufficiency of $k + 2$ guards is shown previously. Occasional necessity follows from the previous case. □

2.2.1 Bounds on k-Guarding the Plane

Much of the reasoning that was applied to guarding arbitrary star-shaped polygons applies directly to the plane. However, the bounds are now always rather than sometimes necessary, as it is impossible for dark ray intersections to occur outside the plane. This result is given as a corollary to Theorem 4:

Corollary 4.1 *To k-cover the plane $\mathbb{R}^2$ with g guards:*

1. *When $k \leq 1$, $g = k$ guards are necessary and sufficient.*
2. *When $2 \leq k < 4$, $g = k + 1$ guards are necessary and sufficient.*
3. *When $k \geq 4$, $g = k + 2$ guards are necessary and sufficient.*

2.2.2 Bounds on k-Guarding True Stars

Note that all convex polygons are star-shaped polygons. Thus, the bounds provided in [8] can be considered as bounds on special star-shaped polygon. More specifically, they are the best possible bounds for star-shaped polygons as convex polygons do not have the problems associated with the possibility of 2-dark points being formed inside the polygon but outside the kernel. True stars can be considered as intermediary between convex polygons, where the kernel is the polygon, and star-shaped polygons, where the kernel is strictly interior to the polygon. Specifically, as it is possible to force some dark ray intersections to lie outside the polygon, the bounds from Theorem 4 can be improved to the following:

Theorem 5 *Let P be a true star with an m-gon kernel, $m > 4$. Then, to k-cover P with g guards:*

1. *When $k \leq \lfloor m/2 \rfloor$, $g = k$ guards are necessary and sufficient.*
2. *When $\lfloor m/2 \rfloor < k < m$, $g = k + 1$ guards are necessary and sufficient.*
3. *When $k \geq m$, $g = k + 2$ guards are necessary and sufficient.*

Proof. For the first case, observe that to use more than two guards in k-covering with k guards the guards must be positioned so that the dark rays that they generate are strictly exterior to P. Thus the guards must be located on the kernel vertices, which are the star's reflex vertices. If every vertex has a guard, there will be dark rays running along the sides of the star's limbs. Thus, an additional restriction is applied that at most every other kernel vertex may have a guard. After this restriction, note that no dark rays can be formed internal to the polygon due to the fact that the central m-gon is convex. Thus, when $k \leq \lfloor m/2 \rfloor$, $g = k$ guards are sufficient and the implied bound is the upper bound for this case. As it is impossible to k-cover a polygon with fewer than k guards, this number of guards is also necessary, completing the proof of this part.

Necessity for the second case follows from the bound in the first case being maximal. To determine maximum sufficiency for the second case, consider guard placements that generate interior dark rays but no interior dark ray intersections, which are the required conditions for k-covering with $k + 1$ guards.

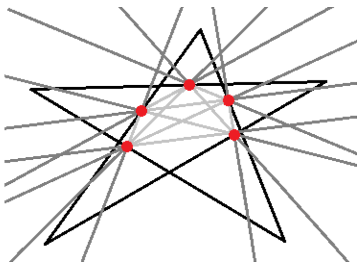


Figure 6: 4-guarding a true star with $m = 5$ using 5 guards.

Note that placing every guard on a vertex creates 2-dark points at the tips of the star limbs for $m > 3$, and is thus unsuitable. A better option is to place guards on the m sides of the kernel. These generate dark rays that are internal to the star limbs and thus do not intersect. Thus, it is possible to $(m - 1)$-cover a polygon with m guards. The question then becomes whether it is possible to add another guard or to come up with a different arrangement of guards that can be augmented. There are two potential methods:

- The first method is to add another guard to the current guarding, such as in Figure 6, to the kernel. Adding an additional guard will cause a dark ray running along at least one of the star limbs, which will intersect another dark ray, so this potential augmentation is impossible.
- The second method is to create a different arrangement of m guards that may be augmented. If these m guards form a convex m-gon, then the problem devolves to a case similar to the previous case and the reasoning holds. Otherwise, then the case devolves to guarding a star with a lower value of m that can be drawn inside the current star. By induction with base cases like Figure 5, this guarding will not work for either star. Thus this case is impossible, completing the proof.

For the third case, the necessity of $k + 2$ guards follows from previous reasoning. Sufficiency is proven earlier, completing the proof. □

2.3 Guarding Unbounded Convex Polygons

Removing one (or more) edges of a convex polygon and extend the resulting unbounded to infinity such that they never intersect gives us *unbounded convex polygons* (See Figure 7 and the wedge in [8]). We show the following theorem:

Theorem 6 *Let P be an unbounded polygon with m sides such that the unbounded sides are not parallel. To k-cover the polygon with g guards:*

- *When $k \leq m$, $g = k$ guards are necessary and sufficient*
- *When $m < k < 3m + 4$, $g = k + 1$ guards are necessary and sufficient.*
- *When $3m + 4 \leq k < 4(m + 1) - 2 = 4m + 2$, $g = k + 2$ guards are sufficient*

- *When $k \geq 4m + 2$, $g = k + 2$ guards are necessary and sufficient.*

Proof. The first case follows from the fact that, to k-cover a convex polygon with k guards, all guards must be on the boundary and there may be at most one guard per edge [8]. The maximum number of such guards is achieved by placing the guards on the edge interiors.

The second case is due to a construction given below.

The third case follows from the second and fourth cases.

The fourth case follows from the fact that $k \geq 4n - 2$ guards are needed to guard a bounded convex polygon with n sides [8]. If could do better than $4m + 2$ for P, then could do better than this bound for a convex n-gon by guarding P and adding a side, as for wedges [8]. □

Previous methods used in reasoning about k-covering a convex polygon with $k+1$ guards do not transfer directly over to unbounded convex polygons [8]. The following method, allows for the placement of a total of $3m+4$ guards with only 1-dark points in P's interior, resulting in a $3m + 3$ guarding.

Guard Stacks Create stacks of three co-linear points, each effectively creating a line terminating in 2-dark rays . Place a stack on each edge such that the base of the stack is on the edge's interior and the topmost guard interior to the polygon so that the mentioned line is almost parallel to the edge, as in Figure 7. Perturb the middle point slightly in from the convex hull of the guards. The guards may be located close enough to the wall such that any dark ray intersections caused by other guards will be exterior to the polygon. Perturbing the middle guard causes the dark rays with respect to the stack to be basically the same as the line through the originally co-linear points, but not 2-dark. Note that the dark rays do not pass underneath other guards.

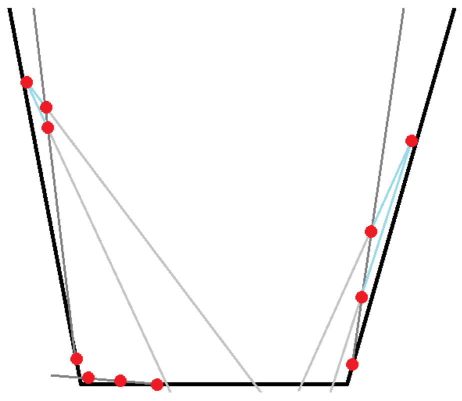
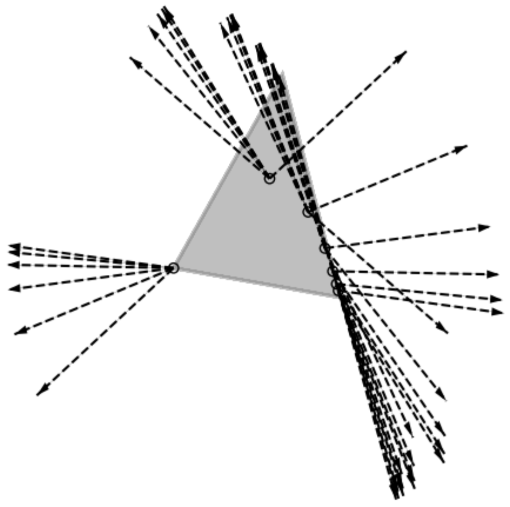


Figure 7: Partial guarding of an unbounded convex polygon illustrating the guards (red), relevant dark rays (gray), and (blue) segments generating the dark rays blue. For full guarding, there would be two additional guards very high up creating dark rays interior to the polygon along with one other dark ray intersecting the bottom of the polygon.



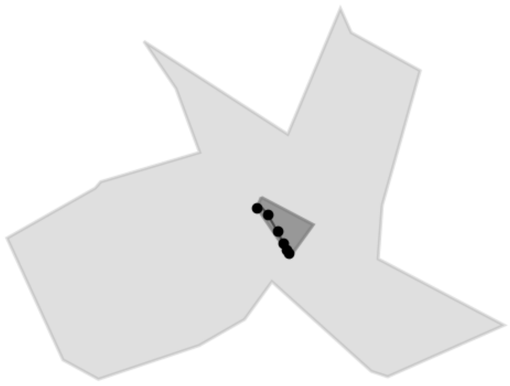


Figure 8: Sample output of our implementation.

Four Additional Guards After positioning the previous guards, place two more guards above the guards currently in the polygon and below the uppermost dark rays. These guards may be placed high enough up so the dark rays generated by all except the uppermost four guards on the unbounded sides do not have interior intersections Additionally, the guards can be placed so that the dark rays they generate with the aforementioned four guards exit the polygon before causing a dark-ray intersection, as in Figure 7. An additional two guards may be placed high enough up so the dark rays emanating from them are at worst parallel to and non-intersecting with the other dark rays and the generated dark rays emanating from the other guards are nearly vertical and do not cause dark ray intersections interior to P. An example of these two points can be seen in Figure 15 of [8]. The addition of these guards results in a $3m + 3$-guarding with $3m + 4$ guards.

References

[1] P. Belleville, P. Bose, J. Czyzowicz, J. Urrutia, and J. Zaks. K-guarding polygons on the plane. In *Proc. of the Canadian Conference on Computational Geometry*, pages 381–386, 1994.

[2] D. Busto, W. S. Evans, and D. G. Kirkpatrick. On k-guarding polygons. In *Proc. of the 25th Canadian Conference on Computational Geometry*, 2013.

[3] B. Grünbaum and G. Shephard. *Tilings and patterns*. 1987.

[4] G. Kalai and J. Matoušek. Guarding galleries where every point sees a large area. *Israel Journal of Mathematics*, (1):125–139, 12 1997.

[5] D. T. Lee and F. P. Preparata. An optimal algorithm for finding the kernel of a polygon. *J. ACM*, 26(3):415–421, July 1979.

[6] S. Mann. *Blossoming Development of Splines*. Springer, 2006.

[7] T. S. Michael and V. Pinciu. Multiply guarded guards in orthogonal art galleries. In *Computational Science - ICCS*, volume 2073, pages 753–762. Springer, 2001.

[8] MIT CompGeom Group, H. A. Akitaya, E. D. Demaine, A. Hesterberg, A. Lubiw, J. Lynch, J. O'Rourke, and F. Stock. Super guarding and dark rays in art galleries. In *Proc. of the 35th Canadian Conference on Computational Geometry*, 2023.

[9] J. O'Rourke. *Art gallery theorems and algorithms*. Oxford University Press, 1987.

[10] I. Salleh. K-vertex guarding simple polygons. *Computational Geometry*, 42(4):352–361, 2009.

Appendix

A Extension to Shapes in $\mathbb{R}^3$

The definitions of dark points, dark rays, and guards, naturally expand to $\mathbb{R}^3$, with the modification that guards have a spherical, rather than a circular, field of view. A more important change is how many dark points from a guarding is sometimes necessary. In two-dimensions, 2-dark points may sometimes be unavoidable. In three dimensions however, the following holds:

Lemma 7 *Any collection of guards in three dimensions may be perturbed so that there are only 1-dark points.*

Proof. Create or perturb an arbitrary arrangement of guards such that there are only 2-dark points and no 2-dark rays. If there is a 2-dark point, it is the result of a dark ray intersection. Consider perturbing one of the guards involved in the 2-dark intersection perpendicular to the plane defined by the intersecting dark rays. The intersection will become separated and cease to exist. The perturbation may be made small or large enough so that no additional dark ray intersections are created. Thus, any 2-dark point may be removed by perturbations without creating additional 2-dark points. Repetition leaves only non-intersecting 1-dark rays completing the proof. □

Note that, as dark rays are unavoidable, the darkness level in Lemma 7 is the best possible for a collection of guards in $\mathbb{R}^3$.

A.1 Guarding Convex Polyhedra

The expansion of convex polygons to three dimensions is convex polyhedra. In this case, the following can be shown:

Theorem 8 *Let P be a convex polyhedron with e edges. Then*

- *When $k \leq e$, $g = k$ guards are necessary and sufficient to k-cover P*
- *When $k > e$, $g = k + 1$ guards are necessary and sufficient to k-cover P*

Proof. The first case follows from the fact that when $g > 1$, any guard interior to the polyhedron generates a dark ray. Thus, to k-cover with k guards, all guards must be located on P's boundary. Observe that, to avoid dark rays in P:

- If a guard is located in the interior of a facet, no guard may be located on the facet's interior or exterior.
- If a guard is located on a vertex, no guard may be located on the incident edges.
- If a guard is located on the interior of an edge, no other guard may be placed on the edge.

The observations divide up the potential guarding placements into three main patterns: One guard per facet, one guard per vertex, or one guard per edge. Let f, v, and e be the numbers of facets, vertices, and edges, respectively, in P. Then, as an edge borders at most two facets and a facet is bounded by at least three edges:

$$2e \geq 3f \Rightarrow e > f$$

Additionally, by Euler's formula:

$$e = v + f - 2 \Rightarrow e \geq v + \frac{2}{3}e - 2 \Rightarrow e \geq 3v - 6$$
$$\because v > 3 \therefore e > v$$

Thus, the number of edges is always greater than the number of facets and the number of vertices. As placing any vertex or facet guard necessitates the omission of at least two edge guards, placing guards on the interiors of the edges is the optimal method of k-covering with $g = k$ guards. This fact proves sufficiency for $k \leq e$ and that e is the upper bound for sufficiency. Necessity for this case is obvious, completing the proof of the first case.

The previous case proved necessity of $g = k + 1$ guards when $k > e$, so only sufficiency remains. By Lemma 7, any collection of $k + 1$ guards may be perturbed so that there are at worst 1-dark points, or the polyhedron is k-covered. Thus, sufficiency holds. □

A.2 Guarding Star-Shaped Polyhedra

Guarding star-shaped polyhedra follows naturally from guarding convex polyhedra and star-shaped polygons. The following specific results are easily shown.

Theorem 9 *To k-cover an arbitrary star-shaped polyhedron with a non-degenerate kernel with g guards:*

- *When $k \leq 1$, $g = k$ guards are necessary and sufficient*
- *When $k > 1$, $g = k + 1$ guards are sometimes necessary and always sufficient.*

Proof. The first case follows similar reasoning to that given for Theorem 4. That is, consider a three-dimensional version of the modified cross in Figure 4. All guards must be located in the kernel to k-cover with k guards, but any two guards generate dark rays interior to the polyhedron. Thus, the bound is necessary and sufficient. Sufficiency in the second case follows from Lemma 7 and from the fact that any point in the kernel can see all points in the polygon. Occasional necessity follows from a construction similar to Figure 4. □

As was the case with star-shaped polygons, the kernel being strictly interior to the polyhedron represents a worst-case. True star polyhedra represent a potentially more typical case, and the following theorem is easily shown:

Theorem 10 *To k-cover a true star polyhedron with a kernel of n vertices:*

1. *When $k \leq \lfloor n/2 \rfloor$, $g = k$ guards are necessary and sometimes sufficient, while $g = k + 1$ guards are always sufficient.*
2. *When $k > \lfloor n/2 \rfloor$, $g = k + 1$ guards are necessary and sufficient.*

Proof. The first case arises from the fact that when there is a guard not on an edge or vertex of the kernel, it will generate an interior dark ray in the presence of any other guards. Observe that when a guard is placed on an edge, no guards may be placed on an edge or vertex of an incident face or vertex, else a dark ray running along the border of the polyhedron will emanate from the original guard. On the other hand consider when a guard is placed instead on a vertex. That guard now only generates a dark ray interior to the polyhedron when a guard is placed at a vertex adjacent to the guard. In other words, there may be multiple guards per kernel facet and there will be on average at least one guard per kernel facet. Since adding edge guards would imply forgoing at least two vertex guards, the optimal strategy of k-covering a star-shaped polygon with k guards is to place the guards on non-adjacent kernel vertices.

The number of guards located at non-adjacent kernel vertices depends on the number of kernel vertices and how they are connected. However, observe that in the best case this guarding results in the placement of at most $\lfloor n/2 \rfloor$ guards, completing the proof of the first part of the first case.

The second part of the first case and the second case follow from Lemma 7. □

A.3 Guarding Three-Dimensional Space

The rationale and proof behind guarding arbitrary star-shaped polyhedra directly translate to guarding three-dimensional space:

Corollary 9.1 *To k-cover $\mathbb{R}^3$ space with g guards:*

1. *When $k \leq 1$, $g = k$ guards are necessary and sufficient*
2. *When $k > 1$, $g = k + 1$ guards are necessary and sufficient*

The VC-Dimension of Limited Visibility on the Boundary of a Simple Polygon

Matt Gibson-Lopez[*] Erik Krohn[†] Zhongxiu Yang[‡]

Abstract

In the classic art gallery problem, each point is assumed to be able to see "infinitely far"; two points see each other as long as the line segment connecting them stays inside the polygon regardless of how far apart the points are. We consider a *limited visibility* variant of the art gallery problem where each point has a sight radius ρ. That is, p sees q if and only if the line segment $\overline{pq}$ stays inside the polygon and the distance between p and q is at most ρ.

In this paper, we consider the VC-dimension of visibility on the boundary of simple polygons under this limited visibility model. That is, we restrict all guards/viewpoints to be on the boundary of the simple polygon. It has been previously shown that the VC-dimension of this problem *without* limited visibility is exactly 6 [11]. And we show that the VC-dimension under the limited visibility model is exactly 8. That is, we give an example of a simple polygon with 8 "guards" on the boundary such that every one of the 2^8 subsets of the 8 guards has a "viewpoint" that sees exactly the guards in the subset. We then show that this is not possible with any set of 9 guards.

1 Introduction

The *art gallery problem* is one of the best-known geometric set cover problems in computational geometry, which aims to find the minimum number of "guards" points to see every point of the interior of a simple polygon. For any two points p and q inside the polygon P, the line segment $\overline{pq}$ is said to be pierced if $\overline{pq}$ intersects with the boundary of the polygon, and we also say that the points p and q are pierced. The typical way to model visibility in the art gallery problem is to say that p sees q if and only if $\overline{pq}$ is not pierced. The idea is that the polygon is modeling the interior of a building (e.g., an art gallery), and two points can see each other as long as there is no wall obstructing the vision of the two points. The geometric set cover problem with respect to this definition of visibility has been considered in many different contexts, for example simple polygons [1, 6, 7, 11, 14], monotone polygons [10, 12, 19–21], rectilinear polygons [20], staircase polygons [3, 22, 23], and terrains [2, 5, 8, 9, 16–18].

1.1 Limited Visibility Model

A potential drawback of the classic art gallery problem for various applications is the assumption that two points can see each other no matter how far apart they are as long as their line segment is not pierced. In this paper, we consider a *limited visibility* variant of the art gallery problem with simple polygons similar to the variant considered in [13] for terrains. Here, each point has a sight radius ρ and cannot see any points that are outside of its sight radius. We assume each point has the same sight radius, and two points p_i and p_j inside the polygon see each other if and only if (1) $\overline{p_i p_j}$ is not pierced and (2) $d(p_i, p_j) \leq \rho$, where $d(p_i, p_j)$ is the Euclidean distance between p_i and p_j. We denote $disk(p)$ to be a disk of radius ρ centered at point p. Clearly, p cannot see any points outside of $disk(p)$. We call an instance of this problem an LVP for *limited visibility polygon*.

1.2 VC-Dimension

The *VC-dimension* is a metric used to measure the complexity of a set system. Given an LVP P, we say that a finite set of "guards" G is *shattered* if for every subset $G' \subseteq G$, there exists some "viewpoint" v in P that sees every point in G' and does not see any point in $G \setminus G'$. In this context, the VC-dimension is the maximum number of guards you can shatter in P. Note that in the limited visibility model, we assume the sight radius constraint applies to all vertices on the boundary of the polygon regardless of whether or not they are a guard or a viewpoint. Brönnimann and Goodrich [4] give a polynomial-time $O(\log OPT)$-approximation algorithm for any set system with constant VC-dimension where OPT is the size of an optimal cover, hence the interest in computing the VC-dimension for various set systems. For example, showing the VC-dimension of LVPs as we have defined above implies that there is an $O(\log OPT)$-approximation for vertex guarding a LVP.

The VC-dimension for the classic art gallery problem has been considered in various settings depending on where the guards/viewpoints are allowed to be located. When the guards and viewpoints are restricted to the boundary of the polygon, it is known that the

[*]The University of Texas at San Antonio, matthew.gibson@utsa.edu
[†]University of Wisconsin - Oshkosh, krohne@uwosh.edu
[‡]Millersville University of Pennsylvania, zhongxiu.yang@millersville.edu

VC-dimension is exactly 6 [11, 12]. When the guards and the viewpoints are allowed to be in the interior of the polygon, the problem has proven to be much more difficult. In fact, a tight bound is not known for this problem. In 1998, Valtr showed that the VC-dimension is at least 6 [24], and in 2014 Gilbers and Klein showed that it is at most 14 [15]. It is generally believed that the lower bound of 6 is likely closer to the correct result than the upper bound of 14, but the tight result has remained elusive.

In the limited visibility model, we would certainly expect the VC-dimension to increase, as the classic variants of these problems can be seen as a special case of the limited visibility variant with the sight radius set to be sufficiently large. In the case of terrains, King showed that the VC-dimension of the classic terrain guarding problem is exactly 4 [17], and in the limited visibility variant Gibson-Lopez and Yang showed that the VC-dimension is exactly 7 [13].

1.3 Our Results

In this paper, we consider the VC-dimension problem on LVPs when all of the guards and viewpoints are restricted to being on the boundary of the polygon. Note that by adding a "ceiling" to the limited visibility terrain result from [13], one can trivially obtain a lower bound of 7 for this problem. In this paper, we will show that the tight bound is actually 8.

Theorem 1 *The VC-Dimension of visibility on the boundary of an LVP is 8.*

As far as we are aware, this is the largest *constant* tight bound proven for the VC-dimension of an art gallery variant. The proofs of tight upper bounds on the VC-dimension often are tedious case analyses even for smaller VC-dimensions such as in [11]. We use a computer program to show that it is not possible to shatter 9 guards. We prove a set of lemmas that all LVPs must satisfy in this paper, and then the computer program helps to verify that every way of ordering the guards/viewpoints around the boundary of an LVP must violate at least one of these lemmas.

2 Lower Bound

We show that the VC-dimension of LVPs is at least 8. We have the coordinates of the vertices of our LVP in Table 2, 3, 4, 5 in the appendix. In our polygon, a set of 8 guards is shattered for sight radius ρ of 1, and the guards that are shattered are the guards $G = \{g_A, g_B, \ldots, g_H\}$. The viewpoints $V = \{vp_{AB}, vp_{AC}, \ldots, vp_{ABCDEFGH}\}$ denote which guards of G the viewpoint sees, for example, the viewpoint vp_{AB} sees guard g_A and g_B, and does not see any other guards. Some of the vertices are neither a viewpoint nor a guard in G and are labeled "blocker" (their role is simply to block a viewpoint from seeing a vertex in G). The table does not contain a viewpoint for the empty set or for the subsets of size 1, since it is trivial to add these points. We wrote a computer program to formally verify our coordinates are correct, which is available at https://github.com/computational-geometry-lab/visibility-simple-polygon-lower-bound. Figure 1 shows the static figure of the LVP, where point A, B, …, H represents g_A, g_B, …, g_H. The dynamic version can be found here: https://www.geogebra.org/classic/a98kkt7x.

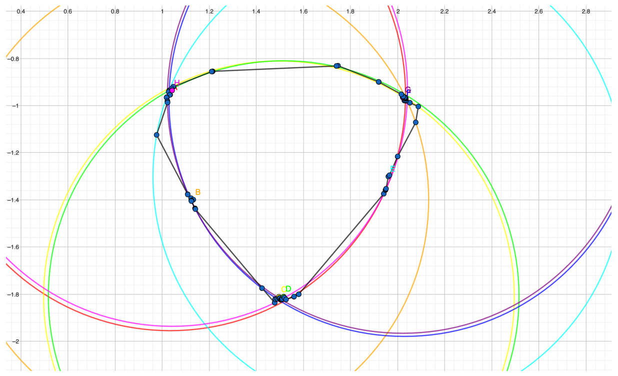


Figure 1: Polygon boundary. Guards A and H are near each other at the top left. Guard B is "alone" on the left. Guards C and D are near each other near the bottom. Guard E is "alone" on the right. Guards F and G are near each other at the top right.

3 Upper Bound

In this section, we show that the VC-dimension of LVPs is at most 8. We will begin with a high-level overview of our approach for showing that it is not possible to shatter 9 points on the boundary of an LVP with viewpoints that must also lie on the boundary. Our proof incorporates both lemmas with geometric proofs as well as a computer program that is used to show that there is no way to order the guards/viewpoints around the boundary of an LVP so that the viewpoint visibilities are correct without violating one of the lemmas. We begin with a high-level overview of the process, then give the lemmas, and finally, we go over the pseudocode of the computer program we used to complete the proof.

3.1 High Level Overview

3.1.1 Orderings

Assume for contradiction that there exists an LVP P with a set of points $G = \{g_0, \ldots g_8\}$ (which we call *guards*) and set V of 2^9 *viewpoints* on the boundary of P such that for any subset $S \subseteq G$, there is a point v in V such that v sees all of the guards in S and none of the guards in $G \setminus S$. Without loss of generality, we assume the indices of the guards are in counterclockwise order. Given P, we say the *ordering* O of P is the permutation of $G \cup V$ such that O is the counterclockwise ordering of the points in G and V. Without loss of generality, we assume g_8 is the last point of O, and therefore the first point in O is the first point in $G \cup V$ that is counterclockwise of g_8. Note that there may be other vertices of P that are not in G or V, and these vertices do not show up in O.

Given an LVP with the points G and V labeled, the computation of its ordering O is trivial. What we are mostly interested in is going in the other direction. That is, given an ordering O, we want to determine whether or not there exists an LVP P such that O is the ordering of P. If O does have such an LVP, then we say that O is *realizable*. Our strategy for showing that 9 guards cannot be shattered is to show that every ordering O of 9 guards and 2^9 viewpoints is not realizable.

3.1.2 Partial Orderings and Extensions

There are more than $(2^9)!$ orderings, and so our approach is to consider *partial orderings* O' which are subsequences of an ordering O that contains all points of G (O' can contain only a subset $V' \subseteq V$, but the order of V' must be the same as they occur in O). For a partial ordering O' of the points G and V', we can obtain an *extension* of O' by "inserting" a subset of viewpoints of $V \setminus V'$ into O'. That is, an extension E of O' is another partial ordering such that O' is a subsequence of E. We say that a partial ordering O' is realizable if there is an LVP P such that O' is the ordering of P. Note that if a partial ordering O' is not realizable, then all extensions of O' are also not realizable. A partial ordering that is not realizable and does not contain too many viewpoints can therefore be used to show many orderings are not realizable.

3.1.3 Strategy for Rejecting Orderings

If we can prove an ordering to not be realizable, then we say that we *reject* the ordering. If we prove a partial ordering is not realizable, then we reject all extensions of that partial ordering. To reject orderings, we first prove a set of lemmas that gives properties that must be satisfied by any realization of an LVP. The proofs of the lemmas are generally fairly simple geometric arguments. Ideally, we could directly show that a set of simple geometric lemmas is enough to reject all orderings, but unfortunately, we were not able to do this as there did not seem to be a small subset of viewpoints that we could use to reject all partial orderings that include that subset. VC-dimension upper bound proofs often entail a tedious case analysis involving a small subset of points, but unfortunately, that does not feel likely to be possible for this problem. Instead of performing a case analysis by hand for the cases that satisfy the lemmas, we perform a case analysis by computer program. The program starts with a partial ordering O', and then checks extensions of the partial ordering to see if there exists an extension of O' that does not violate any of the lemmas. If we can show that there does not exist an extension that satisfies all of the properties of the lemmas, then we can reject O'.

In our realization of the lower bound using eight guards, all of the guards see each other. This property intuitively seemed to be quite important to obtain the realization. We observed that if we enforced this property to be a constraint in our computer program considering nine guards, then we could reject the empty partial ordering (that contains only the guards and no viewpoints) considering only extensions that used viewpoints from a set of fifteen viewpoints. In other words, if you require that all guards see each other, there is a set of fifteen viewpoints that cannot be placed in an LVP simultaneously. Moreover, this property holds even if we require that the guards cannot be pierced (we allow guards to not see each other but only because they are too far from each other). Unfortunately, if we allow just a single pair of guards to be pierced, then the fifteen viewpoints have a realization, and it seemed that we needed to consider many more viewpoints to be able to obtain a rejection of the empty partial ordering. Moreover, if we change which guards are pierced, the set of viewpoints needed to obtain a rejection also changed dramatically. This led us to believe that even with the use of a computer program, a large case analysis (using more cases with more assumptions about the structure of the cases) is needed to keep the running time of the program to be on the order of weeks as opposed to on the order of years. With fewer cases incorporating fewer assumptions, we observed that our program would generate the same subcases repeatedly.

3.1.4 Cases

We consider cases in the form where for each pair of guards, we specify whether the two guards should 1) not be pierced, 2) be pierced, or 3) we do not care if they are pierced or not. We have 36 pairs of guards, which we partition into four groups of 9 pairs. The first group we call *one-gap* pairs. These are pairs of the form g_i and g_{i+1}. The second group we call *two-gap* pairs. These are pairs of the form g_i and g_{i+2}. The third group we call *triangle* pairs. These pairs are of the form g_i and g_{i+3}. We call them triangle pairs as the guards g_i, g_{i+3}, and g_{i+6} forms a "triangle" of these pairs. The final group we call *three-four* pairs, and they are of the form

g_i and g_{i+4} (these pairs have three guards on one side of the polygon between them and they have four guards on the other side between them). All indices are taken modulo 9. Our intuition was that it was probably a bad idea to allow triangle pairs and three-four pairs to be pierced. If two guards g_i and g_j "across" the polygon from each other are such that $\overline{g_i g_j}$ is pierced, then you are dramatically restricting where on the boundary of the polygon the guards can see, and intuitively this should not lead to a realizable ordering that shatters the nine guards. We hoped that by setting some of these pairs to have to be pierced from each other and making the majority of the other pairs to be don't care, we could eliminate many of the cases with this one more general case. What we discovered was that our code would "cycle" the one-gap and two-gap pairs over and over again. That is, when considering the location of a viewpoint v in a partial ordering, we eventually would get a rejection after trying various combinations of piercing/not piercing the one-gap and two-gap pairs. Then we would try v in a different location in a partial ordering, and we would then repeat the same combinations for the one-gap and two-gap pairs that we had just considered prior to rejecting again. This cycling produced enormous running times (likely would have taken years for most of the single cases with limited assumptions).

We then turned our focus on cases where we specified whether we want pierced or not pierced for all of the one-gap and triangle pairs. By specifying in the case what the one-gap pairs should be, we can prevent much of this cycling, and by specifying the triangle pairs, we get a majority of cases that can be rejected in under a few hours. For any cases that seemed to be taking too long to terminate, we then broke those cases up into subcases by specifying pierced or not pierced for the two-gap pairs, as we observed that for most cases that were slow to reject, we were still getting the cycling on the two-gap pairs even with the fixed choices for the one-gap pairs. The majority of these were rejected very quickly but there were a few pesky cases that remained. We observed that we would get a different version of cycling: not with respect to pierced or not pierced for the guards but cycling with respect to viewpoint placement in the partial order that we were considering. For example, maybe there were viewpoints vp_1, vp_2, vp_3, and vp_4. We might try one location for vp_1, then the other three viewpoints would iterate through a set of configurations prior to reaching a rejection. Then we try another location for vp_1, then once again the other three viewpoints repeat through the exact same configurations prior to reaching a rejection. So our final strategy for the remaining cases was to break them up into subcases where we specify the exact location in the partial ordering of the viewpoints we found to be cycling. This allowed us to finally reject all of the remaining cases, completing the proof.

3.2 Lemmas

We now prove a set of lemmas that all realizations of an LVP must satisfy. These lemmas are critical for allowing us to reject all of the orderings. In these lemmas, for any two points p and q on the boundary of the polygon we let (p, q) denote the counterclockwise portion of the boundary of P starting at p and walking towards q.

We begin with a well-known lemma that applies even to the classic art gallery problem (without limited visibility). This observation is not new, but it is critical in our code, and we state it here for completeness.

Lemma 2 ***Encircling lemma***
Let $(p_i, p_j, p_k, p_l, p_m, p_n)$ be a partial ordering on LVP. If $\overline{p_i p_k}$, $\overline{p_j p_l}$, $\overline{p_m p_i}$, and $\overline{p_l p_n}$ are not pierced, then $\overline{p_i p_l}$ is not pierced.

Proof. See Figure 2 for the illustration. Since $\overline{p_i p_k}$ and $\overline{p_j p_l}$ are both not pierced, then $\overline{p_i p_l}$ cannot be pierced by any point in (p_i, p_l). Since $\overline{p_l p_n}$ and $\overline{p_m p_i}$ are both not pierced, then $\overline{p_i p_l}$ cannot be pierced by any point in (p_l, p_i). Let a, b be two intersections points from $\overline{p_i p_k}$, $\overline{p_j p_l}$, and $\overline{p_i p_m}$, $\overline{p_l p_n}$ respectively. Consider the Jordan curve obtained by $\overline{p_i a}$, $\overline{a p_l}$, $\overline{p_l b}$, and $\overline{b p_i}$. The boundary of the polygon cannot enter the Jordan curve, as that would contradict the assumption of $\overline{p_i p_k}$, $\overline{p_j p_l}$, $\overline{p_m p_i}$, and $\overline{p_l p_n}$ are not pierced. This implies $\overline{p_i p_l}$ cannot be pierced since $\overline{p_i p_l}$ is contained within the Jordan curve. □

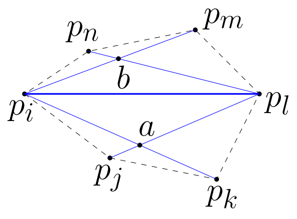


Figure 2: $\overline{p_i p_l}$ is unpierceable

The next lemma that we prove is a "helper lemma" for two later lemmas and allows us to be able to argue that for some orderings, every realization must have the line segments connecting two pairs of points crossing each other.

Lemma 3 *Let (p_i, p_j, p_k, p_l) be the partial ordering on LVP. Suppose $\overline{p_i p_k}$ and $\overline{p_j p_l}$ are not pierced, then $\overline{p_i p_k}$ and $\overline{p_j p_l}$ must cross each other.*

Proof. See Figure 3 for the illustration. Consider the Jordan curve obtained by the counterclockwise border of polygon from p_i to p_k and $\overline{p_i p_k}$. Note that p_j is on the Jordan curve somewhere. The counterclockwise border from p_k to p_i must leave p_k "above" $\overline{p_ip_k}$ or else $\overline{p_ip_k}$ would be pierced and therefore is outside of the Jordan curve. Then by the Jordan curve theorem, p_l must be also outside of the Jordan curve. This implies that $\overline{p_jp_l}$ must cross outside of the Jordan curve. Since $\overline{p_jp_l}$ is not pierced, it cannot intersect the counterclockwise border from p_i to p_k, and therefore it must cross $\overline{p_ip_k}$. □

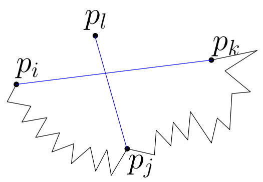


Figure 3: $\overline{p_ip_k}$ and $\overline{p_jp_l}$ must cross

Lemma 4 ***Crossing lemma***
Suppose (p_i, p_j, p_k, p_l) is the partial ordering on LVP. If p_i sees p_k and p_j sees p_l, then $d(p_i, p_l)$ and $d(p_j, p_k)$ cannot both be greater than ρ.

Proof. See Figure 4 for the illustration. Since p_i sees p_k and p_j sees p_l, from Lemma 3 we know $\overline{p_ip_k}$ and $\overline{p_jp_l}$ must cross. Since p_i sees p_k and p_j sees p_l, it must be that $d(p_i, p_k) \le \rho$ and $d(p_j, p_l) \le \rho$. Now assume that $d(p_i, p_l) > \rho$ and $d(p_j, p_k) > \rho$. Let p_o be the intersection of $\overline{p_ip_k}$ and $\overline{p_jp_l}$. Then $d(p_i, p_o)+d(p_l, p_o) > d(p_i, p_l)$ and $d(p_j, p_o)+d(p_k, p_o) > d(p_j, p_k)$. But $d(p_i, p_k) + d(p_j, p_l) \le 2\rho$. This implies that $d(p_i, p_l) + d(p_j, p_k) \le 2\rho$. Since $d(p_i, p_l) > \rho$, we have that $d(p_j, p_k) \le \rho$, which contradicts our assumption that $d(p_j, p_k) > \rho$. □

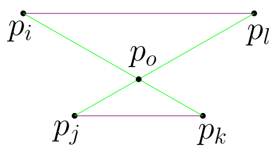


Figure 4: contradicts our assumption that $d(p_j, p_k) > \rho$

The remaining lemmas are in the appendix due to the page limit.

3.3 Edge Colors

When we consider a partial ordering, we encode information regarding the line segments connecting points in the ordering using a complete graph where there is a vertex for each point in the partial ordering and the edges have a color which indicates some properties that the corresponding points must satisfy. In particular, there are three properties that we are concerned about. The first is about whether or not the pair of points are pierced in a realization. The possible answers we might know are "yes" (*every* realization of the partial order must have the points be pierced), "no" (*every* realization of the partial order must have the points *not* be pierced), and "maybe". The second property we are concerned with is whether or not the points are farther than ρ apart or not (i.e., are they in each other's disks or not). Again the answers are "yes" (inside each other's disks in *every* realization), "no" (outside of each other's disks in *every* realization), and "maybe". The final property is whether or not the points can see each other, where "yes" means they must see each other in every realization, "no" means they must not see each other in every realization, and "maybe". Out of the twenty-seven ways of assigning answers to these properties, there are ten combinations that are feasible (e.g., if two points see each other then clearly they cannot be pierced and they must be inside each other's disks). We assign an edge color to represent the various combinations (see Table 1). In the figures for the lemmas and in the lower bound construction, we draw the figure with the edge color that corresponds to this table. Also, the code to prove the upper bound uses these edge colors to check to see if a partial ordering is violating any lemma.

$\overline{p_ip_j}$ pierced	$d(p_i, p_j) \le \rho$	p_i sees p_j	Color
yes	yes	no	red
no	yes	yes	green
maybe	yes	maybe	yellow
yes	no	no	brown
no	no	no	purple
maybe	no	no	plum
yes	maybe	no	pink
no	maybe	maybe	blue
maybe	maybe	maybe	cyan
maybe	maybe	no	orange

Table 1: Edge colors

3.4 Our code

We now give an overview of how our computer program that is used to reject partial orderings. At the beginning of the execution of the code, we will be considering some partial ordering (which at a minimum contains all guards $g_0, \ldots, g_8$ in order and may also contain some viewpoints depending on the case) as well as an assignment of edge colors for each of the guard/guard pairs. Initially, the colors for the guard/guard pairs will either be pink (we require them to be pierced), blue (we require them not to be pierced) or cyan (we do not care initially if they are pierced or not). The goal is to show that *every* ordering whose realization would satisfy these initial assumptions we are making would have to violate one of the lemmas, contradicting the existence of the realization. Then by considering enough of these cases, we can rule out any realization of any ordering.

The key function in our code is the function called *isFeasibleOrdering*() which we have provided high-level pseudocode for in Algorithm 1. In this paper, we have color coded the different parts of the algorithm, and the colors match the colors seen in a rich text document of the actual Java source code found at `https://github.com/computational-geometry-lab/visibility-simple-polygon-upper-bound`. The first time the function is called, we initialize edge colors for all pairs of points in the ordering to the most specific edge color we can given what we know about the case. For example, if one point is a guard and the other point is a viewpoint, then either we know the guard should see it (the edge must be green) or the guard should not see it. Initially, we may not yet know *why* a guard should not see the viewpoint (it could be pierced or it could be outside of the disk), so we initialize these edges to orange to reflect this. For pairs of viewpoints, we are not concerned whether or not they see each other so they are initialized to cyan, but as the program moves on, the lemmas may imply that some viewpoints need to be pierced, some viewpoints might need to see each other, etc. The cases that we are considering determine the edge color of any guard/guard pairs.

We then begin an iterative process of "evaluating" the lemmas to see if we can make any of the edge colors more specific, or maybe even if we can reject the partial ordering entirely. Indeed, if we see that the edge colors violate a lemma then we simply reject the partial ordering. If we have all of the conditions of the lemma except for one, then we might be able to make some edge color become more specific. For example, suppose we are evaluating the crossing lemma. Perhaps we see that we have points p_i, p_j, p_k, and p_l such that p_i and p_k are green, p_j and p_l are green, and p_i and p_l are purple. Then the crossing lemma implies that p_j and p_k must be inside of each other's disks, so we may be able to make their edge color more specific. In this example, if it currently is cyan, we can update it to yellow. If it is currently orange, we can update it to red (if they don't see each other and have to be inside each other's disks then they must be pierced). If it is currently blue, we can update it to green. An updated edge color for these two points may in turn end up impacting another lemma, causing additional pairs to get more specific edge colors assigned to them. This process continues until either we find an outright violation of a lemma (where the function rejects the partial order), or eventually the edge colors converge. If the edge colors converge, the function returns true, indicating that we were not able to reject the current partial order for the case we are considering.

If we are unable to reject the current partial order for the case we are considering, then we consider extending the partial order by adding one viewpoint to the partial order. We place the viewpoint in the first "gap" of the partial order, obtaining an extension, and we then call *isFeasibleOrdering*() again on this extension. If we get a rejection, then we move the viewpoint to the next gap of the partial order and try again. If all gaps reject, we can reject the partial order. This process repeats until either we are able to reject all extensions of the initially considered partial order, or until we have found an extension containing all of the viewpoints that we are considering that our code is unable to reject (which as it turns out does not happen).

Finally, we remark about the forbidden subgraphs referred to in the red block of Algorithm 1. We were able to determine that there are a few quite general cases (all but 3 or 4 guard/guard pairs are cyan) that do not directly violate the lemmas but lead to rejections eventually. We found it advantageous to hardcode these patterns to be rejections rather than to let the code execute until the lemmas reject them (similar to a memoization dynamic programming algorithm, except the rejection was determined in a completely different execution of the algorithm rather than earlier in the same execution of the algorithm).

Algorithm 1 boolean isFeasibleOrdering(*ordering*, *edgeColors*)

```
if $edgeColors$ is null then
    initialize all edges to default values:
    - guard/viewpoint pairs are initialized to green if the guard is supposed to see the viewpoint and otherwise are initialized to orange.
    - guard/guard pairs are initialized to be pink, blue, or cyan depending on the case we are currently considering.
    - viewpoint/viewpoint pairs are initialized to cyan.
else
    we are extending a previously-considered ordering where we know some of the colors
while $edgeColors$ has not converged do
    evaluate encircling lemma
    evaluate crossing lemma
    evaluate purple/green crossing lemma
    check forbidden subgraphs
    for any pink edges do
        compute blockers and evaluate every blocker lemmas
return true
```

References

[1] James Abello, Hua Lin, and Sekhar Pisupati. On visibility graphs of simple polygons. *Congressus Numerantium*, 90:119–128, 1992.

[2] Boaz Ben-Moshe, Matthew J. Katz, and Joseph S. B. Mitchell. A constant-factor approximation algorithm for optimal 1.5d terrain guarding. volume 36, pages 1631–1647, 2007.

[3] Therese Biedl and Saeed Mehrabi. Grid-obstacle representations with connections to staircase guarding. In *Graph Drawing and Network Visualization*, pages 81–87, 2018.

[4] Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[5] Alon Efrat and Sariel Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006.

[6] Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *Discrete & Computational Geometry*, 17(2):143–162, 1997.

[7] Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *Discret. Comput. Geom.*, 17(2):143–162, 1997.

[8] Matt Gibson, Gaurav Kanade, Erik Krohn, and Kasturi Varadarajan. An approximation scheme for terrain guarding. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 140–148, 2009.

[9] Matt Gibson, Gaurav Kanade, Erik Krohn, and Kasturi R. Varadarajan. Guarding terrains via local search. *JoCG*, 5:168–178, 2014.

[10] Matt Gibson, Erik Krohn, and Matthew Rayford. Guarding monotone polygons with half-guards. In *Canadian Conference on Computational Geometry*, pages 168–173, 2017.

[11] Matt Gibson, Erik Krohn, and Qing Wang. The VC-dimension of visibility on the boundary of a simple polygon. In *Algorithms and Computation*, 2015.

[12] Matt Gibson, Erik Krohn, and Qing Wang. The VC-dimension of visibility on the boundary of monotone polygons. *Computational Geometry*, 77:62–72, 2019.

[13] Matt Gibson-Lopez and Zhongxiu Yang. The VC-Dimension of Limited Visibility Terrains. In *32nd International Symposium on Algorithms and Computation*, volume 212, pages 5:1–5:17, 2021.

[14] Alexander Gilbers and Rolf Klein. New results on visibility in simple polygons. In *Algorithms and Data Structures*, pages 327–338, 2009.

[15] Alexander Gilbers and Rolf Klein. A new upper bound for the VC-dimension of visibility regions. *Comput. Geom.*, 47(1):61–74, 2014.

[16] James King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In *LATIN*, volume 3887 of *Lecture Notes in Computer Science*, pages 629–640, 2006.

[17] James King. VC-dimension of visibility on terrains. In *Canadian Conference on Computational Geometry*, pages 27–30, 2008.

[18] James King and Erik Krohn. Terrain guarding is NP-hard. *SIAM J. Comput.*, 40(5):1316–1339, 2011.

[19] Erik Krohn and Bengt J. Nilsson. The complexity of guarding monotone polygons. In *Canadian Conference on Computational Geometry*, pages 167–172, 2012.

[20] Erik Krohn and Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013.

[21] Erik A. Krohn and Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, pages 1–31, 2012.

[22] Thomas Prellberg. Uniform q-series asymptotics for staircase polygons. *Journal of Physics A: Mathematical and General*, 28(5):1289–1304, 1995.

[23] Sven Schuierer, Gregory J. E. Rawlins, and Derick Wood. A generalization of staircase visibility. In *Computational Geometry-Methods, Algorithms and Applications*, pages 277–287, 1991.

[24] Pavel Valtr. Guarding galleries where no point sees a small area. *Israel Journal of Mathematics*, 104:1–16, 1998.

Appendix

A The Coordinates of Lower Bound Construction

Vertex #	Label	x-coordinate	y-coordinate
1	vp_{BCFGH}	1.02959	-0.935554
2	vp_{BCFG}	1.029589844	-0.935553995
3	vp_{ABCFGH}	1.0296	-0.9356
4	vp_{CF}	1.02958398	-0.935599312
5	vp_{BCF}	1.029583982	-0.935599318
6	vp_{ABCF}	1.029584	-0.9355994
7	vp_{ACFGH}	1.029546	-0.93586
8	vp_{ACFG}	1.02954	-0.9358599
9	vp_{CEGH}	1.029532	-0.935942
10	vp_{ACEGH}	1.029534	-0.935946
11	vp_{ACEG}	1.029532	-0.935946
12	vp_{ACG}	1.0295124	-0.9359456
13	vp_{ACGH}	1.0295116	-0.9359464
14	$blocker$	1.02948	-0.93604
15	vp_{CEG}	1.0294764	-0.93604
16	vp_{CGH}	1.02946	-0.93606
17	vp_{CEFGH}	1.029538	-0.936214
18	vp_{ACEFGH}	1.029556	-0.936282
19	vp_{ACEFG}	1.02954	-0.936282
20	$blocker$	1.0295	-0.9363
21	vp_{ACE}	1.029472	-0.936292
22	vp_{ACDEGH}	1.02932	-0.93662
23	vp_{ACDE}	1.02926	-0.9366
24	vp_{ACD}	1.029255	-0.936601
25	vp_{CDGH}	1.0269	-0.941392
26	vp_{ACDGH}	1.027066	-0.941702
27	vp_{ACDG}	1.026898	-0.941742
28	vp_{AEFGH}	1.0334	-0.9542
29	vp_{AH}	1.03348	-0.9544
30	g_A	1.033642	-0.954372
31	vp_{ACDEFG}	1.03196	-0.95468
32	vp_{ADEFG}	1.031962	-0.954732
33	$vp_{ABCDEFG}$	1.032	-0.9548
34	vp_{AE}	1.01877	-0.964708
35	vp_{ADE}	1.01878	-0.964724
36	vp_{ABCDE}	1.018846	-0.964838
37	vp_{ABCD}	1.018122	-0.965586
38	vp_{ADEG}	1.021764	-0.980328
39	vp_{ACDEG}	1.021778	-0.980352
40	vp_{ABCDEG}	1.0218	-0.9804
41	$blocker$	1.0224	-0.9856
42	vp_{CDEGH}	1.022358	-0.985712
43	vp_{ADEGH}	1.02268	-0.986324
44	vp_{DEGH}	1.02267964	-0.98632496
45	$vp_{ABCDEGH}$	1.022746	-0.986436
46	vp_{BCDEGH}	1.0226	-0.98684
47	vp_{ACDH}	0.97592	-1.12428
48	vp_{ABCDH}	0.976	-1.1244
49	vp_{ABEH}	1.108	-1.376
50	vp_{BEH}	1.10816	-1.37696
51	vp_{BE}	1.10816	-1.377
52	vp_{ABEFGH}	1.122468	-1.392592
53	g_B	1.132	-1.399
54	vp_{BCDE}	1.131996	-1.399001
55	vp_{BF}	1.124616	-1.4024728
56	vp_{BCDEF}	1.1246168	-1.4024726
57	vp_{BDEF}	1.1231612	-1.4031576
58	vp_{BDE}	1.12316	-1.40316
59	vp_{BEFG}	1.1244	-1.40452
60	vp_{BCDEFG}	1.124532	-1.404592
61	vp_{CDEG}	1.1238	-1.405
62	vp_{BGH}	1.1391437	-1.4357441
63	vp_{BDEGH}	1.139144	-1.435744
64	vp_{EGH}	1.13914711	-1.43576
65	$blocker$	1.13916	-1.43576
66	vp_{BDEFGH}	1.139352	-1.436052
67	vp_{DEFGH}	1.139352039	-1.4360522
68	vp_{BEFGH}	1.13938	-1.43609
69	$vp_{BCDEFGH}$	1.1394	-1.4361
70	vp_{CDEFGH}	1.1394063	-1.436132
71	vp_{BCDEH}	1.13946	-1.4364
72	vp_{CDEH}	1.139460088	-1.436400444
73	vp_{BDEH}	1.14032	-1.43832

Table 2: The coordinates of our construction (Part 1 of 4). The boundary of the polygon starts from Vertex #1 to Vertex #73

Vertex #	Label	x-coordinate	y-coordinate
74	vp_{DEH}	1.14032016	-1.4383208
75	$blocker$	1.1404	-1.4384
76	vp_{BFH}	1.1407896	-1.4388912
77	vp_{BEFH}	1.14079	-1.438891
78	vp_{BDEFH}	1.140792	-1.438892
79	vp_{BCDEFH}	1.1408	-1.4389
80	vp_{CDEFH}	1.140802	-1.43891
81	vp_{BFGH}	1.1408344	-1.4389524
82	$blocker$	1.140836	-1.438952
83	vp_{ABDEH}	1.140872	-1.439072
84	vp_{ADEH}	1.14087202	-1.4390721
85	vp_{ABCDEH}	1.14088	-1.43908
86	vp_{ACDEH}	1.1408808	-1.4390836
87	vp_{ABFH}	1.4236	-1.7744
88	$vp_{ABCDEFH}$	1.424	-1.774
89	vp_{ABH}	1.4774	-1.8358
90	vp_{AB}	1.478	-1.836
91	vp_{ABCEH}	1.48	-1.82
92	vp_{EH}	1.480279	-1.8205608
93	vp_{CEH}	1.48028	-1.82056
94	vp_{AEH}	1.48076	-1.82124
95	vp_{ACEH}	1.4808	-1.8212
96	vp_{ABEFH}	1.48776	-1.81908
97	vp_{AEFH}	1.487766	-1.81909
98	vp_{EFH}	1.48776688	-1.8190917
99	vp_{EF}	1.48776689	-1.8190918
100	vp_{ABCEFH}	1.4884	-1.8184
101	vp_{ACEFH}	1.488402	-1.818403
102	vp_{AFH}	1.48868	-1.81879
103	vp_{ABCFH}	1.48872	-1.81874
104	vp_{CFH}	1.4887262	-1.818752
105	vp_{ABCH}	1.4956	-1.8136
106	vp_{CH}	1.495672	-1.813736
107	g_C	1.49581	-1.810934
108	vp_{CG}	1.49768	-1.816472
109	vp_{CFG}	1.4978	-1.81628
110	vp_{CDEFG}	1.498	-1.816
111	vp_{DEFG}	1.4983	-1.8171
112	vp_{FG}	1.49837	-1.81724
113	vp_{EFG}	1.4984	-1.8172
114	vp_{CFGH}	1.49904	-1.817
115	$vp_{ACDEFGH}$	1.4992	-1.8168
116	vp_{ACEF}	1.503532	-1.823676
117	vp_{AF}	1.50353249	-1.8236769
118	vp_{AEF}	1.5035325	-1.8236769
119	vp_{ACDEF}	1.5036	-1.8236
120	vp_{ADEF}	1.503628	-1.823648
121	vp_{ACH}	1.50529	-1.82202
122	vp_{ACDEFH}	1.5053	-1.822
123	vp_{ADEFH}	1.50533	-1.82204
124	vp_{ACDFGH}	1.505504	-1.821984
125	vp_{ACFH}	1.50550784	-1.8219882
126	vp_{ACDFH}	1.505508	-1.821988
127	vp_{AC}	1.50551291	-1.8219911
128	vp_{ACF}	1.50551292	-1.8219911
129	vp_{ACDF}	1.505513	-1.821991
130	vp_{ACDFG}	1.50554	-1.821992
131	vp_{ADFG}	1.5055406	-1.8219928
132	vp_{ABCDF}	1.5058	-1.8222
133	vp_{ABF}	1.5058054	-1.8222061
134	vp_{ABDF}	1.5058054	-1.8222061
135	vp_{ABCFG}	1.50784	-1.82225
136	vp_{AG}	1.5078418	-1.8222523
137	vp_{ABG}	1.50784191	-1.8222521
138	vp_{ABFG}	1.507842	-1.822252
139	vp_{AFG}	1.5078428	-1.8222532
140	vp_{ABCDFG}	1.508	-1.822
141	vp_{ABDFG}	1.50804	-1.82204
142	vp_{DFG}	1.50808	-1.82212
143	vp_{ABCG}	1.5101	-1.8196
144	$vp_{ABCDFGH}$	1.51586	-1.8101
145	vp_{DG}	1.5158604	-1.8101016
146	vp_{DGH}	1.51586076	-1.8101014

Table 3: The coordinates of our construction (Part 2 of 4). The boundary of the polygon starts from Vertex #74 to Vertex #146

Vertex #	Label	x-coordinate	y-coordinate
147	g_D	1.515882	-1.810066
148	vp_{ADEFGH}	1.5188	-1.814
149	vp_{GH}	1.5191176	-1.814584
150	vp_{EFGH}	1.51912	-1.81458
151	vp_{ADG}	1.52584	-1.822272
152	*blocker*	1.526	-1.822
153	vp_{BDFG}	1.5596	-1.8106
154	vp_{BFG}	1.55964	-1.81062
155	vp_{BG}	1.559656	-1.810616
156	vp_{BDEFG}	1.56	-1.81
157	vp_{BCD}	1.579916	-1.800128
158	vp_{BCDFG}	1.57992	-1.80012
159	vp_{BCDG}	1.579981	-1.800035
160	vp_{BC}	1.5799814	-1.8000349
161	vp_{BCG}	1.5799814	-1.8000349
162	*blocker*	1.58	-1.8
163	*blocker*	1.9412	-1.3738
164	vp_{AEFG}	1.94132	-1.37384
165	*blocker*	1.94845	-1.358256
166	vp_{ABCEG}	1.948452	-1.358254
167	vp_{ABEG}	1.94845204	-1.3582539
168	vp_{AEG}	1.94845208	-1.358254
169	vp_{BCDEG}	1.9485	-1.3582
170	vp_{EG}	1.948502	-1.358202
171	vp_{BEG}	1.948504	-1.358198
172	vp_{BCEG}	1.948535	-1.358164
173	vp_{ABCDGH}	1.949584	-1.355282
174	vp_{ABCGH}	1.94958456	-1.3552814
175	*blocker*	1.9496	-1.3552
176	vp_{AEGH}	1.949604	-1.355201
177	vp_{ABCEGH}	1.9496092	-1.3551906
178	vp_{ABEGH}	1.94960928	-1.3551906
179	vp_{BCDH}	1.9509998	-1.3526012
180	*blocker*	1.951	-1.3526
181	vp_{BCGH}	1.95100388	-1.3525961
182	vp_{BCEGH}	1.9510039	-1.352596
183	vp_{BEGH}	1.95100392	-1.352596
184	g_E	1.96059	-1.300768
185	*blocker*	1.96064	-1.30077
186	vp_{FGH}	1.960676	-1.300784
187	vp_{AFGH}	1.960684	-1.30078
188	vp_{ABCE}	1.965229	-1.295611
189	$vp_{ABCEFGH}$	1.9652	-1.2956
190	*blocker*	1.96539	-1.29539
191	*blocker*	2	-1.216
192	*blocker*	2.00048	-1.21542
193	vp_{CDG}	2.0771061	-1.0712023
194	vp_{CD}	2.07708	-1.07108
195	vp_{CDFG}	2.077	-1.071
196	vp_{DEF}	2.088	-1.004
197	vp_{DE}	2.05228	-0.98792
198	vp_{CDE}	2.0522	-0.98804
199	vp_{CDEF}	2.052	-0.988
200	*blocker*	2.036	-0.98
201	vp_{BH}	2.036228	-0.979692
202	vp_{BCFH}	2.0362	-0.97969
203	g_F	2.02862	-0.9787248
204	vp_{FH}	2.03002	-0.97878
205	*blocker*	2.03	-0.97877
206	vp_{ABFGH}	2.0333	-0.9765
207	vp_{ABGH}	2.0333166	-0.9764922
208	vp_{AGH}	2.03336	-0.97648
209	vp_{ABCDFH}	2.033504	-0.970801
210	*blocker*	2.0335	-0.9708
211	vp_{BCDFH}	2.0335092	-0.9707842
212	vp_{BCDFGH}	2.033508	-0.970784
213	vp_{BCDGH}	2.03350815	-0.9707838
214	vp_{CDF}	2.0347616	-0.9687436
215	vp_{CDFH}	2.03476	-0.968744
216	vp_{CDH}	2.03476189	-0.9687409
217	vp_{CDFGH}	2.03474	-0.96874

Table 4: The coordinates of our construction (Part 3 of 4). The boundary of the polygon starts from Vertex #147 to Vertex #217

Vertex #	Label	x-coordinate	y-coordinate
218	vp_{BCDF}	2.03458	-0.9684876
219	*blocker*	2.0345776	-0.9684876
220	vp_{DFH}	2.03452	-0.96832
221	vp_{DF}	2.0344632	-0.9683068
222	vp_{BDF}	2.034462	-0.9683076
223	vp_{BDFH}	2.03446	-0.968308
224	vp_{BDFGH}	2.0344	-0.9683
225	vp_{BDGH}	2.0344068	-0.9682884
226	vp_{ABDFH}	2.033512	-0.967701
227	vp_{ABDFGH}	2.0335	-0.9677
228	vp_{ABDGH}	2.03352	-0.96766
229	vp_{BDH}	2.0335652	-0.9675704
230	*blocker*	2.03356	-0.96757
231	vp_{DFGH}	2.033576	-0.966208
232	vp_{ADF}	2.0335514	-0.9662022
233	vp_{ADFH}	2.033544	-0.966202
234	vp_{ADFGH}	2.0335	-0.9662
235	vp_{AD}	2.03350488	-0.966192
236	vp_{ADH}	2.0335048	-0.966192
237	vp_{ADGH}	2.033504	-0.966192
238	vp_{ABCDEF}	2.029928	-0.9658114
239	*blocker*	2.029912	-0.965811
240	$vp_{ABCDEFGH}$	2.0296	-0.9658
241	*blocker*	2.0294	-0.96564
242	g_G	2.02164	-0.965478
243	vp_{ABCDG}	2.02387	-0.9617402
244	*blocker*	2.02386	-0.96174
245	vp_{BDG}	2.02404	-0.9612
246	vp_{ABDG}	2.024	-0.961208
247	*blocker*	2.0236	-0.9612
248	vp_{DEG}	2.01611	-0.95197
249	vp_{BDEG}	2.0161	-0.95198
250	vp_{ABDEG}	2.01604	-0.9520002
251	vp_{ABDEGH}	2.016	-0.952
252	vp_{ABDEFG}	1.920059	-0.899997
253	vp_{ABDEF}	1.9200584	-0.8999966
254	vp_{ABDE}	1.92005836	-0.8999966
255	$vp_{ABDEFGH}$	1.92	-0.9
256	vp_{ABDEFH}	1.9194	-0.8996
257	*blocker*	1.9193	-0.8994
258	vp_{BD}	1.9193322	-0.8993193
259	vp_{ABD}	1.9193318	-0.8993194
260	vp_{ABDH}	1.91932	-0.89932
261	vp_{ABEFG}	1.7468	-0.83092
262	vp_{ABE}	1.74677	-0.83089
263	vp_{BEF}	1.7467214	-0.8308794
264	vp_{ABEF}	1.74672	-0.83088
265	*blocker*	1.74	-0.832
266	vp_{BCEFG}	1.2142	-0.8549
267	vp_{BCE}	1.21412	-0.85488
268	vp_{CEF}	1.21402	-0.854874
269	vp_{ABCEFG}	1.214	-0.85504
270	vp_{ABC}	1.2139996	-0.8550396
271	*blocker*	1.212	-0.856
272	*blocker*	1.211996	-0.855996
273	*blocker*	1.2099222	-0.8556876
274	vp_{CE}	1.20992204	-0.8556876
275	vp_{BCEF}	1.209922	-0.855688
276	vp_{ABCEF}	1.20992	-0.85569
277	*blocker*	1.2099	-0.8557
278	vp_{BCEFGH}	1.048	-0.92
279	vp_{BCH}	1.0479	-0.9199
280	vp_{BCEFH}	1.0469	-0.91994
281	vp_{BCEH}	1.046884	-0.919936
282	vp_{CEFG}	1.039996	-0.927988
283	*blocker*	1.04	-0.928
284	vp_{CEFH}	1.039744	-0.927988
285	*blocker*	1.04163	-0.93519
286	vp_{DEFH}	1.0416296	-0.93519
287	vp_{DH}	1.04162	-0.93518
288	g_H	1.042	-0.9359

Table 5: The coordinates of our construction (Part 4 of 4). The boundary of the polygon starts from Vertex #218 to Vertex #288

B Additional Lemmas

Lemma 5 ***Purple/green crossing lemma***
Suppose (p_i, p_j, p_k, p_l) is the partial ordering on LVP such that (1) p_i and p_j is plum, (2) p_k and p_l is plum, (3) p_i and p_k are purple, (4) p_j and p_l are green, (5) p_i sees some point in (p_j, p_k), and (6) p_j and p_k are blue, then $d(p_i, p_l) \leq \rho$.

Proof. The lemma is proved by contrapositive. We assume that $d(p_i, p_l) > \rho$, and we will show that at least one of the six conditions must be false. In particular, we will show that if (1), (3), (4), (5), and (6) are all true, then this implies that (2) is false (which means that p_k and p_l are inside of each other's disks). See Figure 5 for the illustration. Since $\overline{p_ip_k}$ cannot be pierced and p_j sees p_l, from Lemma 3 we know $\overline{p_ip_k}$ and $\overline{p_jp_l}$ must cross. Without loss of generality, we set p_j at $(0, 0)$ and p_i at $(0, \rho + \epsilon)$ for some $\epsilon > 0$. Assume p_r is the point seen by p_i in (p_j, p_k), clearly $y(p_r) > \epsilon$. Since (p_i, p_j, p_k, p_l) is counterclockwise ordering of the LVP and $\overline{p_jp_k}$ is not pierced, then $x(p_r) > 0$ otherwise $\overline{p_ip_r}$ cannot be fully inside of polygon. This also implies that $x(p_k) > 0$ or else p_r would pierce $\overline{p_jp_k}$. Since $\overline{p_ip_k}$ cannot be pierced and $d(p_i, p_k) > \rho$, we know that $\overline{p_jp_k}$ starts outside of $disk(p_i)$, then intersect $disk(p_i)$, and ends outside of $disk(p_i)$. Thus $y(p_k) > \epsilon$. Assume $d(p_i, p_l) \geq \rho$. We know that $x(p_l) > 0$ or else p_l would pierce $\overline{p_kp_i}$. Since p_j sees p_l, then p_k cannot pierce $\overline{p_jp_l}$, thus $y(p_l) > 0$.

Let p_o be the right intersection of $disk(p_i)$ and $disk(p_j)$. Consider the points $(-\rho, 0)$ and $(\rho, 0)$. Note that these points are on $disk(j)$. We now consider whether p_k is in $disk(p_o)$. We know that $x(p_k) > 0$, $y(p_k) > 0$ and p_k is outside of $disk(p_i)$. If p_k is in $disk(p_j)$, then clearly p_k is in $disk(p_o)$.

We now consider if p_k is outside of $disk(p_j)$. Since $d(p_j, p_l) \leq \rho$, p_l is outside of $disk(p_i)$ and p_l cannot block $\overline{p_kp_i}$, then the $x(p_k)$ should less than the intersection of $y = 0$ and the ray $\overrightarrow{p_ip_o}$, where p_i is at $(0, \rho + \epsilon)$ and p_o is at $(\frac{\sqrt{3\rho^2 - 2\rho\epsilon - \epsilon^2}}{2}, \frac{\rho + \epsilon}{2})$. Thus $x(p_k) < \sqrt{3\rho^2 - 2\rho\epsilon - \epsilon^2}$. Therefore the maximum of $d(p_o, p_k)$ is strictly less than ρ.

Now we need to show $disk(k)$ contain $(\rho, 0)$ and does not contain $(-\rho, 0)$. If p_k is in $disk(p_j)$, then clearly $d(p_k, (\rho, 0)) < \rho$ and $d(p_k, (-\rho, 0)) > \rho$. We now consider p_k is at far right and below $\overrightarrow{p_ip_o}$, clearly we have $d(p_k, (-\rho, 0)) > \rho$ and $d(p_k, (\rho, 0)) < \sqrt{3\rho^2 - 2\rho\epsilon - \epsilon^2} - \rho < \rho$. Therefore, $disk(k)$ contains the entire arc of $disk(j)$ between p_o and $(\rho, 0)$.

We know that p_l also has to be above $\overrightarrow{p_ip_k}$, and now we need to show $\overrightarrow{p_kp_i}$ must enter $disk(p_i)$ before it leaves $disk(p_k)$. We previously showed that $d(p_o, p_k)$ is strictly less than ρ, then $d(p_i, p_k) < 2\rho$. Now we have a convex feasible region for p_l and the entire boundary of that convex feasible region is contained inside $disk(k)$. So we have $d(p_k, p_l) < \rho$, that contradict with out assumption of $d(p_k, p_l) > \rho$, therefore $d(p_i, p_l) < \rho$. □

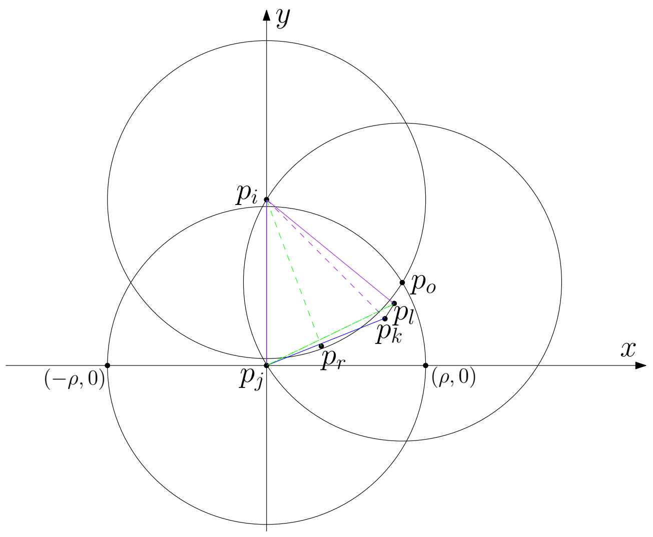


Figure 5: $d(p_k, p_l) < \rho$

The following five blocker lemmas allow us to consider the situation when the "blocker" $p_z \in (p_i, p_j)$ that pierces $\overline{p_ip_j}$, how does this impact the points in (p_j, p_i)? For instance, what these three points p_i, p_j, p_z can see from p_j to p_i?

We can say that a point p_z must be a blocker for a pink or red pair of points p_i and p_j such that $p_z \in (p_i, p_j)$ if p_z is blue to both p_i and p_j, there is a $p_m \in (p_j, p_i)$ that is blue to p_i, and there is a point $p_n \in (p_m, p_i)$ that is blue to p_j. See Figure 6. Indeed, there cannot be a blocker in (p_j, p_i) without blocking either p_i/p_m or p_j/p_n. Also there cannot be a blocker in (p_i, p_z) or (p_z, p_j) without blocking p_i/p_z or p_z/p_j. Therefore the only point that can block p_i and p_j is p_z.

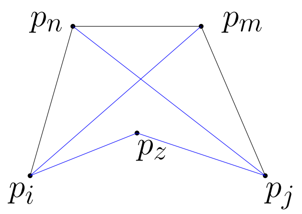


Figure 6: p_z is the only point that can block p_i and p_j

Lemma 6 ***Blocker lemma 1***
Suppose $p_z \in (p_i, p_j)$ such that p_z blocks p_i from p_j, p_z is blue to both p_i and p_j. Further suppose there is a p_x in (p_j, p_i) such that p_x sees p_i and p_j. Then p_x sees p_z.

Proof. See Figure 7 for the illustration. Since p_z pierces $\overline{p_ip_j}$ and p_x sees p_i and p_j, then $\triangle p_xp_ip_j$ has to be inside $disk(p_x)$ by convexity. Since p_z is in the $\triangle p_x p_i p_j$ then we have that $d(p_z, p_x) < \rho$. Since $\overline{p_i p_z}$ and $\overline{p_z p_j}$ are both unpierceable, then no point can pierce $\overline{p_x p_z}$. Therefore p_x sees p_z. □

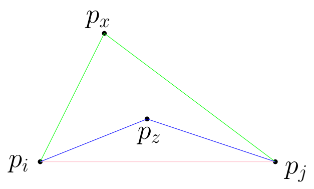


Figure 7: p_x sees p_z

Lemma 7 ***Blocker lemma 2***
Suppose $p_z \in (p_i, p_j)$ such that p_z blocks p_i from p_j, p_z is blue to both p_i and p_j, then there is no point sees p_i and p_j but does not see p_z.

Proof. See Figure 7 for illustration. Consider p_x is the point that sees p_i and p_j, but does not see p_z. Clearly p_x cannot be placed in (p_i, p_z) and (p_z, p_j) otherwise $\overline{p_i p_z}$ or $\overline{p_z p_j}$ will be pierced. Now consider p_x is in (p_j, p_i), from the previous lemma 6 we know that p_x sees p_z, which contradicts our assumption that p_x only sees p_i and p_j, therefore, a guard cannot block two other guards. □

Lemma 8 ***Blocker lemma 3***
Suppose $p_z \in (p_i, p_j)$ such that p_z blocks p_i from p_j, p_z is blue to both p_i and p_j, then every point in (p_i, p_z) cannot see any point in (p_z, p_j).

Proof. Clearly for any point $p_a \in (p_i, p_z)$, p_z pierces $\overline{p_a p_b}$ where $p_b \in (p_z, p_j)$. Therefore every point in (p_i, p_z) cannot see any point in (p_z, p_j). □

Lemma 9 ***Blocker lemma 4***
Suppose we have a point $p_z \in (p_i, p_j)$ where $(p_x, p_z) > \rho$, that pierces $\overline{p_i p_j}$ where $(p_x, p_i) \leq \rho$ and $(p_x, p_j) > \rho$. If $\overline{p_i p_z}$ and $\overline{p_z p_j}$ are both unpierceable, then p_x does not see any point between (p_z, p_j).

Proof. See Figure 8 for the illustration. By convexity, $\overrightarrow{p_i p_z}$ starts inside $disk(p_x)$ and leaves the $disk(p_x)$ prior to touching p_z, so p_j also must be outside of the $disk(p_x)$ for p_z to pierce $\overline{p_i p_j}$. But then since $\overline{p_z p_j}$ is unpierceable, all of the points in (p_z, p_j) cannot pierce $\overline{p_z p_j}$ and therefore cannot be inside $disk(p_x)$ without p_z or p_j blocking it from p_x. □

Lemma 10 ***Blocker lemma 5***
Suppose $p_z \in (p_i, p_j)$ pierces $\overline{p_i p_j}$, and $\overline{p_i p_z}$ and $\overline{p_z p_j}$ are both unpierceable. Further suppose there is a $p_x \in (p_j, p_i)$ such that $d(p_x, p_i) > \rho$, $d(p_x, p_j) > \rho$, and $d(p_x, p_z) > \rho$. If there is a point in (p_i, p_z) in $disk(p_x)$, then p_x cannot see any point in (p_z, p_j).

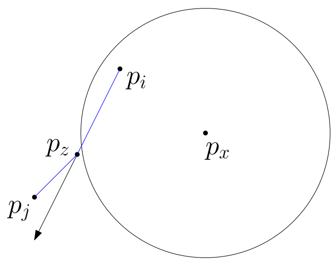


Figure 8: all points in (p_z, p_j) cannot be in $disk(x)$

Proof. Since $d(p_x, p_i) > \rho$ and $d(p_x, p_z) > \rho$, then $\overrightarrow{p_i p_z}$ must enter and leave $disk(p_x)$ prior to reaching p_z. Since p_z pierces $\overline{p_i p_j}$, then p_j must "above" the extension of $\overrightarrow{p_i p_z}$. If p_z pierces $\overline{p_x p_j}$, then clearly p_x does not see p_j and any point in (p_z, p_j), since $\overline{p_z p_j}$ is unpierceable, see Figure 9 for the illustration. Suppose p_z does not pierce $\overline{p_x p_j}$, then the extension of the ray $\overrightarrow{p_z p_j}$ must be entirely outside of $disk(p_x)$, and since $\overline{p_z p_j}$ is unpierceable, it cannot be that p_x sees any point in (p_z, p_j), see Figure 10 for the illustration. □

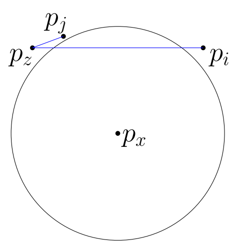


Figure 9: p_z pierces $\overline{p_x p_j}$

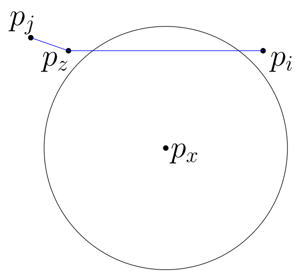


Figure 10: $\overrightarrow{p_z p_j}$ must be entirely outside of $disk(p_x)$

The Number of Non-overlapping Unfoldings in Convex Polyhedra*

Takumi Shiota† Yudai Enomoto‡ Masashi Gorobe† Takashi Horiyama† Tonan Kamata§
Toshiki Saitoh¶ Ryuhei Uehara‡

Abstract

An unfolding of a polyhedron is a flat polygon obtained by selecting candidate cutting lines, cutting along them, and flattening the faces onto the plane. Several methods have been proposed to efficiently count the number of unfoldings by treating the candidate cutting lines as a graph. One such method is to use a zero-suppressed binary decision diagram (ZDD), a compact data structure for representing families of sets. However, some unfoldings overlap depending on the shape of the polyhedra and how they are unfolded. In such cases, two distinct faces intersect or touch in the plane, making it impossible to embed the unfolding in the plane. In this study, we address the problem of counting the number of non-overlapping edge unfoldings in convex polyhedra. We propose a ZDD-based algorithm that excludes overlapping unfoldings by removing their minimal overlapping patterns. Our method applies to both edge and lattice unfoldings, and we present experimental results on several convex polyhedra.

1 Introduction

An unfolding of a polyhedron is a flat polygon obtained by selecting candidate cutting lines, cutting along them, and flattening the faces onto the plane. Depending on the shape of the polyhedron and how it is unfolded, the resulting unfolding may have overlaps, i.e., two distinct faces overlap, or their boundaries are in touch (see Figure 1). When the candidate cutting lines are restricted to the edges of the polyhedron, the unfolding is called an *edge unfolding*. Shephard proposed the following conjecture about edge unfoldings.

Conjecture 1 ([13]) *For any convex polyhedron, at least one non-overlapping edge unfolding exists.*

*A preliminary version was presented at EuroCG2024.
†University of Hyogo, Hyogo, Japan
`takumi_shiota@gsis.u-hyogo.ac.jp`
‡Hokkaido University, Hokkaido, Japan
`yudai1006@gmail.com`, `gorobe.masashi.b0@elms.hokudai.ac.jp`, `horiyama@ist.hokudai.ac.jp`
§Japan Advanced Institute of Science and Technology, Ishikawa, Japan
`{kamata,uehara}@jaist.ac.jp`
¶Kyushu Institute of Technology, Fukuoka, Japan
`toshikis@ai.kyutech.ac.jp`

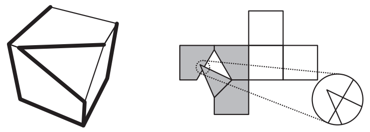


Figure 1: A cube with truncated corners and its overlapping unfolding [11].

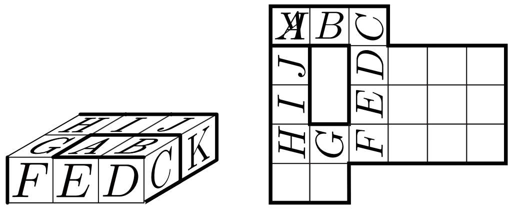


Figure 2: Examples of overlapping lattice unfoldings for $(1, 2, 3)$-cuboid. Faces A and K overlap.

This conjecture is still unsolved, and some studies to solve it are ongoing. One such study is determining whether an overlapping edge unfolding exists for a given polyhedron. This line of investigation is based on the idea that one possible approach to a negative resolution of Conjecture 1 is to find a convex polyhedron for which every edge unfolding overlaps. Shiota and Saitoh presented an algorithm called "rotational unfolding" that can quickly find an overlapping edge unfolding of a polyhedron [15]. This algorithm can determine whether the edge unfolding of a polyhedron overlaps. For convex regular-faced polyhedra (convex polyhedra in which every face is a regular polygon), the existence of overlapping edge unfoldings has been completely demonstrated [2, 4, 6, 7, 15].

General unfoldings, which allow cuts across the faces and edges of a polyhedron, have been studied. Some general unfoldings permit cuts only along specific candidate lines drawn on the faces. One such example is the lattice unfolding of a cuboid formed by connecting multiple $(1, 1, 1)$-cubes [10]. In lattice unfolding, we cut along the edges of the lattice formed by unit squares. For lattice unfoldings of cuboids, the existence of overlapping unfoldings has also been fully demonstrated [10, 14] (see Figure 2 for an example).

The problem of counting the number of unfoldings has been studied in previous works. Schevon experimentally showed that, for randomly generated convex polyhedra,

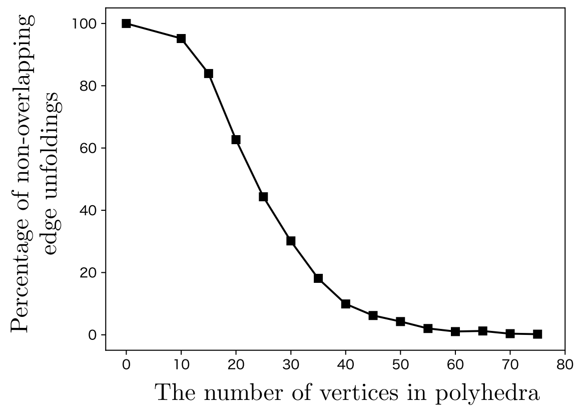


Figure 3: Shevon's experiment on randomly generated convex polyhedra. Each point shows the average percentage of non-overlapping edge unfoldings, computed from 1,000 random unfoldings for each of 5 polyhedra.

the percentage of non-overlapping edge unfoldings decreases as the number of vertices increases [12] (see Figure 3).

The number of unfoldings (including those with overlaps) is equal to the number of *cutting trees* (trees satisfying specific conditions on the candidate cutting lines) [3, 10]. Horiyama et al. counted the number of cutting trees using *zero-suppressed binary decision diagrams (ZDDs)*, a compact data structure for representing families of sets [5, 7].

Our contributions. Herein, we propose an algorithm for counting the number of non-overlapping unfoldings of a given polyhedron using ZDDs and operations over them. The algorithm first enumerates the *minimal overlapping partial unfoldings (MOPUs)*, which are minimal units of unfoldings obtained through the rotational unfolding [15] (the gray faces in Figure 1 correspond to this). Then, we construct a ZDD representing non-overlapping unfoldings by removing the unfoldings containing MOPUs from the ZDD of all possible unfoldings. In this paper, we apply the proposed algorithm to edge unfoldings of convex regular-faced polyhedra and lattice unfoldings of cuboids, and we present the number and percentage of non-overlapping unfoldings for each convex polyhedron (see Tables 1 and 2 for selected results). These results suggest that the number of non-overlapping unfoldings is more significantly affected by the number of faces comprising each MOPU than by the number of MOPUs themselves.

2 Preliminaries

2.1 Edge unfolding of polyhedra

Let Q be a polyhedron. An *unfolding* of the polyhedron Q is a flat polygon formed by cutting Q's edges or faces and unfolding it into a plane. An *edge unfolding* of Q is an unfolding formed by cutting only edges. Q can be viewed as a graph $G_Q = (V_Q, E_Q)$, where V_Q is a set of vertices and E_Q is a set of edges. We have the following lemma for an edge unfolding of Q.

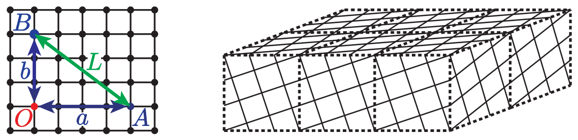


Figure 4: (a) Definition of the edge length L of a cube. (b) An $(\sqrt{10}, 2\sqrt{10}, 3\sqrt{10})$-cuboid.

Lemma 1 ([3] Lemma 22.1.1) *The cutting lines of an edge unfolding for Q form a spanning tree of G_Q.*

We say that two distinct polygons *overlap* if there exists a point p contained in both of the polygons. Note that any point on a boundary is included in the polygons in this paper. An unfolding is *overlapping* if there exists a pair of distinct faces such that the faces overlap. *Rotational unfolding* is an efficient algorithm for determining whether a given polyhedron Q has an overlapping edge unfolding [15].

2.2 Lattice unfolding of cuboids

Let us consider a square lattice where each square has an area of 1×1. Let $A = (a, 0)$ and $B = (0, b)$ be lattice points, where $a \in \mathbb{N}^+$, $b \in \mathbb{N}$, and $a \geq b$, as illustrated in Figure 4 (a). We define $L = \sqrt{a^2 + b^2}$ as the length of the segment AB. An (xL, yL, zL)*-cuboid* is then defined as a box whose edge lengths are xL, yL, and zL along the x-, y-, and z-axes, respectively, where $x, y, z \in \mathbb{Z}^+$. Figure 4 (b) shows an example of a lattice cuboid. Here, a square enclosed by a solid line corresponds to one unit of the original square grid, and a square enclosed by a dotted line has a side length of L.

A *lattice unfolding* of a cuboid C is a planar shape obtained by cutting along the edges of unit squares on the faces of the cuboid. C can be viewed as a graph $G_C = (V_C, E_C)$, where V_C is the set of lattice points (i.e., integer-coordinate points) on the surface of C, and E_C is the set of edges between them. We have the following lemma for a lattice unfolding of C.

Lemma 2 (Figure 5, and [10] Theorems 1 and 3) *Let $S(V_C) \subseteq V_C$ be the set of lattice points located at the vertices of C. Then, the following are equivalent for a subgraph $G_L \subseteq G_C$:*

(1) *A lattice unfolding can be obtained by cutting along G_L.*

(2) *G_L is a tree that satisfies $S(V_C) \subseteq G_L$, and for any vertex v in G_L, if the degree of vertex v is 1, then $v \in S(V_C)$.*

Table 1: The number and percentage of non-overlapping edge unfoldings for convex regular-faced polyhedra (excerpt).

Archimedean solids	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert F\rvert$	#(MOPUs)	#(Edge unfoldings)	#(Non-overlapping edge unfoldings)	Pct.(%)
Truncated dodecahedron	60	90	32	120	4,982,259,375,000,000,000	1,173,681,002,295,455,040	23.56
Truncated icosahedron	60	90	32	240	375,291,866,372,898,816,000	371,723,160,733,469,233,260	99.05
Archimedean n-prisms	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert F\rvert$	#(MOPUs)	#(Edge unfoldings)	#(Non-overlapping edge unfoldings)	Pct.(%)
27-prism	54	81	29	216	37,403,957,244,654,675	35,348,297,730,550,335	94.50
28-prism	56	84	30	336	144,763,597,316,784,768	136,369,030,045,792,768	94.20
29-prism	58	87	31	580	559,560,282,425,278,229	377,763,966,359,384,333	67.51
30-prism	60	90	32	720	2,160,318,004,043,512,500	1,457,228,998,699,944,660	67.45
Archimedean m-antiprisms	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert F\rvert$	#(MOPUs)	#(Edge unfoldings)	#(Non-overlapping edge unfoldings)	Pct.(%)
16-antiprism	32	64	34	64	151,840,963,183,392	146,378,600,602,880	96.40
17-antiprism	34	68	36	204	1,105,779,284,582,146	989,008,190,008,480	89.44
18-antiprism	36	72	38	432	8,024,954,790,380,544	1,517,682,139,108,200	18.91
19-antiprism	38	76	40	456	58,059,628,319,357,318	10,550,126,657,845,736	18.17

Table 2: The number and percentage of non-overlapping lattice unfoldings for cuboids (excerpt).

					Faces-in-touch			Edges-in-touch			Vertices-in-touch		
Cuboids	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert F\rvert$	# (Lattice unfoldings)	#(MOPUs)	# (No(faces) unfoldings)	Pct.(%)	#(MOPUs)	# (No(edges) unfoldings)	Pct.(%)	#(MOPUs)	# (No(vertices) unfoldings)	Pct.(%)
$(1,1,1)$	8	12	6	384	0	384	100.00	0	384	100.00	0	384	100.00
$(1,1,2)$	12	20	10	12,124	0	12,124	100.00	0	12,124	100.00	32	11,484	94.72
$(1,1,3)$	16	28	14	240,304	16	240,240	99.97	80	238,432	99.22	304	212,920	88.60
$(1,1,4)$	20	36	18	3,708,380	80	3,705,820	99.93	512	3,644,600	98.28	1,232	3,075,400	82.93
$(1,1,5)$	24	44	22	49,206,176	208	49,156,592	99.90	1,504	47,970,720	97.49	3,408	38,043,936	77.32
$(1,1,6)$	28	52	26	592,188,796	464	591,487,340	99.88	3,808	573,122,568	96.78	8,448	424,509,028	71.68
$(1,1,7)$	32	60	30	6,671,469,328	1,104	6,663,017,440	99.87	9,360	6,409,933,496	96.08	20,432	4,407,661,888	66.07
$(1,1,8)$	36	68	34	71,772,242,780	2,704	71,679,140,716	99.87	22,912	68,429,543,676	95.34	49,456	43,445,829,708	60.53
$(1,1,9)$	40	76	38	747,116,459,968	6,544	746,143,953,328	99.87	55,584	706,395,487,984	94.55	119,504	412,096,369,696	55.16
$(1,1,10)$	44	74	42	7,593,452,118,844	15,760	7,583,621,450,924	99.87	134,368	7,114,772,651,372	93.70	288,416	3,797,487,539,408	50.01
$(1,2,2)$	18	32	16	1,675,184	0	1,675,184	100.00	32	1,553,536	92.74	128	1,228,824	73.35
$(1,2,3)$	24	44	22	131,478,632	544	130,212,292	99.04	1,648	111,177,796	84.56	3,312	75,653,292	57.54
$(1,2,4)$	30	56	28	7,692,072,382	14,920	7,528,985,598	97.88	32,048	5,970,306,978	77.62	52,960	3,535,269,930	45.96
$(1,2,5)$	36	68	34	375,631,947,892	141,816	364,028,460,124	96.91	291,736	270,654,176,916	72.05	449,552	140,837,624,986	37.49
$(2,2,3)$	34	64	32	203,758,066,112	5,824	196,470,177,268	96.42	19,392	109,840,848,592	53.91	34,704	48,990,450,676	24.04
$(\sqrt{2},2\sqrt{2},2\sqrt{2})$	34	64	32	207,761,826,744	13,296	198,307,283,288	95.45	45,776	135,619,116,108	65.28	76,432	67,737,527,156	32.60

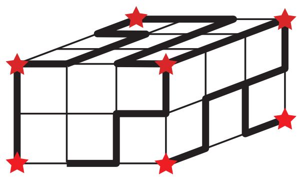
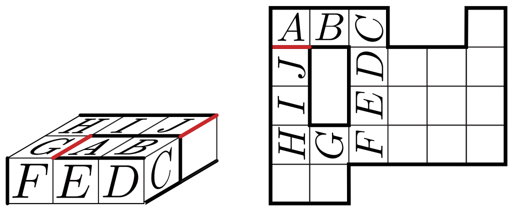
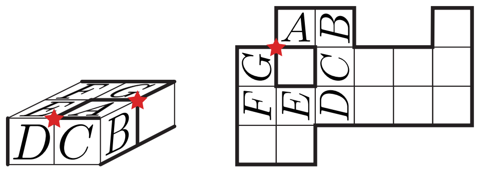


Figure 5: An example of a cutting line in a $(2,3,3)$-cuboid. The cutting line forms a tree that includes all eight lattice cuboid vertices (the starred ones).



(a) Edges-in-touch unfolding in the $(1,2,3)$-cuboid



(b) Vertices-in-touch unfolding in the $(1,2,2)$-cuboid

Figure 6: Overlapping lattice unfolding [10, 14]. In the cuboids shown in (a) and (b), the red edges and the starred vertices, respectively, are not in touch.

In a lattice unfolding, the original cuboid's unit squares are arranged on a plane, with their edges connected. The relationship between any pair of unit squares that are not adjacent on the original cuboid is classified as follows [14]:

(1) Overlap in the same position (Figure 2).

(2) Share one edge (Figure 6 (a)).

(3) Share one vertex (Figure 6 (b)).

(4) Do not share any edges or vertices.

Herein, we say that a lattice unfolding is *faces-in-touch*, *edges-in-touch*, or *vertices-in-touch* if it has a pair of unit squares satisfying condition (1), (2), or (3), respectively. If any of the conditions (1)-(3) is satisfied, we say the *lattice unfolding is overlapping*. Otherwise, when all pairs of unit squares that are not adjacent on the original cuboid satisfy condition (4), we say the *lattice unfolding is non-overlapping*. Note that for any cuboid,

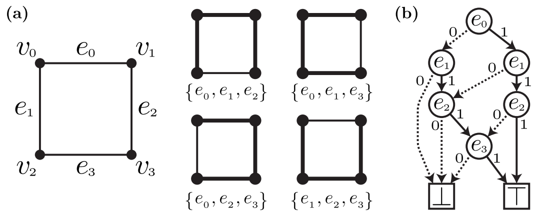

(a)
v_0 e_0 v_1
e_1 e_2
v_2 e_3 v_3
$\{e_0,e_1,e_2\}$ $\{e_0,e_1,e_3\}$
$\{e_0,e_2,e_3\}$ $\{e_1,e_2,e_3\}$
(b)

Figure 7: (a) The graph C_4 and its spanning trees. (b) A ZDD representing the spanning trees of C_4.

the following strict inclusion relations hold among the families of unfoldings: $\mathcal{U}_f \subset \mathcal{U}_e \subset \mathcal{U}_v$, where $\mathcal{U}_f$, $\mathcal{U}_e$, and $\mathcal{U}_v$ denote the sets of unfoldings with face-in-touch, edge-in-touch, and vertex-in-touch, respectively.

2.3 Counting the number of unfoldings

From Lemmas 1 and 2, the number of unfoldings (including those with overlaps) is equal to the number of trees satisfying specific conditions on the candidate cutting lines (hereafter called *cutting trees*). One method for counting cutting trees is using a *Zero-suppressed Decision Diagram (ZDD)* [5]. A ZDD is a data structure that compactly represents a family of sets as a directed acyclic graph [9] (see the example in Figure 7). It consists of two types of nodes: *terminal nodes* with out-degree zero ($\top$, $\bot$), and *branching nodes* labeled by elements of the set. Each branching node has two outgoing edges: a 1-*edge*, which indicates the inclusion of the labeled element, and a 0-*edge*, which indicates its exclusion. A ZDD has a unique *root node* with no incoming edges, and each path from the root to $\top$ corresponds to a specific set.

3 Counting algorithm for the number of non-overlapping unfoldings

In this section, we present an algorithm for counting non-overlapping unfoldings. Section 3.1 introduces the notion of *minimal overlapping partial unfoldings (MOPU)*, which serve as the basis for our algorithm. In Section 3.2, we present a ZDD-based algorithm for counting only non-overlapping unfoldings.

3.1 Minimal overlapping partial unfoldings

We begin by introducing several notions used to define minimal overlapping partial unfoldings. Let Q be a polyhedron. Two faces in Q are *adjacent* if they are connected through a common cutting line. The *dual graph* of Q is a graph $G_D = (V_D, E_D)$, where each vertex in V_D corresponds to a face of Q, and two vertices are connected by an edge in E_D if and only if the corresponding faces are adjacent. A *partial unfolding* is a flat polygon consisting of a set of faces that correspond to a connected induced subgraph of G_D. A *minimal partial unfolding* is a partial unfolding consisting of the faces along a simple path between two vertices in G_D. A *minimal overlapping partial unfolding (MOPU)* is a minimal partial unfolding in which the two end faces overlap. Figure 8 shows examples of MOPUs in edge unfoldings. In each case, the two end faces overlap, and removing any face would break the connectivity of the unfolding, which makes them minimal. The partial unfoldings consisting of the lettered faces in Figures 2 and 6 are also examples of MOPUs in lattice unfoldings. MOPUs in convex regular-faced polyhedra and lattice cuboids can be enumerated using the rotational unfolding [14,15].

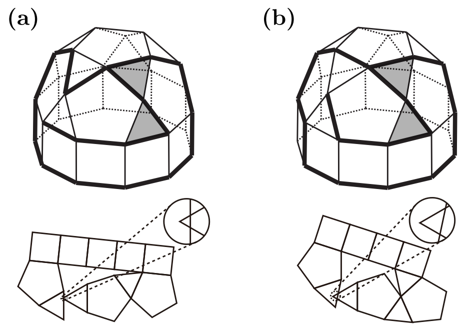


Figure 8: Examples of MOPUs in a convex regular-faced polyhedron (J21). The gray faces indicate the end faces.

3.2 ZDD-based algorithm

In this section, we describe an algorithm for counting the number of non-overlapping unfoldings of a given polyhedron. As described in Section 2.3, the number of cutting trees can be counted by constructing a ZDD $\mathcal{Z}_{\mathcal{T}}$. However, some cutting trees result in overlapping unfoldings, depending on the shape of the polyhedron. In such cases, $\mathcal{Z}_{\mathcal{T}}$ includes these overlapping unfoldings. To efficiently remove overlapping unfoldings from $\mathcal{Z}_{\mathcal{T}}$, we use *subsetting*, a ZDD operation [8]. The subsetting technique constructs a new ZDD $\mathcal{Z}_{\mathcal{N}}$ by extracting the family of sets that satisfy the constraint $\mathcal{C}$ from ZDD $\mathcal{Z}$.

We now present a method for removing overlapping unfoldings by subsetting. Let U be a partial unfolding, and let $NC[U]$ be the set of edges that are not cut when unfolding the polyhedron. We denote by k the number of MOPUs, enumerated by the rotational unfolding. The following lemma holds for any MOPU M_i ($1 \le i \le k$).

Lemma 3 *If an unfolding U satisfies $NC[M_i] \subseteq NC[U]$, then U is an overlapping unfolding.*

Proof. Let the sequence of faces in MOPU M_i be $\langle f_1, f_2, \ldots, f_\ell \rangle$, and let e_j be the edge shared between each pair of adjacent faces f_j and f_{j+1} (where the faces

f_1 and f_ℓ overlap). Since $NC[M_i]$ represents the set of uncut edges in the partial unfolding M_i, we can write $NC[M_i] = \{e_1, e_2, \dots, e_{\ell-1}\}$. On the other hand, from the condition $NC[M_i] \subseteq NC[U]$, it follows that the set $\{e_1, e_2, \dots, e_{\ell-1}\}$ must be included in the unfolding U. Therefore, the sequence of faces $\langle f_1, f_2, \dots, f_\ell \rangle$ appears in U, indicating that U has overlaps. □

From Lemma 3, removing the family of sets $\mathcal{U}_i$ (which represents unfoldings containing the MOPU M_i) from the ZDD $\mathcal{Z}_\mathcal{T}$ yields a ZDD that represents only unfoldings that do not include the structure of M_i. On the other hand, to construct the family of sets $\mathcal{U}_i$ representing unfoldings that include MOPU M_i, we need a ZDD representing the family of cutting trees that contain $NC[M_i]$. However, by applying the following lemma, we can avoid constructing a ZDD representing all cutting trees containing $NC[M_i]$, and instead create a simpler ZDD.

Lemma 4 *Given the family of sets $\mathcal{Z}_\mathcal{T}$ representing all unfoldings, the following conditions are equivalent:*

(1) *The family of sets obtained by removing the unfoldings that include MOPU M_i from $\mathcal{Z}_\mathcal{T}$.*

(2) *The family of sets obtained by removing the family $\mathcal{F}_i = \{NC[M_i] \cup E' \mid E' \subseteq E \setminus NC[M_i]\}$, which contains all subsets that include $NC[M_i]$, from $\mathcal{Z}_\mathcal{T}$.*

Proof. From the condition, we know that $\mathcal{U}_i \subseteq \mathcal{F}_i$. Now, if we define $\mathcal{N}_i = \mathcal{F}_i \setminus \mathcal{U}_i$, then $\mathcal{N}_i$ contains no sets that represent unfoldings, meaning $\mathcal{N}_i \not\subseteq \mathcal{Z}_\mathcal{T}$. Therefore, we have the following equivalence:

$$\mathcal{Z}_\mathcal{T} \setminus \mathcal{F}_i = \mathcal{Z}_\mathcal{T} \setminus (\mathcal{N}_i \cup \mathcal{U}_i) = \mathcal{Z}_\mathcal{T} \setminus \mathcal{U}_i,$$

which completes the proof. □

Note that the removal procedure based on Lemma 4 is order-independent: if an unfolding contains the uncut edge set of any MOPU M_i, it will be excluded at that step, regardless of whether it also contains other $NC[M_j]$. Therefore, the resulting ZDD correctly retains only non-overlapping unfoldings.

Therefore, we can construct a ZDD that represents non-overlapping unfoldings by following three steps:

Step 1. Enumerate MOPUs using rotational unfolding [14, 15].

Step 2. Construct the ZDD $\mathcal{Z}_\mathcal{T}$ that represents all possible unfoldings, and for each i $(1 \le i \le k)$, construct a ZDD $\mathcal{F}_i$ representing the family of all sets that do not simultaneously contain all elements of $NC[M_i]$.

Step 3. Apply the subsetting technique [8] on $\mathcal{Z}_\mathcal{T}$ using the constraints from each $\mathcal{F}_i$, to construct a ZDD $\mathcal{Z}_\mathcal{N}$ that excludes MOPUs M_1 through M_k.

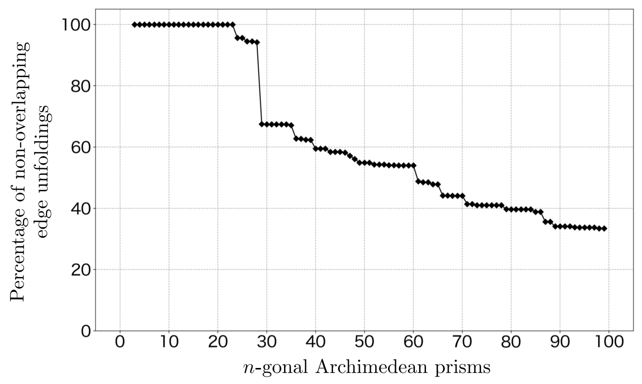


Figure 9: The percentage of non-overlapping edge unfoldings in Archimedean prisms.

4 Computational experiments

Here, we present the results of applying the algorithm for counting non-overlapping unfoldings to the edge unfoldings of 175 convex regular-faced polyhedra and the lattice unfoldings of 23 cuboids. We used the TdZdd library[1] to construct the ZDD $\mathcal{Z}_\mathcal{T}$, which represents all unfoldings, the ZDD $\mathcal{F}_i$, which represents the family of sets containing all elements of $NC[M_i]$, and applied the subsetting method. To enumerate MOPUs for the convex regular-faced polyhedra and lattice cuboids, we used rotational unfolding[2] [14, 15]. All computational experiments were conducted on an Intel(R) Xeon(R) CPU E5-2643 v4 at 3.40 GHz with 512 GB of memory, running CentOS 7.9[3].

4.1 The number of non-overlapping edge unfoldings for convex regular-faced polyhedra

As introduced in Section 1, Table 1 shows the number and percentage of non-overlapping edge unfoldings for selected convex regular-faced polyhedra[4]. Figures 9 and 10 show line plots of the percentage of non-overlapping edge unfoldings for Archimedean prisms and antiprisms, respectively. In each figure, the horizontal axis indicates the number of sides n (or m) of the base polygon, and the vertical axis shows the percentage of non-overlapping edge unfoldings.

From the results of these experiments, we can find the following. Let us compare the truncated icosahedron and the truncated dodecahedron in Archimedean solids (see Table 1, under Archimedean solids). Both polyhedra have the same number of vertices, edges, and faces,

[1]`https://github.com/kunisura/TdZdd`

[2]`https://shiotatakumi.github.io/MyPage/library/RotationalUnfolding.html`

[3]The source code and details on how it was tested can be found at `https://shiotatakumi.github.io/MyPage/library/CountingNonoverlappingUnfoldings.html`.

[4]The full version can be found in `https://shiotatakumi.github.io/MyPage/contents/250813-CCCG-2025.html#A`

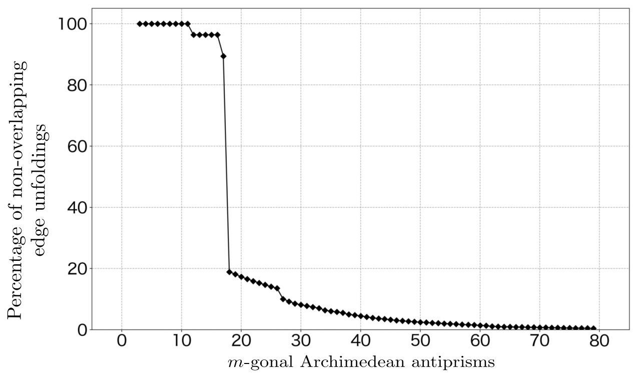


Figure 10: The percentage of non-overlapping edge unfoldings in Archimedean antiprisms.

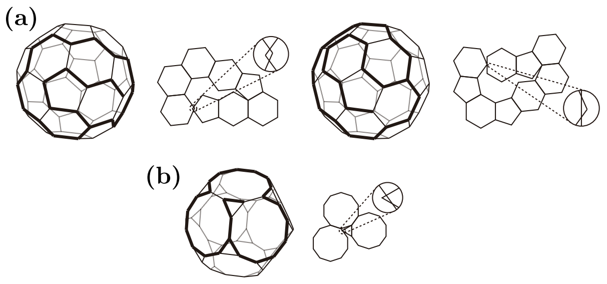


Figure 11: MOPUs in (a) Truncated icosahedron and (b) Truncated dodecahedron, excluding rotational and mirror equivalents [7,15].

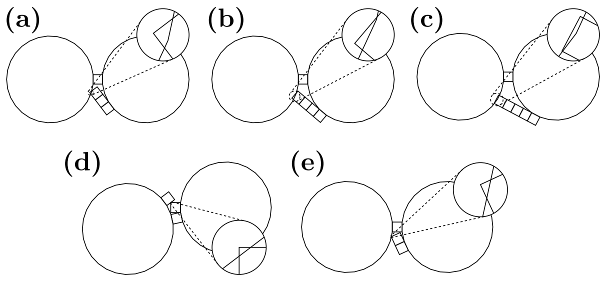


Figure 12: MOPUs in Archimedean prisms: (a)-(c) for 28-gonal and (a)-(e) for 29-gonal prisms, excluding rotational and mirror equivalents [15]. (a)-(c) consist of 6, 7, and 8 faces, respectively, and (d) and (e) each consist of 5 faces.

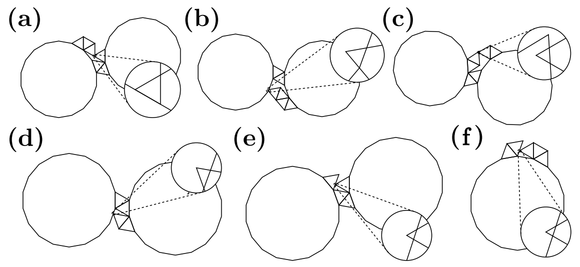


Figure 13: MOPUs in Archimedean antiprisms: (a)-(c) for 17-gonal and (a)-(f) for 18-gonal antiprisms, excluding rotational and mirror equivalents [15]. (a)-(c) each consist of 8 faces, and (d)-(f) each consist of 6 faces.

but the truncated icosahedron has more MOPUs than the truncated dodecahedron. However, the truncated dodecahedron has a lower percentage of non-overlapping edge unfoldings. The MOPUs of the truncated icosahedron consist of eight or nine faces (Figure 11 (a)), whereas the truncated dodecahedron has a MOPU composed of only four faces (Figure 11 (b)). Thus, we can observe that the presence of MOPUs composed of fewer faces has a greater influence on reducing the percentage of non-overlapping edge unfoldings than the number of MOPUs.

This trend is also observed in Archimedean n-gonal prisms and m-gonal antiprisms. In both cases, the percentage of non-overlapping edge unfoldings significantly decreases at $n = 29$ and $m = 18$, respectively, coinciding with the appearance of MOPUs composed of fewer faces (see Figure 9 and 10; see also Table 1, under Archimedean prisms and antiprisms). Figures 12 and 13 show how the structure of MOPUs changes around the point where the percentage decreases.

4.2 The number of non-overlapping lattice unfoldings for cuboids

Table 2, which also appears in Section 1, shows the number and percentage of non-overlapping lattice unfoldings in selected (xL, yL, zL)-cuboids that do not have faces-in-touch, edges-in-touch, or vertices-in-touch, respectively[5]. In what follows, we use "No(xx)" to denote a lattice unfolding that does not have any xx-in-touch.

From the results for $(1, 1, z)$-cuboids $(1 \leq z \leq 10)$, we observe that the percentage of non-overlapping lattice unfoldings tends to decrease as z increases. Similar trends are also observed in other results.

Among lattice cuboids, there exist cuboids that have the same surface area but different volumes. Tables 3 and Table 4 show the percentages of lattice unfoldings without face-, edge-, or vertex-in-touch for cuboids with a surface area of 32 and 34, respectively. These tables indicate that the percentages of No(edges) and No(vertices) unfoldings tend to decrease as the volume increases. However, despite the increasing volume, the percentage of No(faces) unfoldings decreases by approximately 1% at surface area 32, but increases by about 3% at 34. Thus, while a larger volume tends to lower the percentage of non-overlapping lattice unfoldings, volume alone does not fully explain the trend[6].

[5]The full version can be found in https://shiotatakumi.github.io/MyPage/contents/250813-CCCG-2025.html#B

[6]Similar trends are also observed for cuboids with other surface areas; see https://shiotatakumi.github.io/MyPage/contents/250813-CCCG-2025.html#C

Table 3: The percentage of non-overlapping lattice unfoldings for a cuboid with a surface area of 32.

Cuboids	Volume	No(faces)	No(edges)	No(vertices)
$(\sqrt{2}, 2\sqrt{2}, 2\sqrt{2})$	$8\sqrt{2}$	95.45	65.28	32.60
$(2, 2, 3)$	12	96.42	53.91	24.04

Table 4: The percentage of non-overlapping lattice unfoldings for a cuboid with a surface area of 34.

Cuboids	Volume	No(faces)	No(edges)	No(vertices)
$(1, 1, 8)$	8	99.87	95.34	60.53
$(1, 2, 5)$	10	96.91	72.05	37.49

5 Conclusion

In this paper, we proposed an algorithm to count the number of non-overlapping unfoldings using ZDDs and MOPU-based operations. We applied the algorithm to two types of unfoldings and obtained insights into the structural conditions under which the number of non-overlapping unfoldings decreases.

There are four main directions for future work. The first is to investigate the underlying reason for the observation in Section 4.2, where the percentage of No(faces) unfoldings does not decrease monotonically with increasing volume for cuboids with the same surface area. The second is to apply the proposed method to other classes of general unfoldings, such as pseudo-edge unfoldings [1]. The third is to investigate whether the proposed method can be extended to enumerate non-overlapping unfoldings of non-convex polyhedra. Since the method treats a polyhedron as a graph and does not depend on its geometric shape, it is expected to naturally generalize to non-convex cases. In this type of unfolding, the vertices correspond to the original vertices of the polyhedron, the edges are shortest geodesic paths between vertex pairs, and the surface is unfolded by cutting along these paths. The fourth is to pursue a theoretical analysis of the computational complexity of the proposed method. However, since the following problem remains open, providing a formal runtime bound is difficult:

Open problem 5 *Is it computationally hard to decide whether a given unfolding has overlaps?*

Addressing this problem would naturally precede any formal analysis of the method. These directions may provide new insights toward resolving Conjecture 1.

Acknowledgments.

This work was supported in part by JSPS KAKENHI Grants JP24KJ1816, JP22H03549, and JP20K11673, and by MEXT KAKENHI Grant JP20H05964.

References

[1] N. Barvinok and M. Ghomi. Pseudo-Edge Unfoldings of Convex Polyhedra. *Discrete Comput. Geom.*, 64(3):671–689, 2020.

[2] H. T. Croft, K. J. Falconer, and R. K. Guy. *Unsolved Problems in Geometry.* Springer-Verlag, reissue edition, 1991.

[3] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge University Press, 2007.

[4] K. Hirose. Hanseitamentai no tenkaizu no kasanari ni tsuite (On the overlap of Archimedean solids), in Japanese, 2015. Saitama Univ. graduation thesis.

[5] T. Horiyama, M. Miyasaki, and R. Sasaki. Isomorphism Elimination by Zero-Suppressed Binary Decision Diagrams. In *30th CCCG*, 2018.

[6] T. Horiyama and W. Shoji. Edge Unfoldings of Platonic Solids Never Overlap. In *23rd CCCG*, 2011.

[7] T. Horiyama and W. Shoji. The Number of Different Unfoldings of Polyhedra. In *24th ISAAC*, volume 8283 of *LNCS*, pages 623–633. Springer, 2013.

[8] H. Iwashita and S. Minato. Efficient Top-Down ZDD Construction Techniques Using Recursive Specifications. Technical Report TCS-TRA-1369, Hokkaido Univ., 2013.

[9] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *30th Design Automation Conf. (DAC)*, pages 272–277. ACM Press, 1993.

[10] J. Mitani and R. Uehara. Polygons Folding to Plural Incongruent Orthogonal Boxes. In *20th CCCCG*, 2008.

[11] M. Namiki and K. Fukuda. Unfolding 3-dimensional convex polytopes, 1993. A package for Mathematica 1.2 or 2.0, Mathematica Notebook.

[12] C. A. Schevon. Algorithms for geodesics on convex polytopes, 1989. Johns Hopkins University, Master's thesis.

[13] G. C. Shephard. Convex polytopes with convex nets. *Mathematical Proceedings of the Cambridge Philosophical Society*, 78(3):389–403, 1975.

[14] T. Shiota, T. Kamata, and R. Uehara. Overlapping of lattice unfolding for cuboids. *IEICE Trans. Inf. Syst.*, E108.D(3):252–259, 2025.

[15] T. Shiota and T. Saitoh. Overlapping edge unfoldings for convex regular-faced polyhedra. *Theor. Comput. Sci.*, 1002:114593, 2024.

On the Diameters of Reconfiguration Graphs and ZDD-Based BFS-Algorithm for Optimal Reconfiguration Problems of Optimal Ladder Lotteries

Shoon Mineyoshi* Kazuhisa Seto† Takashi Horiyama†

Abstract

A ladder lottery is a network corresponding to a permutation. We investigate the reconfiguration problems and the antipode distance problems (i.e., finding the farthest instances) of optimal ladder lotteries. In this paper, we propose an algorithm for solving these problems using ZDDs. We also show the diameters of the reconfiguration graphs, along with pairs of ladder lotteries that realize the diameters.

1 Introduction

Reconfiguration problems have a common structure: fix a combinatorial problem $\mathscr{C}$ and define an adjacency relation on the set of feasible solutions for an instance I of $\mathscr{C}$. The reconfiguration problem of $\mathscr{C}$ is, given an instance I and its two feasible solutions S and T, to determine whether there exists a reconfiguration sequence $\langle S_0, S_1, ..., S_m\rangle$ of feasible solutions such that (1) $S_1 = S$, $S_m = T$ and (2) S_{i-1} and S_i are adjacent for all $i = 1, 2, ..., m$. This can be viewed as the problem of finding a path from S to T in the reconfiguration graph, whose vertices correspond to the feasible solutions of I, and whose edges connect adjacent solutions under the adjacency relation. Note that the graph is implicitly defined and not given explicitly.

Regarding computational complexity, most reconfiguration problems are known to be PSPACE-complete. For example, the reconfiguration problem for graph k-colorings ($k \geq 4$) [1] and that for SAT [5] are both PSPACE-complete. Ito et al. [6] further showed that several other reconfiguration problems are PSPACE-complete via reductions from the reconfiguration problem of 3SAT. On the other hand, some reconfiguration problems are solvable in polynomial time. For example, the reconfiguration problem for graph 3-colorings [2] is in P. Among these tractable cases, the reconfiguration problem of optimal ladder lotteries is particularly interesting, as it is also in P due to the fact that a reconfiguration sequence always exists for any pair of feasible solutions.

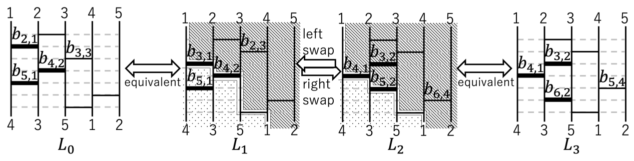


Figure 1: Modifications on a ladder.

A *ladder lottery* (*ladder* for short) is a network consisting of vertical lines (*lines* for short) and horizontal bars (*bars* for short). Each bar corresponds to an *adjacent transposition*, and a ladder represents a permutation. For example, ladders in Fig. 1 represent the permutation (4, 3, 5, 1, 2), where the permutation is determined by the order of the bottom ends of lines. A ladder L of a permutation π is said to be *optimal* if L contains the minimum number of bars among all ladders of π. A ladder is closely related to an arrangement of pseudolines, or equivalently, a tangle. In both cases, each bar corresponds to an intersection of two curves—referred to as pseudolines [14] or wires [4], respectively. Ladders also have a close connection to primitive sorting networks, where comparators (instead of bars) sort two elements. Knuth [8] studied the counting of primitive sorting networks.

In this paper, we investigate the reconfiguration problem of optimal ladders. Two optimal ladders L and L' are adjacent if L' can be obtained from L via a local modification called a *local swap*. The problem is, given a permutation π and two optimal ladders L, L' of π, to determine whether there exists a reconfiguration sequence from L to L'. From a combinatorial perspective, Yamanaka et al. [14] proposed an algorithm for enumerating all optimal ladders of π by defining a tree structure over the optimal ladders, where edges correspond to local swaps. While this tree always gives a reconfiguration sequence between any two optimal ladders, the sequence is not necessarily *optimal* (i.e., shortest). We focus on the *optimal reconfiguration sequences* between two optimal ladders and address the following three problems: (1) the optimal reconfiguration problem, (2) the antipode distance problem, and (3) the diameter of the reconfiguration graph.

*Graduate School of Information Science and Technology, mineyoshi.sean.t6@elms.hokudai.ac.jp

†Faculty of Information Science and Technology, {seto, horiyama}@ist.hokudai.ac.jp

The optimal reconfiguration problem is, given π and two optimal ladders L, L' of π, to find the distance between them—that is, the optimal (minimum) length of a reconfiguration sequence from L to L'. Horiyama et al. [13] showed that the distance between two ladders can be measured by the *reverse triple*, which is the difference in the positions of bars. Based on this notion, they also proposed a polynomial-time algorithm to find an optimal reconfiguration sequence from L to L' by greedily swapping reverse triples until reaching L'.

The antipode distance problem is, given π and an optimal ladder L of π, to find the distance to its antipode. Here, L' is called *antipode* of L if the distance from L to L' is maximized over all optimal ladders of π. Intuitively, the antipode of L is the ladder that lies furthest from L in the reconfiguration graph. This type of problem is also studied in the context of puzzles, such as the Rubik's Cube and the 15-puzzle. In such puzzles, an initial configuration is given, and the goal is to reach a target configuration through a sequence of small modifications. A natural question in this context is: what is the maximum distance from the target configuration to any other reachable configuration? For the $3 \times 3 \times 3$ Rubik's Cube, the antipode distance was shown to be exactly 20 [10], after decades of investigation. For the generalized $n \times n \times n$ Rubik's Cube, the antipode distance is $\theta(n^2/\log n)$ [3].

Our first contribution is to propose a new approach for the antipode distance problem of optimal ladders, which also solves the optimal reconfiguration problem. The most straightforward method for determining the antipode distance is to construct the reconfiguration graph based on adjacency relations and then conduct a breadth-first search. This approach treats each vertex (ladder) individually. In contrast, the ZDD-based framework [7] enables the simultaneous exploration of multiple vertices without explicitly constructing the graph. Our algorithm is based on this framework, which was originally proposed for the reconfiguration of independent sets and some other problems. A ZDD [9] is a directed acyclic graph representing a family of sets. Thanks to the compactness of ZDDs—often significantly smaller than the families they represent—, this approach achieved strong empirical performance on instances with large reconfiguration graphs in the CoRe Challenge [11,12]. In our setting, a ladder is represented as a set of its bars, and a local swap is realized as an operation on ZDDs. Our implementation demonstrates that the algorithm effectively utilizes the compactness of ZDDs, particularly their ability to efficiently represent families of ladders.

Our second contribution is to show the diameter of the reconfiguration graph of optimal ladders and to provide a pair of optimal ladders that achieves the diameter. Formally, given a permutation π, the diameter (problem) asks for the maximum distance between any two optimal ladders of π. The diameter of the reconfiguration graphs has been studied in both algorithmic and puzzle-related contexts. For example, the diameter of the reconfiguration graph for 3-colorings of graph $G = (V, E)$ is $O(|V|^2)$ [2]. Unfortunately, however, for many reconfiguration problems, the upper and lower bounds of the diameter of their reconfiguration graphs remain unknown. We resolve the diameter of the reconfiguration graphs of optimal ladders using a lemma in [13], which states that, for any permutation π, the distance between two optimal ladders of π is equal to the number of reverse triples. To obtain the diameter, we focus on a pair of optimal ladders called left-stacked and right-stacked brick optimal ladders, which are uniquely defined for each π [14]. We prove that the number of reverse triples between this pair is maximum among all pairs of optimal ladders of π.

2 Preliminaries

2.1 Optimal Ladder Lotteries

A *ladder lottery* (*ladder* for short) is a network consisting of n (> 0) vertical lines (*lines* for short) and ℓ (≥ 0) horizontal bars (*bars* for short). Each bar connects two consecutive lines (see, e.g., L_0 in Fig. 1). We divide a ladder L with ℓ bars into ℓ horizontal layers, with each layer containing exactly one bar. Let $b_{i,k}$ denote the bar on the i-th layer from the top that connects the k-th and $(k+1)$-th lines from the left, where $1 \leq i \leq \ell$ and $1 \leq k < n$. We represent a ladder L as the set of its bars. Each bar $b_{i,k}$ exchanges the two elements on the k-th and $(k+1)$-th lines from the left. Thus, a ladder L defines a permutation π such that the order of the elements at the bottom ends of the lines corresponds to π. A ladder L of π is *optimal* if L contains the minimum number of bars among all ladders of π.

We next describe modifications that remove and add bars in a ladder. In some cases, different sets of bars represent essentially the same (i.e., equivalent) ladder. For example, the ladders L_0 and L_1 in Fig. 1 are equivalent although $b_{2,1}, b_{3,3}$ appear only in L_0. In general, for some $1 \leq i < \ell$ and $1 \leq k, k' < n$ with $|k - k'| \geq 2$, two ladders L and L' are equivalent if L' is obtained from L by removing two bars $b_{i,k}, b_{i+1,k'}$ and adding $b_{i+1,k}, b_{i,k'}$. That is, $L' = L \cup \{b_{i+1,k}, b_{i,k'}\} \setminus \{b_{i,k}, b_{i+1,k'}\}$. We call this modification a *layer swap*. Note that any ladder obtained from L by applying a sequence of zero or more layer swaps is also equivalent to L. This equivalence relation defines the equivalence class $[L]$ of L. We extend this notation to a family $\mathcal{F}$ of ladders by defining $[\mathcal{F}] = \bigcup_{L \in \mathcal{F}} [L]$.

A *local swap* is a local modification involving only three bars that preserves the permutation. It comes in two forms: a *right swap* and a *left swap*. To describe a local swap, we begin with a *local swap with adjacent layers*, which is an operation on the following two sets of bars. For indices $1 \leq i \leq \ell - 2$ and $1 \leq k \leq n-2$, we define the set of three bars on three adjacent layers as $B_>(b_{i,k}) = \{b_{i,k}, b_{i+1,k+1}, b_{i+2,k}\}$. Similarly, we define $B_<(b_{i,k}) = \{b_{i,k+1}, b_{i+1,k}, b_{i+2,k+1}\}$, where the roles of k and $k+1$ are swapped. A *right swap with adjacent layers* is the operation that removes the three bars in $B_>(b_{i,k})$ and adds those in $B_<(b_{i,k})$. A *left swap with adjacent layers* does the reverse. For example, by applying a left swap with adjacent layers to the bars $b_{3,2}$, $b_{4,1}$, and $b_{5,2}$ in L_2 (bold bars in Fig. 1), we obtain L_1. Suppose that a local swap with adjacent layers is not directly applicable to ladders L and L'. Even in such a case, it may still be applicable between equivalent ladders of L and L'. We define such a swap to be a local swap from L to L' if there exist $\hat{L} \in [L]$ and $\hat{L}' \in [L']$ such that $\hat{L}'$ is obtained from $\hat{L}$ by applying a local swap with adjacent layers. An optimal ladder L of π is called *left-stacked brick* (resp., *right-stacked brick*) if no left (resp., right) swap can be applied to L. Both the left-stacked and right-stacked bricks are uniquely determined for each permutation π [14].

The distance between two ladders L and L' is the minimum number of local swaps to obtain L' from L, and it is equal to the number of *reverse triples* between them [13]. We denote by $R_\ell(L, z)$ and $R_r(L, z)$ the left and right regions of the route where element $z \in \{1, 2, ..., n\}$ goes through in a ladder L, respectively. For example, in L_1 in Fig. 1, the dotted and striped regions correspond to $R_\ell(L_1, 1)$ and $R_r(L_1, 1)$, respectively. For three elements $x, y, z \in \{1, 2, ..., n\}$, let $\{x, y\}$ be the bar that exchanges x, y. $\{x, y, z\}$ is a reverse triple[1] if (1) both L and L' contain the bar $\{x, y\}$, and (2) either $\{x, y\}$ is in both $R_r(L, z)$ and $R_\ell(L', z)$ or $\{x, y\}$ is in both $R_\ell(L, z)$ and $R_r(L', z)$. For example, see L_1, L_2 in Fig. 1. Both a bar $b_{5,1}$ in L_1 and a bar $b_{3,2}$ in L_2 exchange 3 and 4. In this case, $\{1, 3, 4\}$ is a reverse triple because $b_{5,1}$ is in $R_\ell(L_1, 1)$ and $b_{3,2}$ is in $R_r(L_2, 1)$. Let $\mathrm{Diff}_\pi(L, L')$ be the family of all reverse triples between L, L'. By applying a local swap to L, $|\mathrm{Diff}_\pi(L, L')|$ decreases or increases by 1, which derives the following lemma.

Lemma 1 ([13]) *Let L, L' be two optimal ladders of π. Then, the distance between L and L' is equal to the number of reverse triples between them.*

2.2 ZDD Framework for Reconfiguration Problems

Let $X = \{x_1, x_2, ...\}$ be a totally ordered set such that $x_1 \prec x_2 \prec \cdots$. A ZDD (Zero-suppressed Binary Decision Diagram) is a directed acyclic graph representing a family $\mathcal{F} \subseteq 2^X$ [9]. It has two *terminal* nodes of outdegree zero, labeled 0 and 1. The other nodes are called *non-terminal* nodes. Each non-terminal node v has outdegree two and is labeled with an element $\ell(v) \in X$. It has two outgoing arcs, called 0-arc and 1-arc. If these arcs point to nodes v_0 and v_1, respectively, and $x_i = \ell(v)$, we write $v = (x_i, v_0, v_1)$ (see Fig. 2). We require that $\ell(v) \prec \ell(v_0)$ and $\ell(v) \prec \ell(v_1)$ hold whenever v_0 and v_1 are non-terminal nodes. A ZDD has exactly one node with indegree zero called the *root*.

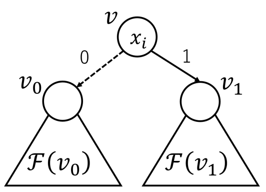


Figure 2: Node v representing $\mathcal{F}(v_0) \cup (\{\{x_i\}\} \bowtie \mathcal{F}(v_1))$.

For a family $\mathcal{F}$ and an element $x_i \in X$, let $\{\{x_i\}\} \bowtie \mathcal{F}$ denote $\{S \cup \{x_i\} \mid S \in \mathcal{F}\}$. In a ZDD, each node v represents a family of sets denoted by $\mathcal{F}(v)$. The terminal nodes 0 and 1 represent $\{\ \}$ and $\{\emptyset\}$, respectively. A non-terminal node $v = (x_i, v_0, v_1)$ represents $\mathcal{F}(v) = \mathcal{F}(v_0) \cup (\{\{x_i\}\} \bowtie \mathcal{F}(v_1))$. The family represented by a ZDD is defined as that represented by its root node. We denote by $\mathcal{Z}(\mathcal{F})$ the ZDD representing the family $\mathcal{F}$.

Kawahara et al. [7] proposed a ZDD-based framework for reconfiguration problems. Let X be a finite universal set, and I be an instance of a reconfiguration problem. This framework applies when feasible solutions of I can be represented as subsets of X. Given I and two feasible solutions S and T, the framework determines whether there exists a reconfiguration sequence from S to T, using a ZDD to represent a family of feasible solutions.

In this framework, we perform a breadth-first search on the reconfiguration graph (which is not explicitly given). For a feasible solution S', let $N(S')$ denote the set of its neighbors in the reconfiguration graph. Similarly, for a family $\mathcal{F}$ of feasible solutions, we define $N(\mathcal{F})$ as $\bigcup_{S' \in \mathcal{F}} N(S')$. Let $\mathcal{F}^S_d$ denote the family of feasible solutions at distance d from S in the reconfiguration graph, with $\mathcal{F}^S_0 = S$. Since $N(\mathcal{F}^S_d)$ may include feasible solutions at distance $d-1$, d, or $d+1$ from S, we have the following recurrence:

$$\mathcal{F}^S_{d+1} = N(\mathcal{F}^S_d) \setminus (\mathcal{F}^S_d \cup \mathcal{F}^S_{d-1}) \quad (1)$$

Thus, by designing an algorithm to compute $N(\mathcal{F}^S_d)$, we can iteratively obtain $\mathcal{F}^S_d$ for $d = 1, 2, ...$ This computation is performed as set operations on ZDDs, without processing individual solutions one by one. At each iteration, by examining $N(\mathcal{F}^S_d)$, we determine whether a reconfiguration sequence from S to T exists: If $T \in \mathcal{F}^S_d$,

[1]Yamanaka et al. [13] defined a reverse triple as an ordered triple $(x, y; z)$, and showed that if $(x, y; z)$ is a reverse triple, then so are $(z, x; y)$ and $(y, z; x)$. Because of this symmetry, we represent a reverse triple as the unordered set $\{x, y, z\}$ in this paper.

at least one reconfiguration sequence of length d exists. If $\mathcal{F}_d^S = \{\ \}$, then all feasible solutions reachable from S have been explored, and no sequence to T exists. Otherwise, we proceed to compute $\mathcal{F}_{d+1}^S$. With a slight modification, this framework can also be used to compute an explicit optimal reconfiguration sequence and to determine the maximum length of such a sequence from S.

3 Reconfiguration Algorithm

In this section, we propose an algorithm based on the ZDD-based framework to solve the optimal reconfiguration problem and the antipode distance problem of optimal ladders. To apply the recurrence in Eq. (1) from Section 2.2, we begin with an initial family $\mathcal{F}_0^L = [L]$, where $[L]$ is the equivalence class of a given ladder L. At each distance d, we define $\mathcal{F}_{d+1}^L$ as the equivalence class of all ladders that can be obtained from ladders in $\mathcal{F}_d^L$ by applying a local swap. This means that we define $N(\mathcal{F})$ as the family of all ladders that can be obtained by applying a local swap to any ladder in $\mathcal{F}$.

Our task is to construct the ZDD of $N(\mathcal{F})$ from the ZDD of a given family $\mathcal{F}$ of ladders. To realize a local swap, we combine a local swap with adjacent layers and the computation of equivalence classes. As described in Section 2.1, a ladder L' is obtained from L by a local swap if there exist $\hat{L} \in [L]$ and $\hat{L}' \in [L']$ such that $\hat{L}'$ is obtained from $\hat{L}$ by applying a local swap with adjacent layers. Let $\mathrm{LS}_{\mathrm{adj}}(\mathcal{F})$ denote the family of all ladders that can be obtained by applying a local swap with adjacent layers to any ladder in $\mathcal{F}$. Then, the neighborhood of $[\mathcal{F}]$ is given by $N([\mathcal{F}]) = [\mathrm{LS}_{\mathrm{adj}}([\mathcal{F}])]$, and thus we can recursively compute

$$\mathcal{F}_{d+1}^L = [\mathrm{LS}_{\mathrm{adj}}(\mathcal{F}_d^L)] \setminus (\mathcal{F}_d^L \cup \mathcal{F}_{d-1}^L),$$

where $\mathcal{F}_0^L = [L]$ holds.

Now, we define the ladder obtained by removing and adding bars to a ladder. Let B be the universal set of bars, $L\ (\subseteq B)$ be a ladder, and let $A, R \subseteq B$. We denote by $\mathrm{RemAdd}(L, B, A, R)$ the ladder obtained from L by removing all bars in R and adding all bars in A:

$$\mathrm{RemAdd}(L,B,A,R) = \begin{cases} (L \cup A) \setminus R & \begin{pmatrix} R \subseteq L, \\ L \cap A = \emptyset \end{pmatrix} \\ \bot & \text{(otherwise)}. \end{cases}$$

The conditions $R \subseteq L$ and $L \cap A = \emptyset$ ensure that all bars to be removed exist in L, and that no bar to be added already exists in L. Otherwise, we denote the result as $\bot$ to indicate that no valid ladder is produced. We extend this notation to a family $\mathcal{F}\ (\subseteq 2^B)$ of ladders as: $\mathrm{RemAdd}(\mathcal{F}, B, A, R) = \{\mathrm{RemAdd}(L, B, A, R) | L \in \mathcal{F}\} \setminus \{\bot\}$. Here, $\bot$ is excluded since it does not correspond to a valid ladder.

As ZDDs support basic family algebra, we can construct $\mathcal{Z}(\mathcal{F}_{d+1}^L)$ (i.e., a ZDD representing $\mathcal{F}_{d+1}^L$) if we can construct both $\mathcal{Z}(\mathrm{LS}_{\mathrm{adj}}(\mathcal{F}))$ and $\mathcal{Z}([\mathcal{F}])$ for any $\mathcal{F}$. Due to space constraints, we describe only the construction of $\mathcal{Z}(\mathrm{LS}_{\mathrm{adj}}(\mathcal{F}))$. Note that $\mathrm{LS}_{\mathrm{adj}}(\mathcal{F})$ can be decomposed into the union of two families: those obtained by applying a left swap with adjacent layers, $\mathrm{RemAdd}(\mathcal{F}, B, B_>(b_{i,k}), B_<(b_{i,k}))$, and those obtained by applying a right swap with adjacent layers, $\mathrm{RemAdd}(\mathcal{F}, B, B_<(b_{i,k}), B_>(b_{i,k}))$. Hence, we have $\mathrm{LS}_{\mathrm{adj}}(\mathcal{F}) = \bigcup_{1 \le i \le \ell-2, 1 \le k \le n-2} (\mathrm{RemAdd}(\mathcal{F}, B, B_>(b_{i,k}), B_<(b_{i,k})) \cup \mathrm{RemAdd}(\mathcal{F}, B, B_<(b_{i,k}), B_>(b_{i,k})))$ Therefore, to construct $\mathcal{Z}(\mathrm{LS}_{\mathrm{adj}}(\mathcal{F}))$, it suffices to construct $\mathcal{Z}(\mathrm{RemAdd}(\mathcal{F}, B, A, R))$ for the universal set B of $\mathcal{F}$ and for arbitrary $A, R \subseteq B$. For a bar $b_{i,k} \in B$, there are four possible cases depending on whether $b_{i,k}$ is included in A and/or R. Due to space limitations, we only describe the case where $b_{i,k} \in A$ and $b_{i,k} \notin R$. Let $\mathcal{F}_0 = \{L \in \mathcal{F} \mid b_{i,k} \notin L\}$ and $\mathcal{F}_1 = \{L \setminus \{b_{i,k}\} \mid L \in \mathcal{F}, b_{i,k} \in L\}$. Note that $\mathcal{F} = \mathcal{F}_0 \cup (\{\{b_{i,k}\}\} \bowtie \mathcal{F}_1)$ holds. Since each ladder in $\{\{b_{i,k}\}\} \bowtie \mathcal{F}_1$ already contains $b_{i,k}$, it is not possible to add $b_{i,k}$ again. Therefore, $\mathrm{RemAdd}(\{\{b_{i,k}\}\} \bowtie \mathcal{F}_1, B, A, R) = \{\ \}$. In contrast, every ladder in $\mathcal{F}_0$ does not contain $b_{i,k}$, so we can safely add it. This yields: $\mathrm{RemAdd}(\mathcal{F}_0, B, A, R) = \{\{b_{i,k}\}\} \bowtie \mathrm{RemAdd}(\mathcal{F}_0, B \setminus \{b_{i,k}\}, A \setminus \{b_{i,k}\}, R)$. Combining the two cases, we obtain the recurrence: $\mathrm{RemAdd}(\mathcal{F}, B, A, R) = (\{\{b_{i,k}\}\} \bowtie \mathrm{RemAdd}(\mathcal{F}_0, B \setminus \{b_{i,k}\}, A \setminus \{b_{i,k}\}, R)) \cup \{\ \}$. This implies that the ZDD $\mathcal{Z}(\mathrm{RemAdd}(\mathcal{F}, B, A, R))$ has the following structure. Let $b_{i,k}$ be the smallest in B with respect to the variable order. Then, the root node of the ZDD is labeled $b_{i,k}$. Its 1-arc points to the node representing $\mathrm{RemAdd}(\mathcal{F}_0, B \setminus \{b_{i,k}\}, A \setminus \{b_{i,k}\}, R)$, and its 0-arc points to 0, which represents the empty family. Thus, we can recursively construct $\mathcal{Z}(\mathrm{RemAdd}(\mathcal{F}, B, A, R))$.

4 Diameter

We prove the following theorem on the diameter of reconfiguration graphs.

Theorem 2 *Let L_ℓ^π and L_r^π be the left-stacked and right-stacked brick optimal ladders of a permutation π, respectively. Let $\mathrm{Diff}_\pi(L_\ell^\pi, L_r^\pi)$ be the family of all reverse triples between them. Then, $|\mathrm{Diff}_\pi(L_\ell^\pi, L_r^\pi)|$ equals the diameter of the reconfiguration graph of π.*

To prove the theorem, we show that for any two optimal ladders L and L' of π, the following upper bound holds: $|\mathrm{Diff}_\pi(L, L')| \le |\mathrm{Diff}_\pi(L_\ell^\pi, L_r^\pi)|$. This result follows from Lemmas 3 and 4. We denote by $A \triangle B$ the symmetric difference between sets A and B.

Lemma 3 *Let L^π_ℓ be the left-stacked brick optimal ladder of a permutation π. Then, for any two optimal ladders L and L' of π, the family of all reverse triples between them satisfies the following:*

$$\mathrm{Diff}_\pi(L, L') = \mathrm{Diff}_\pi(L^\pi_\ell, L) \triangle \mathrm{Diff}_\pi(L^\pi_\ell, L')$$

Proof. For $x, y, z \in \{1, 2, ..., n\}$ with $x < y < z$, we prove the equation by a case analysis on whether the triple $T = \{x, y, z\}$ is in each of $\mathrm{Diff}_\pi(L, L')$, $\mathrm{Diff}_\pi(L^\pi_\ell, L)$, and $\mathrm{Diff}_\pi(L^\pi_\ell, L')$. We denote by $\{x, y\}$ the bar that exchanges x and y. We can observe that if an optimal ladder of a permutation π contains the bar $\{x, y\}$, it is because x and y must be exchanged to obtain π from the identity permutation. Therefore, the existence of $\{x, y\}$ is consistent across all optimal ladders of π. If L^π_ℓ, L, and L' do not contain the bar $\{x, y\}$, then by the definition of a reverse triple, T belongs to none of $\mathrm{Diff}_\pi(L, L')$, $\mathrm{Diff}_\pi(L^\pi_\ell, L)$, or $\mathrm{Diff}_\pi(L^\pi_\ell, L')$. Hence, we focus on the remaining cases where all of L^π_ℓ, L, and L' contain the bar $\{x, y\}$.

(i) $T \notin \mathrm{Diff}_\pi(L^\pi_\ell, L)$ and $T \notin \mathrm{Diff}_\pi(L^\pi_\ell, L')$: By the definition of a reverse triple, $T \notin \mathrm{Diff}_\pi(L^\pi_\ell, L)$ implies that if the bar $\{x, y\}$ is in $R_\ell(L^\pi_\ell, z)$ (resp., $R_r(L^\pi_\ell, z)$), then it is also in $R_\ell(L, z)$ (resp., $R_r(L, z)$). The same holds for $T \notin \mathrm{Diff}_\pi(L^\pi_\ell, L')$. Therefore, in this case, if the bar $\{x, y\}$ is in $R_\ell(L^\pi_\ell, z)$ (resp., $R_r(L^\pi_\ell, z)$), it must also be in both $R_\ell(L, z)$ and $R_\ell(L', z)$ (resp., $R_r(L, z)$ and $R_r(L', z)$). Thus, T is not a reverse triple between L and L', and we conclude that $T \notin \mathrm{Diff}_\pi(L, L')$.

(ii) $T \in \mathrm{Diff}_\pi(L^\pi_\ell, L)$ and $T \in \mathrm{Diff}_\pi(L^\pi_\ell, L')$: By the definition of a reverse triple, $T \in \mathrm{Diff}_\pi(L^\pi_\ell, L)$ implies that if the bar $\{x, y\}$ is in $R_\ell(L^\pi_\ell, z)$ (resp., $R_r(L^\pi_\ell, z)$), then it is in $R_r(L, z)$ (resp., $R_\ell(L, z)$); in other words, it is in the opposite side in L. The same holds for L'. Therefore, in this case, if the bar $\{x, y\}$ is in $R_\ell(L^\pi_\ell, z)$ (resp., $R_r(L^\pi_\ell, z)$), it is in $R_r(L, z)$ and $R_r(L', z)$ (resp., $R_\ell(L, z)$ and $R_\ell(L', z)$). This implies $T \notin \mathrm{Diff}_\pi(L, L')$.

(iii) $T \notin \mathrm{Diff}_\pi(L^\pi_\ell, L)$ and $T \in \mathrm{Diff}_\pi(L^\pi_\ell, L')$: In this case, if the bar $\{x, y\}$ is in $R_\ell(L^\pi_\ell, z)$ (resp., $R_r(L^\pi_\ell, z)$), then it is in $R_\ell(L, z)$ and $R_r(L', z)$ (resp., $R_r(L, z)$ and $R_\ell(L', z)$). Thus, T is a reverse triple between L, L', that is, $T \in \mathrm{Diff}_\pi(L, L')$.

(iv) $T \in \mathrm{Diff}_\pi(L^\pi_\ell, L)$ and $T \notin \mathrm{Diff}_\pi(L^\pi_\ell, L')$: By the same argument as in case (iii), we have $T \in \mathrm{Diff}_\pi(L, L')$. Thus, Lemma 3 holds in all cases. $\square$

Lemma 4 *Let L^π_ℓ and L^π_r be the left-stacked and right-stacked brick optimal ladders of a permutation π, respectively. Then, for any optimal ladder L, $\mathrm{Diff}_\pi(L^\pi_\ell, L) \subseteq \mathrm{Diff}_\pi(L^\pi_\ell, L^\pi_r)$ holds.*

Proof. We can obtain L^π_ℓ from L by applying a sequence of left swaps [14]. Each left swap corresponds one-to-one to a reverse triple that is either decreased or increased in the process [13, 14]. This means that each left swap reduces exactly one reverse triple between L^π_ℓ and L. (Otherwise, an increased reverse triple remains, and by Lemma 1, L^π_ℓ would not be reachable.) Hence, there exists an optimal reconfiguration sequence from L to L^π_ℓ consisting solely of left swaps, and for any intermediate ladder L' in this sequence, we have $\mathrm{Diff}_\pi(L^\pi_\ell, L') \subseteq \mathrm{Diff}_\pi(L^\pi_\ell, L)$. From this observation, if we assume there is a ladder L'' and a sequence from L'' to L^π_ℓ consisting solely of left swaps where L is in this sequence, $\mathrm{Diff}_\pi(L^\pi_\ell, L) \subseteq \mathrm{Diff}_\pi(L^\pi_\ell, L'')$ holds.

Recall that we can obtain L^π_ℓ from L by applying a sequence of left swaps. Similarly, we can obtain L^π_r from L by applying a sequence of right swaps. The concatenation of the sequence from L^π_r to L and from L to L^π_ℓ yields a left-swap-only sequence from L^π_r to L^π_ℓ. Hence, $\mathrm{Diff}_\pi(L^\pi_\ell, L) \subseteq \mathrm{Diff}_\pi(L^\pi_\ell, L^\pi_r)$ holds. $\square$

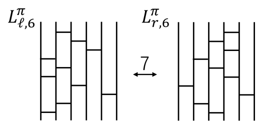


Figure 3: Instances for experiment ($n = 6$).

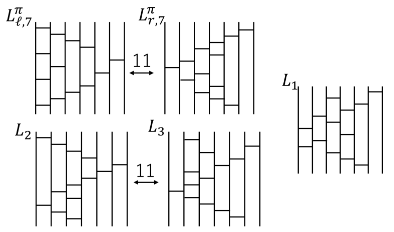


Figure 4: Instances for experiment ($n = 7$).

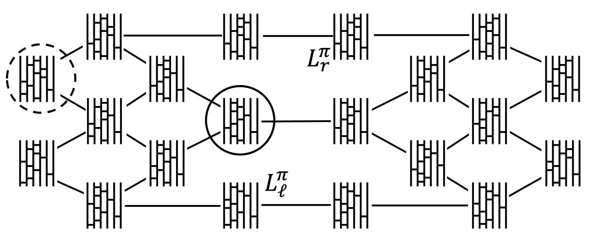


Figure 5: The reconfiguration graph of optimal ladders of $(5, 4, 3, 1, 6, 2)$.

Proof of Theorem 2. From Lemma 3, for any two optimal ladders L and L' of π, $\mathrm{Diff}_\pi(L, L') = \mathrm{Diff}_\pi(L^\pi_\ell, L) \triangle \mathrm{Diff}_\pi(L^\pi_\ell, L')$ holds. By Lemma 4, both $\mathrm{Diff}_\pi(L^\pi_\ell, L)$ and $\mathrm{Diff}_\pi(L^\pi_\ell, L')$ are subsets of $\mathrm{Diff}_\pi(L^\pi_\ell, L^\pi_r)$. Thus, we complete the proof since $\mathrm{Diff}_\pi(L, L') \subseteq \mathrm{Diff}_\pi(L^\pi_\ell, L^\pi_r)$ holds. $\square$

5 Experiments

We implement our algorithm in C++ and conduct experiments on the antipode distance problem of optimal ladders. Experiments are run on a 12th Gen Intel(R)

Table 1: Experimental results for antipode distance problems.

Instance L	n	ℓ	π	d_M	$\lvert\mathcal{F}_0^L\rvert/\lvert\mathcal{F}_1^L\rvert/\cdots/\lvert\mathcal{F}_{d_M}^L\rvert$	$\lvert\bigcup_d \mathcal{F}_d^L\rvert$	$\lvert\mathcal{Z}(\bigcup_d \mathcal{F}_d^L)\rvert$	Time (s)
$L^\pi_{\ell,6}$	6	10	$(5,4,3,1,6,2)$	7	66/51/138/193/156/99/32/33	768	168	0.12
$L^\pi_{r,6}$					33/32/99/156/193/138/51/66			0.11
$L^\pi_{\ell,7}$	7	13	$(7,4,3,2,5,1,6)$	11	286/1,456/1,885/3,848/4,459/4,176/ 2,553/1,205/580/103/29/12	20,592	468	6.6
$L^\pi_{r,7}$					12/29/103/580/1,205/2,553/4,176/ 4,459/3,848/1,885/1,456/286			6.2
L_2					10/167/732/2,324/4,621/5,386/ 1,806/2,321/2,413/586/210/16			6.7
L_1				10	288/516/1,267/2,736/3,598/4,025/ 2,240/2,180/1,583/1,343/816			6.8

Core(TM) i5-12500 (3 GHz) with 8 GB RAM under Ubuntu 22.04.2 LTS. We use six optimal ladders: the left-stacked and right-stacked brick optimal ladders $L^\pi_{\ell,6}$ and $L^\pi_{r,6}$ of $\pi = (5,4,3,1,6,2)$ (see Fig. 3), and four optimal ladders of $\pi = (7,4,3,2,5,1,6)$ shown in Fig. 4 — $L^\pi_{\ell,7}$, $L^\pi_{r,7}$, L_1, and L_2, where $L^\pi_{\ell,7}$ and $L^\pi_{r,7}$ are the left-stacked and right-stacked brick optimal ladders.

In Table 1, n, ℓ, and π indicate the number of lines, the number of bars, and the permutation represented by each instance, respectively. The remaining rows report the experimental results. d_M denotes the distance from each instance to its antipode, and $|\mathcal{F}_0^L|/|\mathcal{F}_1^L|/\cdots/|\mathcal{F}_{d_M}^L|$ are the number of optimal ladders at a distance $d = 0, 1, \ldots, d_M$ from each instance. $|\bigcup_d \mathcal{F}_d^L|$ shows the total number of optimal ladders found, and $|\mathcal{Z}(\bigcup_d \mathcal{F}_d^L)|$ shows the number of nodes in the ZDD of $\bigcup_d \mathcal{F}_d^L$.

Fig. 5 shows the reconfiguration graph of optimal ladders of $(5,4,3,1,6,2)$. Each vertex corresponds to an equivalent class of ladders and is drawn using its representative. $L^\pi_{\ell,6}$ and $L^\pi_{r,6}$ in Fig. 3 is in this graph. This graph has a beautiful symmetry. Its diameter is 7, and the distance between the left-stacked and right-stacked brick optimal ladders equals the diameter, confirming Theorem 2.

In Table 1, comparing the sequences of $|\mathcal{F}_0^L|/|\mathcal{F}_1^L|/\cdots/|\mathcal{F}_{d_M}^L|$ for the left-stacked and right-stacked brick optimal ladders $L^\pi_{\ell,6}$ and $L^\pi_{r,6}$, we observe that the two sequences are reverses of each other. A similar reversal is observed between the sequences for the left-stacked and right-stacked brick optimal ladders $L^\pi_{\ell,7}$ and $L^\pi_{r,7}$. This suggests that the reconfiguration graph of optimal ladders of $(7,4,3,2,5,1,6)$ (the second permutation in the experiment) has a symmetrical structure just like the graph in Fig. 5.

The instance L_2 in Table 1 shows that a pair of two optimal ladders other than the left-stacked and right-stacked brick optimal ones can also give the diameter of reconfiguration graphs. The antipode of L_2 is L_3 in Fig. 4. The fact that the pair of L_2, L_3 gives the diameter can be explained using the equation of Lemma 3. $\mathrm{Diff}_\pi(L^\pi_{\ell,7}, L_2)$ and $\mathrm{Diff}_\pi(L^\pi_{\ell,7}, L_3)$ are the partition of $\mathrm{Diff}_\pi(L^\pi_{\ell,7}, L^\pi_{r,7})$. Hence, we obtain $\mathrm{Diff}_\pi(L_2, L_3) = \mathrm{Diff}_\pi(L^\pi_{\ell,7}, L_2) \triangle \mathrm{Diff}_\pi(L^\pi_{\ell,7}, L_3) = \mathrm{Diff}_\pi(L^\pi_{\ell,7}, L^\pi_{r,7})$. On the other hand, d_M can be smaller than the diameter of the reconfiguration graph (e.g., L_1 in Table 1). To handle such cases, we must solve the following optimization problem: Given a permutation π and a ladder L, find a largest subset $S \subset \mathrm{Diff}_\pi(L^\pi_\ell, L^\pi_r)$ such that there exists a ladder L' satisfying $S = \mathrm{Diff}_\pi(L, L')$. Furthermore, a ladder whose distance (i.e., the size $|S|$) is maximal is not necessarily the antipode of L. For example, consider a ladder in the solid circle in Fig. 5; a ladder in the dotted circle is at a maximal distance of 3, while its antipode distance is 4. This implies a method that greedily maximizes the distance from a given ladder may fail to find the antipode.

Lastly, we compare the size $|\bigcup_d \mathcal{F}_d^L|$ with $|\mathcal{Z}(\bigcup_d \mathcal{F}_d^L)|$. For $n = 6$, we have 768 ladders and the number of nodes for their ZDDs is 168, which is about 22% of the number of ladders. For $n = 7$, the ZDDs are even more compact, using only about 2.3% of the size. These results indicate that ZDDs can represent families of optimal ladders very efficiently.

6 Conclusion

This paper showed the diameter of the reconfiguration graphs of optimal ladders and identified pairs of optimal ladders that attain the diameter. We also proposed an algorithm for solving optimal reconfiguration problems and antipode distance problems of optimal ladders. Furthermore, we conducted experiments using our algorithm and confirmed that the pair of left-stacked and right-stacked brick optimal ladders gives the diameter of the reconfiguration graph for the permutations $\pi = (5,4,3,1,6,2)$ and $\pi = (7,4,3,2,5,1,6)$. The bottleneck of our algorithm lies in identifying the equivalence classes. Although the algorithm finds them by repeatedly applying layer swaps, this process could be accelerated by directly identifying the representative ladders of the classes. Improving this aspect is an important direction for future work.

7 Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers 20H05964, 23K24806, and 25H01114.

References

[1] P. Bonsma and L. Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009.

[2] L. Cereceda, J. Van Den Heuvel, and M. Johnson. Finding paths between 3-colorings. *Journal of graph theory*, 67(1):69–82, 2011.

[3] E. D. Demaine, M. L. Demaine, S. Eisenstat, A. Lubiw, and A. Winslow. Algorithms for Solving Rubik's Cubes. In *Algorithms–ESA 2011*, pages 689–700. Springer, 2011.

[4] O. Firman, P. Kindermann, B. Klemz, A. Ravsky, A. Wolff, and J. Zink. The complexity of finding tangles. In *SOFSEM 2023: Theory and Practice of Computer Science*, volume 13878 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2023.

[5] P. Gopalan, P. G. Kolaitis, E. Maneva, and C. H. Papadimitriou. The connectivity of Boolean satisfiability: computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009.

[6] T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011.

[7] T. Ito, J. Kawahara, Y. Nakahata, T. Soh, A. Suzuki, J. Teruyama, and T. Toda. ZDD-Based Algorithmic Framework for Solving Shortest Reconfiguration Problems. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 167–183. Springer, 2023.

[8] D. E. Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes in Computer Science*. Springer, 1992.

[9] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proceedings of the 30th International Design Automation Conference*, pages 272–277, 1993.

[10] T. Rokicki, H. Kociemba, M. Davidson, and J. Dethridge. God's Number is 20. https://www.cube20.org/, 2010.

[11] T. Soh, Y. Okamoto, and T. Ito. Core Challenge 2022: Solver and Graph Descriptions. https://arxiv.org/abs/2208.02495, 2022. (https://core-challenge.github.io/2022/).

[12] T. Soh, T. Tanjo, Y. Okamoto, and T. Ito. Core Challenge 2023: Solver and Graph Descriptions. https://arxiv.org/abs/2310.17136, 2023. (https://core-challenge.github.io/2023/).

[13] K. Yamanaka, T. Horiyama, and K. Wasa. Optimal reconfiguration of optimal ladder lotteries. *Theoretical Computer Science*, 859:57–69, 2021.

[14] K. Yamanaka, S. Nakano, Y. Matsui, R. Uehara, and K. Nakada. Efficient enumeration of all ladder lotteries and its application. *Theoretical Computer Science*, 411(16–18):1714–1722, 2010.

Fast Approximate Lipschitz Extensions in Doubling Metrics

Donald R. Sheehy[*]

Abstract

We give an efficient algorithm for computing Lipschitz extensions in doubling metrics. After $O(n \log n)$ preprocessing time for sets A and B of size n, any t-Lipschitz function on A can be extended to B in $O(n)$ time. The extension is approximate in the sense that it may be at most $(1+\varepsilon)t$-Lipschitz.

1 Introduction

A real-valued function on a metric space is t-Lipschitz if for all points x and y, we have $f(x) \le f(y) + t\,\mathrm{d}(x,y)$. Let A and B be sets of points in a metric space. The *Lipschitz Extension Problem* is to extend a t-Lipschitz function $f : A \to \mathbb{R}$ to a function $f^+ : A \cup B \to \mathbb{R}$ that is also t-Lipschitz and agrees with f at points of A. In this paper, we give efficient algorithms for computing approximate Lipschitz extensions in metric spaces that exhibit intrinsic low-dimensionality in the form of bounded doubling dimension. This is a generalization of low-dimensional Euclidean spaces that allows for packing arguments similar to geometric volume-packing arguments.

More specifically, we will construct the maximum and minimum Lipschitz extensions. The main application of this work is in topological data analysis where it was shown that topological invariants called *subbarcodes* of an unknown function can be computed from upper and lower bounds on that function [2]. Thus, for a Lipschitz function known only at a subset of points, the algorithms efficiently compute upper and lower bounds on the function values at other points.

The paper provides two specific contributions to the algorithmic theory of Lipschitz extensions. The first is a direct reduction from the problem of computing maximum Lipschitz extensions to the all-nearest neighbor problem. This involves constructing a new metric from the input points and the function f so that the maximum Lipschitz extension can be computed from the nearest neighbors in the new metric. The second contribution is an approximation algorithm that can compute Lipschitz extensions in linear time for any function after $O(n \log n)$ preprocessing time. That is, we pay $O(n \log n)$ time once and then can compute the maximum (or minimum) Lipschitz extension of any function in linear time. This requires a new notion of approximation where the output is guaranteed to lie between the maximum t-Lipschitz extension and the maximum $(1+\varepsilon)t$-Lipschitz extension.

The approach is based on dual-tree algorithms on greedy trees. The background on doubling metric spaces, Lipschitz extensions, and greedy trees is given in Section 2. We then detail the relationship between maximum Lipschitz extension and nearest neighbor search in Section 3. Finally, in Section 4, we explain how dual-tree algorithms can be used to compute the Lipschitz extensions exactly and approximately with good bounds on the running time.

Part of the novelty in this approach comes from the fact that the usual methods to reduce the search space in dual-tree algorithms don't apply directly for Lipschitz extensions. In standard dual-tree applications like all-nearest neighbor search, one can prune neighbors that are far away. However, the maximum Lipschitz extension of a point can be determined be a far away point even if there are other points nearby.

2 Background

2.1 Metric Spaces and Doubling Dimension

A metric space is a pair (X, d) where X is a set and $\mathrm{d} : X \times X \to \mathbb{R}$ is the *metric* or *distance function*. We denote the distance from $x \in X$ to any $S \subseteq X$ as $\mathrm{d}(x,S) = \min_{s \in S} \mathrm{d}(x,s)$.

The *diameter* of a set $A \subseteq X$ is the maximum pairwise distance of points in the set and is denoted diameter(A). A collection of subsets of A is called a *cover* if the union is all of A. The *diameter of a cover* is the maximum diameter among the sets in the cover. A δ*-cover* is a cover of diameter at most δ. For a set $A \subseteq X$ and $\delta \in \mathbb{R}$, the δ*-covering number* is the minimum number of sets in any δ-cover.

The *doubling constant* for a metric space is the maximum over all subsets $A \subseteq X$ of the (diameter$(A)/2$)-covering number. The *doubling dimension* is the base-2 logarithm of the doubling constant.

Note that in many works, the doubling dimension is defined in terms of the radius of a ball rather than the diameter of an arbitrary set. There are some conveniences to that approach, but it leads to certain difficulties when reasoning about the doubling dimension of a subset. Specifically, if the doubling dimension is defined

[*]Computer Science Department, North Carolina State University don.r.sheehy@gmail.com

in terms of radii instead of diameters, then the dimension is not monotone with respect to subsets—removing points can increase the dimension. The definition given here is the original one [6] and should be preferred.

The following lemma is the standard packing argument that motivates the use of doubling dimension in many settings. It is the main tool for bounding the size of sets in doubling metrics.

Lemma 1 (Standard Packing Lemma) *Let P be a finite metric space of doubling dimension d. If D is the diameter of P and for all distinct $a, b \in P$, we have $\mathrm{d}(a, b) > \delta$, then*

$$|P| \leq \left(\frac{2D}{\delta}\right)^d.$$

Proof. Here is the standard proof included for completeness. If $\rho = 2^d$ is the doubling constant, then P can be covered by ρ sets of diameter $D/2$. Covering these sets gives a $D/4$ cover of size ρ^2. Repeating this process until the diameters are less than δ results in cover where each point of P is in its own set. This requires $k = \lceil \lg D/\delta \rceil$ iterations and gives a total size

$$|P| \leq \rho^k = 2^{dk} \leq 2^{d\lceil \lg D/\delta \rceil} \leq \left(\frac{2D}{\delta}\right)^d.$$

□

2.2 Lipschitz Functions and their Extensions

For any set X, there is a natural partial order on real valued functions $f, g : X \to \mathbb{R}$ defined as

$$f \leq g \text{ if and only if for all } x \in X,\ f(x) \leq g(x).$$

Definition 2 *A function $f : X \to \mathbb{R}$ is* t-Lipschitz *iff for all $a, b \in X$,*

$$|f(a) - f(b)| \leq t\mathrm{d}_X(a, b).$$

Equivalently, for all $a, b \in X$,

$$f(b) - t\mathrm{d}_X(a, b) \leq f(a) \leq f(b) + t\mathrm{d}_X(a, b).$$

A function is Lipschitz *if it is 1-Lipschitz.*

Let $A \subseteq X$ be any subset of a metric space and let $f : A \to \mathbb{R}$ be any function. An extension of f to X is a function $f^+ : X \to \mathbb{R}$ such that for all $a \in A$, we have $f^+(a) = f(a)$. If f is t-Lipschitz, then there always exists a t-Lipschitz extension. Perhaps the simplest proof of this statement is to construct such an extension. In fact, we will construct two such extensions with certain universal properties as described in the following lemma.

Lemma 3 *For any metric space X and any t-Lipschitz function $f : A \to X$ with compact*[1] *$A \subseteq X$, there exist t-Lipschitz extensions $\hat{f}, \check{f} : X \to \mathbb{R}$ such that for all t-Lipschitz extensions g, we have $\hat{f} \leq g \leq \check{f}$.*

Proof. Define the *minimum Lipschitz extension* as

$$\hat{f}(x) = \max_{a \in A} f(a) - t\mathrm{d}(x, a).$$

Define the *maximum Lipschitz extension* as

$$\check{f}(x) = \min_{a \in A} f(a) + t\mathrm{d}(x, a).$$

Let $g : X \to \mathbb{R}$ be any t-Lipschitz extension of f. By the definition of Lipschitz,

$$f(a) - t\mathrm{d}(x, a) \leq g(x) \leq f(a) + t\mathrm{d}(x, a)$$

for all $x \in X$ and all $a \in A$. So,

$$g(x) \geq \max_{a \in A} f(a) - t\mathrm{d}(x, a) = \hat{f}(x),$$

and

$$g(x) \leq \min_{a \in A} f(a) + t\mathrm{d}(x, a) = \check{f}(x).$$

□

2.3 Greedy Permutations and Greedy Trees

Given any ordered set of points $P = \{p_0, \ldots, p_{n-1}\}$ in a metric space, we define the ith *prefix* to be

$$P_i := \{p_0, \ldots, p_{n-1}\}.$$

We say that the permutation P is *greedy* if for all $i > 0$, we have

$$\mathrm{d}(p_i, P_i) = \max_{q \in P} \mathrm{d}(q, P_i).$$

In other words, every point is the farthest point to its prefix. Also known as *Gonzalez ordering* or *farthest point traversals*, the greedy permutation is a standard way to produce samples that satisfy both packing and covering properties.

The permutation P is *β-approximate greedy* if for all $i > 0$, we have

$$\beta\mathrm{d}(p_i, P_i) \geq \max_{j \geq i} \mathrm{d}(p_j, P_i).$$

As the definition of (approximate) greedy permutations depends on the distance of each point to its nearest predecessor, it is natural to associate each point with an approximate nearest predecessor called its *parent*. The *insertion distance* of p_i is defined as

$$\varepsilon_i := \mathrm{d}(p_i, \mathrm{par}(p_i)).$$

[1]Here we are assuming A is compact. This is justified because for all of our applications, A will be finite. If one wants to consider these definitions for extensions of noncompact sets, then one simply replaces the max and min with sup and inf.

By convention, we set $\varepsilon_0 = \infty$. For β-approximate greedy permutations we require that parents are β-approximate nearest predecessors, i.e., for all $i > 0$,

$$\varepsilon_i \leq \beta \mathrm{d}(p_i, P_i).$$

Moreover, we require that the insertion distances are in non-increasing order, i.e., that $i < j$ implies $\varepsilon_i \geq \varepsilon_j$. The parent relation turns P into a tree called the *greedy tree.*

Approximate greedy permutations satisfy the following packing bounds.

Lemma 4 *Let P be a β-approximate greedy permutation with insertion distances ε_i. For all i, j, k such that $0 \leq i < j \leq k$, we have*

$$\mathrm{d}(p_i, p_j) \geq \varepsilon_k / \beta.$$

Proof. We use the ordering on insertion distances, the approximate nearest neighbor property of parents, and the fact that $p_i \in P_j$ to derive the bound as follows.

$$\varepsilon_k \leq \varepsilon_j \leq \beta \mathrm{d}(p_j, P_j) \leq \beta \mathrm{d}(p_j, p_i).$$

$\square$

We say that the greedy tree is *α-scaling* iff for all a, b, c such that $a = \mathrm{par}(b)$ and $b = \mathrm{par}(c)$, we have $\mathrm{d}(b, c) \leq \alpha \mathrm{d}(a, b)$. That is, the distance between points is decreasing by a factor of α in each level of the tree.

Greedy permutations and greedy trees can be constructed in $O(n \log n)$ time in doubling metrics [1, 5].

3 Lipschitz Extension as Nearest Neighbor Search

In this secction, we show how the Lipschitz extension problem can be described exactly as an all-nearest-neighbors problem in a new metric space defined by the input function f.

Let A and B be finite subsets of a metric space (X, d). Let $f : A \to \mathbb{R}$ be t-Lipschitz. For any point $b \in B$, computing the maximum t-Lipschitz extension means computing

$$\check{f}(b) = \min_{a \in A} f(a) + t\mathrm{d}(a, b).$$

There is a direct reduction from this problem to nearest neighbor search. Recall that a nearest neighbor of b in A is defined as a point a minimizing $\mathrm{d}(a, b)$. There is an extensive literature on this problem. In this case, we want to minimize a kind of weighted distance where the value $f(a)$ influences the distance.

Define the new metric

$$\mathrm{d}'(x, y) := t\mathrm{d}(x, y) + |f(x) - f(y)|.$$

If $b \in B$ then $f(b)$ is unknown. In that case, let $w = \min_{a \in A} f(a)$ be the smallest value of f and let

$$\mathrm{d}'(a, b) := t\mathrm{d}(a, b) + |f(a) - w|.$$

Note that d' is a proper metric as it is formed from the sum of two other metrics. It can be viewed as the L_1-product metric of d (scaled by t) and the standard metric on $\mathbb{R}$. From this perspective, the points are viewed as pairs $(a, f(a))$ or (b, w).

The following proposition shows how the nearest neighbor with respect to d' determines the maximum Lipschitz extension.

Proposition 5 *For any $b \in B$, let $a \in A$ be the nearest neighbor of b with respect to the distance d'. Then, $\check{f}(b) = f(a) + t\mathrm{d}(a, b)$.*

Proof. Suppose for contradiction that there exists some $a' \in A$ such that

$$\check{f}(b) = f(a') + t\mathrm{d}(a', b) < f(a) + t\mathrm{d}(a, b).$$

It follows that

$$\mathrm{d}'(a', b) < \mathrm{d}'(a, b)$$

and thus, a was not the nearest neighbor of b with respect to d', a contradiction. $\square$

The Trouble With (Approximate) Nearest Neighbors Although the reduction from Lipschitz extension to nearest neighbors gives correct output, it suffers three substantial drawbacks.

1. The nearest neighbor search structure must be reconstructed for each function f.

2. The dimension can increase.

3. Approximate nearest neighbors do not give an approximation to the Lipschitz extension.

This last point is the most critical. Most nearest neighbor data structures that achieve fast running times only give approximate answers. Suppose a is only a c-approximate nearest neighbor of b with respect to d'. Then, for all $a' \in A$,

$$\mathrm{d}'(a, b) \leq c\mathrm{d}'(a', b).$$

There is no direct way to translate this kind of multiplicative approximation to a corresponding multiplicative approximation for the Lipschitz extension. Indeed, if the function value is close to zero, small absolute error may represent a large relative error. This challenge motivates our new notion of approximate Lipschitz extension in the next section.

4 Dual-Tree Algorithms

This section presents algorithms for exact and approximate Lipschtiz extension. The inputs are metric spaces A and B as well as a function $f : A \to \mathbb{R}$. The input metric spaces have been preprocessed into greedy trees. The output is a Lipschtiz extension $g : B \to \mathbb{R}$.

4.1 Traversing Greedy Trees

Given an α-scaling, β-approximate greedy permutation, we process it into a list of 4-tuples of the form `(q, parent, radius, par_radius)`, where

- `q` is the next point,
- `parent` is the parent of `q`,
- `radius` is the length of the longest path from `q` to a leaf, and
- `par_radius` is the length of the longest path from parent to a leaf that only include points appearing *after* `q`.

The length of a path is caclulated as the sum of the lengths of the edges. The only element of this list that is not self-explanatory is the last one. The purpose of `par_radius` is to update the radius of the parent after after inserting `q`. Each insertion can be viewed as removing the subtree rooted at `q` and so the inserted points are all identified with trees that partition P. Thus, removing a subtree from the parent may change its radius.

The 4-tuples are ordered according to the given greedy permutation, so that for each i, we let

$$\varepsilon_i := \mathrm{d}(p_i, \mathrm{par}(p_i)),$$

and have $\varepsilon_0 \geq \varepsilon_1 \geq e_2 \geq \cdots$.

The list of 4-tuples encodes the traversal of the greedy tree. The radii are computed by iterating through the permutation in reverse order. This reversal is like building up the tree from subtrees rather than deconstructing the greedy tree into individual points as will happen in the algorithm. Each time a point is encountered, the radius of its parent is exactly the length of the longest path from the parent to a leaf using only points that come later in the order. The pseudocode is below.[2]

```
def traverse(P):
    radius = {p: 0 for p in P}
    par_radius = {}
    for q in reversed(P):
        p = P.par(q)
        par_radius[q] = radius[p]
        radius[p] = max(radius[p],
                        radius[q] + d(p, q)
                        )
    return [(p, P.par(p), radius[p],
             par_radius[p]
            )
            for p in P]
```

[2]The syntax guide for this style of pseudocode can be found at `python.org`.

By definition, the maximum Lipschitz extension at $b \in P_b$ is defined in terms of a point $a \in P_a$. We say the pair (a, b) is *critical* if

$$\check{f}(b) = f(a) + \mathrm{d}(a, b).$$

There is a trivial quadratic-time algorithm to compute the maximum Lipschitz extension. For each $b \in B$, iterate over the points of A to search for the point a such that (a, b) is critical.

In this section we give two dual-tree algorithms [3,4], one for the exact maximum 1-Lipschitz extension and one for an approximation. The general paradigm of dual-tree algorithms gains efficiency by exploiting the spatial locality inherent in performing many searches on the same tree. The greedy trees represent clusters and the tree structure gives a hierarchy of clusterings. The two trees are traversed simultaneously. In both algorithms, we speed up the naive quadratic-time algorithm by comparing entire clusters of points instead of treating them individually.

4.2 An Exact Algorithm

In this subsection, we give a high-level description of a dual-tree algorithm for the exact maximum Lipschitz extension problem. Later, we give a much more detailed description of the approximate version.

It suffices to consider only 1-Lipschitz functions as one can scale distances and function values for different Lipschitz constants. Also, we don't consider minimum extensions separatly because the min extension of f is the negation of the max extension of $-f$.

The algorithm inserts points one at a time. Each inserted point x has an associated radius $r[x]$ that goes down as the cluster at x shrinks (i.e., as more points are inserted). Let $a \in A$ and $b \in B$ be centers of nodes with radii $r[a]$ and $r[b]$ respectively. Let P_a and P_b denote the corresponding clusters, i.e., the points of the subtrees rooted at the nodes.

The main data structure used by the algorithm is the *viability graph* V, a bipartite graph on nodes maintaining the invariant that if there exists a critical pair $(a' \in P_a, b' \in P_b)$, then there is an edge $a \sim b$ in V. We also store for each point $b \in B$, the current upper bound

$$g[b] := \min_{a \sim b} f(a) + \mathrm{d}(a, b) \geq \check{f}(b).$$

Note that for $r[b]$ and $g[b]$ we use square brackets to remind the reader that these are not fixed functions, but rather dictionaries that are updated as the algorithm progresses.

The points are processed in order of insertion distance by merging the two greedy permutations. For each point b in B that has been inserted, we maintain the invariant that $g[b] \geq \check{f}(b)$. The viability graph starts with a single edge connecting the roots of the two trees. In each iteration, the new point is assigned to have the same neighbors as its parent. The radius of the new point and its parent are updated. For any point b whose neighbors changed (including if it was just inserted), we check to see if the new neighbor reduced $g[b]$. The last step is to prune away edges incident to the new nodes that are no longer needed to satisfy the viability graph invariant. We show that the condition (1) below suffices to guarantee an edge can be safely pruned.

The viablity graph invariant allows that if there are no critical pairs in $P_a \times P_b$, then $a \sim b$ can be pruned. The trick is to prune these edges without searching for a critical pair among the subtrees. Any edge $a \sim b$ satisfying the following condition can be pruned.

$$g[b] + r[b] < (f(a) - r[a]) + (\mathrm{d}(a,b) - r[a] - r[b]). \quad (1)$$

This condition is exactly what is required to guarantee the non-existence of a critical pair in $P_a \times P_b$. By the triangle inequality and the definition of Lipschitz, we can see the following bounds hold for all $a' \in P_a$ and $b' \in P_b$ whenever (1) holds.

$$\begin{aligned} \check{f}(b') &\leq g[b] + r[b] \\ &< (f(a) - r[a]) + (\mathrm{d}(a,b) - r[a] - r[b]) \\ &\leq f(a') + \mathrm{d}(a',b'). \end{aligned}$$

So, (a', b') is not critical.

The correctness of the algorithm follows immediately from the viability graph invariant. At the end of the algorithm, all clusters are single points and there is an edge if and only if the the points are critical. The upper bounds give the exact maximum Lipschitz extension.

The running time is dominated in each iteration by the work of iterating over the new edges and checking if they can be pruned. Although one might expect that many edges can be pruned in practice, there is no clear way to *guarantee* that only a subquadratic number of edges will be considered in the course of the algorithm. Unlike in the all-approximate-nearest-neighbor problem, the critical pairs can be far apart. It seems unlikely that any dual-tree methods will give subquadratic running times for the *exact* Lipschitz extension problem. The next sections show that for approximations, the viability graph remains sparse.

An Approximation Algorithm The goal of the approximation algorithm is to produce a function $g : B \to \mathbb{R}$ such that $\check{f} \leq g \leq \check{f}_{1+\varepsilon}$ for a given $\varepsilon \geq 0$, where

$$\check{f}_{1+\varepsilon}(b) = \min_{a \in A} f(a) + (1+\varepsilon)\mathrm{d}(a,b).$$

The approximation algorithm is the same as the exact algorithm, except that we will modify the pruning condition. If the following condition is satisfied, then the edge $a \sim b$ can be pruned.

$$g[b]+r[b] < (f(a)-r[a])+(1+\varepsilon)(\mathrm{d}(a,b)-r[a]-r[b]). \quad (2)$$

Note that for $\varepsilon = 0$, this condition is the same as the exact case. However, this new pruning condition requires a different correctness proof and allows for a stronger running time guarantee as we will see below. This algorithm can and often will prune away even critical edges, but not before an approximate value of g has been recorded.

For completeness, we give the pseudocode below for edge pruning as a method of the viability graph. For simplicity, we assume the value of ε is fixed.

We assume a basic bipartite graph data structure that is initialized with a pair of vertex sets and supports the operations `add_edge`, `remove_edge`, and `nbrs`. The first two are self-explanatory. The function `nbrs` returns an iterator over the neighbors of a vertex.

```
def try_prune(self, a, b, r, g, f):
    if g[b] + r[b] < (f(a) - r[a])
        + (1+epsilon)(d(a,b) - r[a]- r[b]):
        self.remove_edge(a,b)

def lipschitz_extend(A, B, f):
    V = ViabilityGraph(A,B)
    V.add_edge(A[0], B[0])

    # Initialize r and g
    r = dict()
    g = dict()
    g[B[0]] = f(A[0]) + d(A[0], B[0])

    # order merged lists by insertion distance
    L = merge(traverse(A), traverse(B))

    for (q, parent, radius, par_radius) in L:
        r[parent] = par_radius
        r[q] = radius

        if parent in g:  # (if q in B)
            for x in V.nbrs(parent):
                g[q] = min(g[q], f(x) + d(x,q))
                V.add_edge(x,q)
                V.try_prune(x,q,r,g,f)
                V.try_prune(x,parent,r,g,f)
        else: # (if q in A)
            for x in V.nbrs(parent):
                g[x] = min(g[x], f(q) + d(q,x))
                V.add_edge(q,x)
                V.try_prune(q,x,r,g,f)
                V.try_prune(parent,x,r,g,f)

    return g
```

4.3 A Proof of Correctness

Theorem 6 *The output of* `lipschitz_extend(A,B,f)` *is a function $g : B \to \mathbb{R}$ such that*

$$\check{f} \leq g \leq \check{f}_{1+\varepsilon}.$$

Proof. At any time in the algorithm, let $A' \subseteq A$ and $B' \subseteq B$ be the points that have been inserted so far. We prove that after each iteration of the main loop, we satisfy the invariant that for all $b \in B'$, we have

$$\check{f}(b) \leq g[b] \leq \min_{a \in A'}(f(a) + (1+\varepsilon)\mathrm{d}(a,b)).$$

In other words, the current extension g on B' is always at least as large as the maximum Lipschitz extension and is always at most as large as the maximum $(1+\varepsilon)$-Lipschitz extension of the points that have been inserted so far. This invariant will imply the desired guarantee.

Let $a' \in A'$ and $b' \in B'$ be such that a' minimizes $f(a') + (1+\varepsilon)\mathrm{d}(a', b')$. When either a' or b' was inserted, if the parent was adjacent to the other point, the algorithm updates $g[b']$ guaranteeing that

$$g[b'] \leq f(a') + \mathrm{d}(a', b') \leq f(a') + (1+\varepsilon)\mathrm{d}(a', b')$$

This easily satisfies the invariant, so we may assume that there is no such edge. So, at some earlier time there was an edge $a \sim b$ that was pruned away where a is an ancestor of a' and b is an ancestor of b'. Let r_0 and g_0 denote the values of r and g when the edge was pruned. When each point of B is inserted, the value of g is initialized by the value of the parent plus its distance and can only get smaller after that. So,

$$g[b'] \leq g_0[b] + r_0[b].$$

By the pruning condition, the definition of Lipschitz, and the triangle inequality, we have

$$\begin{aligned} g_0[b] + r_0[b] &\leq (f(a) - r_0[a]) \\ &\quad + (1+\varepsilon)(\mathrm{d}(a,b) - r_0[a] - r_0[b]) \\ &\leq f(a') + (1+\varepsilon)\mathrm{d}(a', b'). \end{aligned}$$

Thus, the invariant holds. □

4.4 Running Time Analysis

Degree Bounds. The key to efficient dual-tree algorithms is a bound on the degree of the viability graph. If there are only a constant number of neighbors of any point, then each new point only requires constant time.

For the following lemmas, let s be the distance from q to its parent in an iteration of the main loop.

Lemma 7 *Then, for all inserted points p we have $r[p] \leq \frac{s}{1-\alpha}$.*

Proof. The value of $r[p]$ is the length of a path in the tree. The first edge in the path has length at most s. Each subsequent edge shrinks by a factor of α. So,

$$r[p] \leq s + \alpha s + \alpha^2 s + \cdots \leq s\left(\sum_{i=0}^{\infty} \alpha^i\right) = \frac{s}{1-\alpha}.$$

□

Lemma 8 *If $a \sim b$ is not pruned, then*

$$\mathrm{d}(a,b) \leq \frac{2 + 4/\varepsilon}{1-\alpha} s.$$

Proof. Because the edge was not pruned, we have

$$g[b] + r[b] > f(a) - r[a] + (1+\varepsilon)(\mathrm{d}(a,b) - r[a] - r[b]).$$

Rearranging this pruning condition, we get

$$\varepsilon\mathrm{d}(a,b) \leq (g[b] - (f(a) + \mathrm{d}(a,b))) + (2+\varepsilon)(r[a] + r[b]).$$

Observe that $g[b] \leq f(a) + \mathrm{d}(a,b)$ because otherwise, we would have updated $g[b]$ at the time the edge was added. So, the inequality above simplifies to

$$\varepsilon\mathrm{d}(a,b) \leq (2+\varepsilon)(r[a] + r[b]).$$

So, by Lemma 7, we have

$$\mathrm{d}(a,b) \leq \frac{2+\varepsilon}{\varepsilon}\frac{2s}{1-\alpha} = \frac{2 + 4/\varepsilon}{1-\alpha} s.$$

□

Theorem 9 *Let A and B be subsets of a doubling metric space that have already been processed into β-approximate greedy permutations with an α-scaling parent mapping. Then* `lipschitz_extend(A, B, f)` *runs in $(2 + 1/\varepsilon)^{O(d)} n$ time.*

Proof. The main loop iterates once for each input point. In each iteration, it loops over the neighbors of a point. It will suffice to show that the number of unpruned edges incident to any point is always at most constant. At any iteration, let s be the last insertion distance. Then, any upruned edges $a \sim b$ will satisfy $\mathrm{d}(a,b) \leq \frac{2+4/\varepsilon}{1-\alpha} s$ by Lemma 8. Therefore, the diamter of the set of neighbors is at most twice this value. Moreover, any pair of neighbors are s/β-separated by Lemma 4. Finally, Lemma 1 implies that there are at most $\left(\frac{4\beta(2+4/\varepsilon)}{1-\alpha}\right)^d$ neighbors. Assuming reasonable choices like $\alpha = 2/3$ and $\beta = 2$, this yields a degree bound of $(48(1 + 2/\varepsilon))^d = (2 + 1/\varepsilon)^{O(d)}$. This means that we can insert a point in $(2 + 1/\varepsilon)^{O(d)}$ time, giving the desired running time for all $2n$ points. □

References

[1] O. Chubet, D. Sheehy, and S. Sheth. Simple construction of greedy trees and greedy permutations, 2024.

[2] O. A. Chubet, K. P. Gardner, and D. R. Sheehy. A theory of sub-barcodes. In *41st International Symposium on Computational Geometry (SoCG 2025)*, 2025.

[3] O. A. Chubet, P. M. Parikh, D. R. Sheehy, and S. S. Sheth. Approximating the directed hausdorff distance. In *Proceedings of the 35th Canadian Conference on Computational Geometry*, 2023.

[4] R. R. Curtin. *Improving Dual-Tree Algorithms.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2016.

[5] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

[6] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2004.

Hausdorff Edit Distance

Jonathan James Perry* Benjamin Raichel*

Abstract

The Hausdorff distance is a standard measure of similarity between two finite point sets. Here we introduce the Hausdorff Edit Distance problem, where given a parameter k, the goal is to make up to k edits, either insertions or deletions, to the point sets so as to minimize their Hausdorff distance. When only deletions are allowed the problem is polynomial time solvable, though the problem is APX-Hard when insertions are allowed, due to its connection with k-center clustering. By reducing to repeated calls to any approximate k-center clustering algorithm, we show how to achieve similar approximation factors and running times for variants of the Hausdorff Edit Distance problem.

1 Introduction

Hausdorff Distance. Given two finite point sets P and Q in a metric space, the Hausdorff distance $d_{\mathcal{H}}(P,Q)$ is a natural and common way of measuring the similarity between P and Q. Define the one sided Hausdorff distance, $d_h(P,Q)$, as the maximum distance of a point in P from its nearest point in Q, that is $d_h(P,Q) = \max_{p\in P} \|p-Q\|$ (where $\|\cdot\|$ denotes the metric over the points). Then the standard (two sided) Hausdorff distance is $d_{\mathcal{H}}(P,Q) = \max\{d_h(P,Q), d_h(Q,P)\}$. Clearly the Hausdorff distance can be computed in quadratic time by looking at all pairwise distances, though using Voronoi diagrams in the plane it can be computed in $O(N \log N)$ time where $N = \max\{|P|,|Q|\}$. Using an approximate nearest neighbor data structure for points in $\mathbb{R}^d$ for any constant d, one can get $(1+\varepsilon)$-approximation in $O(N \log N + N/\varepsilon^d)$ time [17].

The Hausdorff distance is well known to be sensitive to outliers, as even a single outlier point can arbitrarily increase the Hausdorff distance. One way to address this is to use the *Partial Hausdorff Distance* introduced in [19]. Given parameters k and ℓ, the k,ℓ Partial Hausdorff Distance is the maximum of the kth ranked value in $\{\|p-Q\| \mid p \in P\}$ and the ℓth ranked value in $\{\|q-P\| \mid q \in Q\}$. Alternatively, one can consider the RMS Hausdorff distance, where rather than taking the maximum (or kth ranked) nearest neighbor distance, one instead takes the sum of the squares of all nearest neighbor distances, see [2,4] and references therein. These and many other prior works also minimize the Hausdorff distance or other measures under translation (or other transformations), though a discussion of this topic is outside the scope of the current paper.

Edit Distance. In this paper we consider the case that the point sets P and Q may be faulty, and our goal is to make up to k edits to P or Q so as to minimize their Hausdorff distance. Here an edit will either be an insertion or a deletion of a point from either P or Q. When only deletions are allowed, this relates to the Partial Hausdorff Distance discussed above, as well as other outlier problems such as clustering with outliers discussed below. For geometric problems on unordered point sets, the more general case where insertions are allowed appears to be less well studied, and indeed may not make sense for many problems, such as center based clustering. However, for ordered point sets there are several relevant papers (as indeed such cases bear more resemblance to classic string edit distance).

The Fréchet distance is a standard measure of similarity between polygonal curves (i.e. ordered point sets). [13] defined and gave various results for the Fréchet edit distance, where the goal is to make up to k insertions or deletions so as to minimize the Fréchet distance (prior works considered shortcutting, which relates to the deletion only case). In the geometric edit distance problem [1,14,15], given two ordered point sequences, the goal is to find a monotone matching of sequences which minimizes the sum of the distances of the matched points plus a penalty on the number of unmatched points, i.e. arguably further from our notion of edit than [13]. Finally, [12] considers the graph edit distance problem, which requires modifying edges and vertices and gets even further from our problem. In so doing, they define a heuristic approach which they call Hausdorff Edit Distance. This measure, however, differs dramatically from ours, in that it is a measure between graphs, and is a summed distance over all matched vertices and edges, and thus is only mentioned as it shares the same name.

Clustering. Insertions in the Hausdorff Edit Distance problem are closely related to the standard k-center clustering problem. Here one is given a set P of n points

*Department of Computer Science, University of Texas at Dallas, USA {jperry,benjamin.raichel}@utdallas.edu. Work on this paper was partially supported by NSF AF Award CCF-2311179.

from a metric space, and the goal is to select a set C of k *center* points from the metric space, so as to minimize the maximum distance of any point in P to its nearest center in C. This problem is known to be APX-hard. Specifically, unless P=NP, for general metric spaces the problem cannot be approximated with any factor less than 2 [18], and even in the plane it remains hard to approximate with a factor of roughly 1.82 [11]. Conversely, the standard greedy algorithm of Gonzalez [16] yields a 2-approximation in any metric space. When $P \subset \mathbb{R}^d$ for constant d, [3] gave an $O(n \log k) + (k/\varepsilon)^{O(k^{1-1/d})}$ time $(1+\varepsilon)$-approximation.

When we separate the deletion and insertion budgets, Hausdorff Edit Distance then closely relates to k-center clustering with outliers, where given a number $0 \leq \ell \leq n$, you are allowed to choose some ℓ points from P that are not required to be covered by the centers. [7] initiated the formal study of k-center clustering with outliers and gave a simple greedy 3-approximation for any metric on the point set P. This was subsequently improved to a 2-approximation in [6] using LP based methods. As pointed out in [10], these results assume the metric consists only of the point set P, as opposed to allowing P to be a subset from some larger metric space from which the centers C can be chosen (as we do in the current paper, which captures for example the Euclidean case). By the triangle inequality, however, these results would still imply constant factor approximations for the case where P is a subset from some larger metric space. Many other variants, including bi-criteria approximations, streaming, and dynamic variants have been considered [5, 9, 10].

Our Contributions. This paper introduces the Hausdorff Edit Distance problem, where insertions and deletions are allowed. As remarked above, the deletion only case relates to the previously defined Partial Hausdorff Distance problem, though allowing insertions is new, and we believe is a valuable addition given the ubiquity of the Hausdorff distance. Our problem closely relates to k-center clustering, which we directly use to achieve the following results. Below, P is a set of n points, Q is a set of m points, k and ℓ are integer parameters, and $N = \max\{m, n\}$.

- In Section 3, the Hausdorff Edit Distance problem is first shown to be APX-hard to compute. Conversely, our main result shows that given any α-approximation to k-center with run time $T(n, k)$, then for any constant $\varepsilon > 0$, one can compute an $(\alpha + \varepsilon)$-approximation to the Hausdorff Edit Distance in $O\big((mn + k^2N)\log(N) + k \cdot T(N, k)\big)$ time.
- Using the standard greedy 2-approximation for k-center, our main result implies a $(2+\varepsilon)$-approximation for Hausdorff Edit Distance in $O\big((mn + k^2N)\log(N)\big)$ time. In Appendix A.1, we argue that for the Euclidean case where $P, Q \subset \mathbb{R}^d$ for constant d, this can be improved to either a $(2+\varepsilon)$-approximation in $O\big(k^2N\log(N)\big)$ time or a $(1+\varepsilon)$-approximation in $O\big(k^2N\log(N)\big) + (k/\varepsilon)^{O(k^{1-1/d})}$ time.
- In Section 3.1 we consider the case where we have separate insertion and deletion budgets, k and ℓ. This case is again APX-hard, and we show that given any $O(T(n, k, \ell))$ time α-approximation to k-center clustering with ℓ outliers, then for any constant $\varepsilon > 0$, one can compute an $(\alpha + \varepsilon)$-approximation in $O\big((mn + k\ell \cdot T(\max\{m, n\}, k, \ell))\log(mn)\big)$ time.
- In Section 3.2 we consider the case where only deletions are allowed. This case is no longer APX-hard. We show that it can be solved exactly in $O(mn)$ time in general, exactly in $O(N \log N)$ time in the plane, and can be $(1+\varepsilon)$-approximated in $O(N \log N + N/\varepsilon^d)$ time for points in $\mathbb{R}^d$ for constant d. The deletion only variant is very similar to the previously defined Partial Hausdorff Distance, though unlike [19], we provide a formal algorithm and analysis, which both establishes the equivalence to our variant with split budgets, and allows for efficient computation in bounded dimensions.
- Finally, in Appendix A.4 we consider several natural variants of our problem, and remark how our techniques easily extend to these variants.

2 Preliminaries

Let $(X, \|\cdot\|)$ be a metric space with point set X and metric $\|\cdot\|$, where for $p, q \in X$ we write $\|p - q\|$ to denote their distance.[1] Throughout, $(X, \|\cdot\|)$ will be any fixed ambient metric, and when we say we are given a finite point set P, it is inferred that $P \subseteq X$ and we are using the metric $\|\cdot\|$. We assume that $\|p - q\|$ takes constant time to compute (otherwise, our running times must be multiplied by the time it takes to compute $\|p - q\|$).

For finite point sets P, Q, let $\|P - Q\| = \min_{p \in P, q \in Q} \|p - q\|$, where for a single point p we write $\|p - Q\| = \|\{p\} - Q\|$. For finite sets P, Q we then define the one-sided Hausdorff Distance as

$$d_h(P, Q) = \max_{p \in P} \left(\min_{q \in Q} \|p - q\| \right) = \max_{p \in P} \|p - Q\|,$$

where if $P = \emptyset$ then $d_h(P, Q) = 0$, and if $P \neq \emptyset$ and $Q = \emptyset$ then $d_h(P, Q) = \infty$. The standard Hausdorff

[1] We use the standard Euclidean distance notation as our general metric distance notation, both for conceptual ease and to distinguish it from our Hausdorff $d_{\mathcal{H}}$ notation.

Distance is then the bi-directional extension

$$d_{\mathcal{H}}(P,Q) = \max\Big(d_h(P,Q), d_h(Q,P)\Big).$$

For finite point sets P and Q, let their symmetric difference be denoted $P \oplus Q = \{x \mid (x \in P \wedge x \notin Q) \vee (x \in Q \wedge x \notin P)\}$. For a finite point set P, we refer to either an insertion of a point into P or a deletion of a point from P as an *edit*. Then for a finite set P of n points and any integer $0 \leq k \leq n$, let $\mathtt{E}(P,k)$ be the set of all sets produced by up to k edits to P, that is $\mathtt{E}(P,k) = \{Q \mid |P \oplus Q| \leq k\}$.

The Hausdorff Edit Distance is then defined as

$$d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) = \min_{0\leq k' \leq k}\left(\min_{\substack{P' \in \mathtt{E}(P,k'),\\ Q' \in \mathtt{E}(Q,k-k')}}\Big(d_{\mathcal{H}}(P',Q')\Big)\right),$$

that is, the minimum possible Hausdorff Distance after up to k edits are made between P and Q. We write $\mathsf{real}(d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k))$ to denote a pair $P' \in \mathtt{E}(P,k'), Q' \in \mathtt{E}(Q,k-k')$, for some $0 \leq k' \leq k$, realizing $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$.

Let $P' \in \mathtt{E}(P,k'), Q' \in \mathtt{E}(Q,k-k')$ for some $0 \leq k' \leq k$. For $\alpha \geq 1$, we refer to P', Q' as an *α-approximation* to $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$ if $d_{\mathcal{H}}(P',Q') \leq \alpha \cdot d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$. That is, limiting to k edits is a hard constraint and the approximation is on the distance.

Let $\mathtt{I}(P,k) = \{Q \mid P \subseteq Q, |Q| - |P| \leq k\}$ and $\mathtt{D}(P,k) = \{Q \mid Q \subseteq P, |P| - |Q| \leq k\}$. Then one can analogously define the Hausdorff Insertion Distance $d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k)$ or Hausdorff Deletion Distance $d^{\mathtt{D}}_{\mathcal{H}}(P,Q,k)$ by respectively replacing the $\mathtt{E}$ sets in the above definition with $\mathtt{I}$ or $\mathtt{D}$ sets.

We now define the standard k-center clustering objective. For a finite set P and an integer $k \geq 0$ define

$$kcenter(P,k) = \min_{|C|\leq k} \max_{p\in P} \|p - C\|,$$

that is, we wish to cover the set P with a set of centers C so as to minimize the maximum distance of a point in P to its nearest center in C. For $\alpha \geq 1$, we refer to a set C', such that $|C'| \leq k$, as an α-approximation to $kcenter(P,k)$ if $\max_{p\in P} \|p - C'\| \leq \alpha \cdot kcenter(P,k)$.

3 Hausdorff Edit Distance

In this section we first show that computing $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$ is APX-Hard, and then provide approximation algorithms for $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$ and its variants.[2] These results are achieved by making a connection between Hausdorff Edit Distance and k-Center Clustering, which becomes clear after first making the following observation:

Lemma 1 *For any finite point sets P, Q and integer $k \geq 0$, $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) = d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k)$.*

[2]For simplicity our results are presented in terms of computing the value $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$ (or related quantities), however, the proofs implicitly construct sets that achieve this value.

Proof. First, $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) \leq d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k)$ as they are minimization problems respectively over edits or insertions, where since an insertion is a type of edit, for any set P' and integer $k' \geq 0$ we have $\mathtt{I}(P',k') \subseteq \mathtt{E}(P',k')$.

Now we argue $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) \geq d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k)$. So let $\{P',Q'\} = \mathsf{real}(d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k))$ be the pair realizing the Hausdorff Edit Distance. If no deletions occurred, i.e. $P \subseteq P'$ and $Q \subseteq Q'$, then P', Q' is a possibility for $d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k)$ and so we are done. So suppose otherwise, and let p be a point that was deleted from P, that is $p \in P \setminus P'$. Rather than deleting p from P we instead insert p into Q, that is, let $\hat{P} = P' \cup \{p\}$ and $\hat{Q} = Q' \cup \{p\}$. We claim that $d_{\mathcal{H}}(\hat{P},\hat{Q}) \leq d_{\mathcal{H}}(P',Q')$. First, observe that as $p \in \hat{Q}$, $d_h(\hat{P},\hat{Q}) = \max_{x\in\hat{P}} \|x - \hat{Q}\| = \max_{x\in P'} \|x - \hat{Q}\| \leq \max_{x\in P'} \|x - Q'\| = d_h(P',Q')$. Similarly, as $p \in \hat{P}$, $d_h(\hat{Q},\hat{P}) = \max_{x\in\hat{Q}} \|x - \hat{P}\| = \max_{x\in Q'} \|x - \hat{P}\| \leq \max_{x\in Q'} \|x - P'\| = d_h(Q',P')$. Thus $d_{\mathcal{H}}(\hat{P},\hat{Q}) = \max\{d_h(\hat{P},\hat{Q}), d_h(\hat{Q},\hat{P})\} \leq \max\{d_h(P',Q'), d_h(Q',P')\} = d_{\mathcal{H}}(P',Q')$. This in turn implies $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) \geq d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k)$ as we then can replace all deletions from either set by corresponding insertions without increasing the cost. $\square$

Lemma 2 *For any finite point set P and integer $k \geq 0$, $d^{\mathtt{E}}_{\mathcal{H}}(P,\emptyset,k) = kcenter(P,k)$.*

Proof. By Lemma 1, we have $d^{\mathtt{E}}_{\mathcal{H}}(P,\emptyset,k) = d^{\mathtt{I}}_{\mathcal{H}}(P,\emptyset,k)$, thus we will equivalently prove that $d^{\mathtt{I}}_{\mathcal{H}}(P,\emptyset,k) = kcenter(P,k)$. First, we show that $d^{\mathtt{I}}_{\mathcal{H}}(P,\emptyset,k) \geq kcenter(P,k)$.

For readability, we write "$\min\limits_{k',P',Q'}$" as shorthand for "$\min_{0\leq k'\leq k} \min_{P'\in\mathtt{I}(P,k'),Q'\in\mathtt{I}(\emptyset,k-k')}$" below.

$$\begin{aligned} d^{\mathtt{I}}_{\mathcal{H}}(P,\emptyset,k) &= \min_{k',P',Q'}\Big(d_{\mathcal{H}}(P',Q')\Big) \geq \min_{k',P',Q'}\Big(d_h(P',Q')\Big) \\ &= \min_{k',P',Q'}\left(\max_{p\in P'}\|p - Q'\|\right) \geq \min_{Q'\in\mathtt{I}(\emptyset,k)}\left(\max_{p\in P}\|p - Q'\|\right) \\ &= \min_{|Q'|\leq k}\left(\max_{p\in P}\|p - Q'\|\right) = kcenter(P,k) \end{aligned}$$

Now we argue $d^{\mathtt{I}}_{\mathcal{H}}(P,\emptyset,k) \leq kcenter(P,k)$. So let $C = \mathsf{real}(kcenter(P,k))$, and let $r = kcenter(P,k) = \max_{p\in P} \|p - C\|$. Thus $d_h(P,C) = \max_{p\in P} \|p - C\| = r$.

Suppose there exists some $c \in C$ such that $\|c - P\| > r$, then $\max_{p\in P} \|p - C \setminus c\| = \max_{p\in P} \|p - C\| = r$. Hence any such point can be deleted without affecting the quality of the k-center solution, and so we assume every point in C has some point from P within distance r, which implies $d_h(C,P) = \max_{c\in C} \|c - P\| \leq r$. Thus $d^{\mathtt{I}}_{\mathcal{H}}(P,\emptyset,k) \leq d_{\mathcal{H}}(P,C) = \max\{d_h(P,C), d_h(C,P)\} \leq r$. $\square$

Computing $kcenter(P,k)$ is known to be an APX-Hard problem. Specifically, unless P=NP, in general metric spaces it is hard to approximate within less than a factor of 2 [18], and even in the plane it is hard to approximate within a factor of roughly 1.8 [11]. Thus the above lemmas immediately imply the following.

Theorem 3 *For any finite point sets P, Q and integer $k \geq 0$, the problems of computing either $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$ or $d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k)$ are APX-Hard. The problems remain APX-Hard even when $P, Q \subset \mathbb{R}^2$.*

We now provide an approximate decision algorithm for Hausdorff Edit Distance, capable of using any α-approximation algorithm for $kcenter(P,k)$ as a subroutine, where $\alpha > 1$ is some constant. Let this subroutine be denoted $\mathbf{kcen}(P,k)$.

Algorithm 1: HausEdit(P,Q,k,r)

```
1 Mark all $p \in P$ such that $\|p - Q\| \leq r$.
    Mark all $q \in Q$ such that $\|q - P\| \leq r$.
2 Create sets $P' \subseteq P$ and $Q' \subseteq Q$ by removing all
    marked points.
3 $\beta = \infty$
4 for $k' = 0$ to $k$ do
5     $\beta = \min\{\beta, \max\{\mathbf{kcen}(P',k'), \mathbf{kcen}(Q',k-k')\}\}$
6 if $\beta \leq \alpha r$ then    // kcen$(P,k)$ is an $\alpha$-approx
7     return True
8 else
9     return False
```

Lemma 4 *Let P be a set of n points, Q be a set of m points, and let $k \geq 0$ be an integer. Let $\mathbf{kcen}(P,k)$ be an algorithm which returns an α-approximation to $kcenter(P,k)$ in $O(T(n,k))$ time.*[3] *Then if $\mathbf{HausEdit}(P,Q,k,r)$ returns True then $r \geq d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)/\alpha$, and if $\mathbf{HausEdit}(P,Q,k,r)$ returns False then $r < d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$. The running time of $\mathbf{HausEdit}(P,Q,k,r)$ is $O\big(mn + k\cdot T(\max\{m,n\},k)\big)$.*

Proof. Recall by Lemma 1 that $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) = d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k)$ and thus it suffices to consider the insertion only case. Let P' and Q' be as defined in $\mathbf{HausEdit}(P,Q,k,r)$. First, observe that $d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k) \leq r$ if and only if $d^{\mathtt{I}}_{\mathcal{H}}(P',Q',k) \leq r$. This follows as any point in $P \setminus P'$ is already within distance r to a point in Q (and similarly for any point in $Q \setminus Q'$), and thus does not require an insertion to r-cover it. Moreover, no point $p \in P \setminus P'$ can be used to r-cover a point $q \in Q'$ as then $\|p - q\| \leq r$ which would imply $q \in Q \setminus Q'$. Finally, any point added to Q' in $\mathsf{real}(d^{\mathtt{I}}_{\mathcal{H}}(P',Q',k))$ to r-cover a point in P' will then itself be r-covered by P' and hence does not require points in $P \setminus P'$ to cover it.

Next observe that $d^{\mathtt{I}}_{\mathcal{H}}(P',Q',k) \leq r$ if and only if $\min_{0\leq k'\leq k} \max\{d^{\mathtt{I}}_{\mathcal{H}}(P',\emptyset,k'), d^{\mathtt{I}}_{\mathcal{H}}(\emptyset,Q',k-k')\} \leq r$, as shown by the following series of implications, where $\mathcal{X}$ is the set of all pairs of point sets $(\mathcal{C}_{P'},\mathcal{C}_{Q'})$ such that $|\mathcal{C}_{Q'}| + |\mathcal{C}_{P'}| \leq k$ and $||c - Q'|| \leq r$ for all $c \in \mathcal{C}_{P'}$ and $||c - P'|| \leq r$ for all $c \in \mathcal{C}_{Q'}$. (Note that $\mathcal{C}_{P'}$ will be the subset of points we are inserting into P' to r-cover points in Q', so it suffices to restrict to subsets such that every point in $\mathcal{C}_{P'}$ is within distance r of a point in Q'.)

For readability, we write "$\exists$ s.t. " as shorthand for "$\exists(\mathcal{C}_{P'},\mathcal{C}_{Q'}) \in \mathcal{X}$ *s.t.* " below.

$$\begin{aligned}
& d^{\mathtt{I}}_{\mathcal{H}}(P',Q',k) \leq r \\
\Leftrightarrow\ & \exists \text{ s.t. } d_{\mathcal{H}}(P' \cup \mathcal{C}_{P'}, Q' \cup \mathcal{C}_{Q'}) \leq r \\
\Leftrightarrow\ & \exists \text{ s.t. } \max\{d_h(P' \cup \mathcal{C}_{P'}, Q' \cup \mathcal{C}_{Q'}), \\
& \qquad d_h(Q' \cup \mathcal{C}_{Q'}, P' \cup \mathcal{C}_{P'})\} \leq r \\
\Leftrightarrow\ & \exists \text{ s.t. } \max\{d_h(P', Q' \cup \mathcal{C}_{Q'}), d_h(\mathcal{C}_{P'}, Q' \cup \mathcal{C}_{Q'}), \\
& \qquad d_h(Q', P' \cup \mathcal{C}_{P'}), d_h(\mathcal{C}_{Q'}, P' \cup \mathcal{C}_{P'})\} \leq r \\
\Leftrightarrow\ & \exists \text{ s.t. } \max\{d_h(P', \mathcal{C}_{Q'}), d_h(\mathcal{C}_{P'}, Q' \cup \mathcal{C}_{Q'}), \\
& \qquad d_h(Q', \mathcal{C}_{P'}), d_h(\mathcal{C}_{Q'}, P' \cup \mathcal{C}_{P'})\} \leq r \\
\Leftrightarrow\ & \exists \text{ s.t. } \max\{d_h(P', \mathcal{C}_{Q'}), d_h(Q', \mathcal{C}_{P'})\} \leq r \\
& \qquad \text{since } \mathcal{C}_{Q'} \text{ and } \mathcal{C}_{P'} \text{ must be } r\text{-covered} \\
\Leftrightarrow\ & \exists \text{ s.t. } \max\{d_{\mathcal{H}}(P', \mathcal{C}_{Q'}), d_{\mathcal{H}}(Q', \mathcal{C}_{P'})\} \leq r \\
& \qquad \text{since the other direction must be } \leq r. \\
\Leftrightarrow\ & \min_{0\leq k'\leq k} \max\{d^{\mathtt{I}}_{\mathcal{H}}(P',\emptyset,k'), d^{\mathtt{I}}_{\mathcal{H}}(\emptyset,Q',k-k')\} \leq r
\end{aligned}$$

Recall that by Lemma 1 and Lemma 2, for any point set S and integer ℓ, $d^{\mathtt{I}}_{\mathcal{H}}(S,\emptyset,\ell) = d^{\mathtt{E}}_{\mathcal{H}}(S,\emptyset,\ell) = kcenter(S,\ell)$. Thus, putting everything together,

$$\begin{aligned}
& d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) \leq r \quad \text{if any only if} \qquad (3.1) \\
& \min_{0\leq k'\leq k} \max\{kcenter(P',k'), kcenter(Q',k-k')\} \leq r.
\end{aligned}$$

Now at the end of $\mathbf{HausEdit}(P,Q,k,r)$, $\beta = \min_{0\leq k'\leq k} \max\{\mathbf{kcen}(P',k'), \mathbf{kcen}(Q',k-k')\}$. As $\mathbf{kcen}(S,\ell)$ is an α-approximation to $kcenter(S,\ell)$ for any point set S and integer $\ell \geq 0$, Eq. (3.1) then implies that if $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) \leq r$ then $\beta \leq \alpha r$ and so the algorithm returns True. Conversely, if $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) > \alpha r$ then again by Eq. (3.1) we have $\beta > \alpha r$ as $\mathbf{kcen}(S,\ell) \geq kcenter(S,\ell)$ for any set S and integer $\ell \geq 0$, and thus the algorithm returns False.

As for the running time, the sets P' and Q' can be computed in $O(mn)$ time. The running times of $\mathbf{kcen}(P',k')$ and $\mathbf{kcen}(Q',k-k')$ are respectively bounded by $T(n,k)$ and $T(m,k)$, and there are $k+1$ possible values for k'. Thus the overall running time is $O\big(mn + k\cdot T(\max\{m,n\},k)\big)$. □

Lemma 5 *Let R be the set consisting of 0 together with all pairwise distances in the set $P \cup Q$. Then there exists value $\rho \in R$ such that $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) \leq \rho \leq 2d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$.*

Proof. Eq. (3.1) from the proof of Lemma 4 implies that if $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k) \neq 0$ then it either occurs at a value

[3]We assume $T(n,k)$ is an increasing function of n and k.

$kcenter(P', k')$ or $kcenter(Q', k - k')$ for some k' and sets $P' \subseteq P$ and $Q' \subseteq Q$, or at a value of r where the P' and Q' sets change.

First, note that P' and Q' are determined by removing all points within distance r of any point in Q or P, respectively. Thus the set of distances between a point in P and a point in Q captures all values of r where the sets P' and Q' might change.

For any point set S and integer $\ell \geq 0$, the greedy algorithm of Gonzalez [16] achieves a 2-approximation to $kcenter(S, \ell)$, while only placing centers at points in S. (Recall S may be a strict subset from a metric on a larger point set X.) Thus if $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$ is achieved at some $kcenter(P', k')$ or $kcenter(Q', k - k')$ value, then there is some value ρ in the set of all pairwise distances in P or all pairwise distances in Q such that $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k) \leq \rho \leq 2d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$.

Combining the cases we have that there exists value $\rho \in R$ such that $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k) \leq \rho \leq 2d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$. □

Lemma 6 *Let P be a set of n points, Q be a set of m points, and $k \geq 0$ be an integer. Let* $\mathbf{kcen}(P, k)$ *be an algorithm which for some constant $1 < \alpha \leq 2$ returns an α-approximation to $kcenter(P, k)$ in $O(T(n, k))$ time. Then for any constant $\varepsilon > 0$, one can compute an $(\alpha + \varepsilon)$-approximation to $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$ in $O\big((mn + k^2 \max\{m, n\}) \log(mn) + k \cdot T(\max\{m, n\}, k)\big)$ time.*

Proof. Let R be the set of values from Lemma 5. We sort R and then binary search over it with $\mathbf{HausEdit}(P, Q, k, r)$, where for the $\mathbf{kcen}(P, k)$ subroutine we use the standard $O(nk)$ time 2-approximation algorithm for $kcenter(P, k)$ due to Gonzalez [16] (which may, for now, differ from the α-approximate subroutine from the current lemma statement). This ultimately yields a pair of values $r' < r^*$ that are consecutive in the sorted order of R, such that $\mathbf{HausEdit}(P, Q, k, r')$ returned False and $\mathbf{HausEdit}(P, Q, k, r^*)$ returned True.[4] Lemma 4 then implies $r' < d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k) \leq 2r^*$.

By Lemma 5 there exists a value $\rho \in R$ such that $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k) \leq \rho \leq 2d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$. Thus by Lemma 4 for any $x \geq \rho$, $\mathbf{HausEdit}(P, Q, k, x)$ must return True. This implies $r' < \rho$ as $\mathbf{HausEdit}(P, Q, k, r')$ returned False, and as $\rho \in R$ and r^* is the next value in the sorted order of R after r', this in turn implies $r^* \leq \rho$. Combining this with the above inequalities gives $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k) \leq 2r^* \leq 2\rho \leq 4d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$, or equivalently $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k) \in [r^*/2, 2r^*]$.

Consider the set of values $X = \{x_0, x_1, \ldots, x_z\}$, where $x_i = \frac{r^*}{2}(1 + \varepsilon/2)^i$ and $z = \lceil \log_{1+\varepsilon/2}(4) \rceil = O(1)$ for any constant $\varepsilon > 0$. Note that for any i we have $x_{i+1}/x_i = (1 + \varepsilon/2)$, and z was chosen such that $x_z \geq 2r^*$. Now we binary search over X for an adjacent pair $x_j, x_{j+1} \in X$ such that $\mathbf{HausEdit}(P, Q, k, x_j)$ returns False and $\mathbf{HausEdit}(P, Q, k, x_{j+1})$ returns True, except this time we use the α-approximate subroutine for $\mathbf{kcen}(P, k)$ from the current lemma statement. (By the above $\mathbf{HausEdit}(P, Q, k, x_z)$ must return True, so not all in X can return False, and if all return True then αx_0 is the desired approximation.) Since $\mathbf{HausEdit}(P, Q, k, x_j)$ returns False, by Lemma 4, $x_j < d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$, and as $x_{j+1}/x_j = (1 + \varepsilon/2)$, this implies that $x_{j+1} < (1 + \varepsilon/2)d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$. Since $\mathbf{HausEdit}(P, Q, k, x_{j+1})$ returns True we have $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k) \leq \alpha x_{j+1}$. Thus $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k) \leq \alpha x_{j+1} \leq \alpha(1 + \varepsilon/2)d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k) \leq (\alpha + \varepsilon)d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$ since $\alpha \leq 2$. In other words, αx_{j+1} is the desired $(\alpha + \varepsilon)$-approximation to $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$.

As for the running time, the set R can be computed and sorted in $O(mn \log(mn))$ time. Each of the $O(\log(mn))$ calls to $\mathbf{HausEdit}(P, Q, k, r)$ when using the $O(nk)$ time 2-approximation for $\mathbf{kcen}(P, k)$ take $O(mn + k^2 \max\{m, n\})$ time. The $O(1)$ calls to $\mathbf{HausEdit}(P, Q, k, x_i)$ when using the $O(T(n, k))$ time α-approximation for $\mathbf{kcen}(P, k)$ take $O\big(mn + k \cdot T(\max\{m, n\}, k)\big)$ time.[5] Thus the total time is $O\big((mn + k^2 \max\{m, n\}) \log(mn) + k \cdot T(\max\{m, n\}, k)\big)$ as claimed. □

Using the standard $O(nk)$ time 2-approximation algorithm for $kcenter(P, k)$ due to Gonzalez [16] for $\mathbf{kcen}(P, k)$ in Lemma 6 directly gives the following.

Theorem 7 *Given a set P of n points, a set Q of m points, and an integer $k \geq 0$, then for any chosen constant $\varepsilon > 0$, one can compute $(2 + \varepsilon)$-approximation in $O((mn + k^2 \max\{m, n\}) \log(mn))$ time.*

In Appendix A.1 we show that for points in low dimensional Euclidean space we can get improved running times and approximations by using grids and WSPDs.

Theorem 8 *Given a set $P \subset \mathbb{R}^d$ of n points, a set $Q \subset \mathbb{R}^d$ of m points, and an integer $k \geq 0$, where d is a constant and $N = \max\{m, n\}$, then for any chosen constant $\varepsilon > 0$, one can compute a:*

- *$(2 + \varepsilon)$-approximation in $O\big(k^2 N \log(N)\big)$ time.*
- *$(1 + \varepsilon)$-approximation in $O\big(k^2 N \log(N)\big) + (k/\varepsilon)^{O(k^{1-1/d})}$ time.*

3.1 Separate Budgets

Let $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k, \ell)$ be the Hausdorff Separate Budget Edit Distance, which differs from the Hausdorff Edit Distance by separating the budgets for insertion and

[4] Since $0 \in R$, Lemma 4 implies if it returned True for all $r \in R$ then $d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k) = 0$. Since $\rho \in R$ (as described in Lemma 5), it cannot return False for all $r \in R$ as Lemma 4 then implies $\rho < d_{\mathcal{H}}^{\mathsf{E}}(P, Q, k)$.

[5] For $0 < \varepsilon \leq 1$, we are searching over $z = \lceil \log_{1+\varepsilon/2}(4) \rceil = O(1/\log(1 + \varepsilon)) = O(1/\varepsilon)$ values. Thus more precisely we make $O(\log(1/\varepsilon))$ calls, which is a constant when assuming ε is constant.

deletion, which are noted as k and ℓ, respectively. Below is our main theorem from this section, where *kcenterOut*(P, k, ℓ) denotes the k-center with ℓ outliers objective. All other details are in Appendix A.2

Theorem 9 *Let P be a set of n points, Q be a set of m points, and let $k, \ell \geq 0$ be integers. Let* $\mathbf{kcen}(P, k, \ell)$ *be an algorithm which returns an α-approximation to kcenterOut(P, k, ℓ) in $O(T(n, k, \ell))$ time, for some constant $\alpha > 1$. Then for any constant $\varepsilon > 0$, one can compute an $(\alpha + \varepsilon)$-approximation to $d^{\mathtt{E}}_{\mathcal{H}}(P, Q, k, \ell)$ in $O\big((mn + k\ell \cdot T(\max\{m, n\}, k, \ell)) \log(mn)\big)$ time.*

3.2 Deletion Only

Let $d^{\mathtt{D}}_{\mathcal{H}}(P, Q, k)$ be the Hausdorff Deletion Distance, which minimizes $d_{\mathcal{H}}\big(\mathtt{D}(P, k'), \mathtt{D}(Q, k - k')\big)$ where $0 \leq k' \leq k$. This is very similar to the Partial Hausdorff Distance [19], as discussed in the introduction, though unlike [19], here we provide a formal algorithm and analysis. (Indeed, the lemma below provides the formal argument of equivalence of the problems for separated budgets, as discussed in Appendix A.4.)

Algorithm 2: HausDel(P, Q, k)

```
1 If $k \geq n + m$ then **return** 0. If $P = \emptyset$ or $Q = \emptyset$
    **return** $\infty$.
2 Compute the multisets $D^P = \{\|p - Q\| \mid p \in P\}$
    and $D^Q = \{\|q - P\| \mid q \in Q\}$.
3 Let $\mathfrak{D} = \langle \mathfrak{D}_1, \mathfrak{D}_2, \ldots, \mathfrak{D}_{n+m} \rangle$ denote the values
    in $D = D^P \uplus D^Q$ listed in decreasing order.
4 **return** $\mathfrak{D}_{k+1}$
```

In Algorithm 2 and throughout this section, $X \uplus Y$ denotes the mutiset union (i.e. additive union) of two multisets X and Y. Thus $|X \uplus Y| = |X| + |Y|$.

Lemma 10 *Let P be a set of n points, Q be a set of m points, and $k \geq 0$ an integer. Then* $\mathbf{HausDel}(P, Q, k)$ *computes $d^{\mathtt{D}}_{\mathcal{H}}(P, Q, k)$ in $O(nm)$ time.*

Proof. The first line of the algorithm handles some trivial cases, and in the remainder of the proof we assume $k < n + m$ and neither $P \neq \emptyset$ nor $Q \neq \emptyset$.

Let $\mathfrak{D}(X, Y) = \langle \mathfrak{D}(X, Y)_1, \ldots \mathfrak{D}(X, Y)_{|X|+|Y|} \rangle$ be the list of (bi-chromatic) nearest neighbor distances between any sets X and Y, sorted in descending order. Thus $d_{\mathcal{H}}(X, Y) = \mathfrak{D}(X, Y)_1$, and note that in the algorithm $\mathbf{HausDel}(P, Q, k)$, we have $\mathfrak{D} = \mathfrak{D}(P, Q)$.

Consider any pair of sets $P' \in \mathtt{D}(P, k'), Q' \in \mathtt{D}(Q, k - k')$ for any value $0 \leq k' \leq k$. Observe that when we delete points from P and Q, producing the sets P' and Q', the nearest neighbor distances of the remaining points cannot decrease. Thus the list $\mathfrak{D}(P', Q')$ is obtained from the list $\mathfrak{D}(P, Q)$ by deleting up to k values from the list, and mapping each remaining value $\mathfrak{D}(P, Q)_i$ to a unique value $\mathfrak{D}(P', Q')_j$ such that $\mathfrak{D}(P, Q)_i \leq \mathfrak{D}(P', Q')_j$. Thus $\mathfrak{D}(P, Q)_{k+1} \leq \mathfrak{D}(P', Q')_1$, as clearly over the space of all such mappings of the list $\mathfrak{D}(P, Q)$, the resulting list with the minimum first value would arise from deleting the top k values and not increasing the $(k + 1)$th value.

Let z be the largest index such that $z \leq k$ and $\mathfrak{D}(P, Q)_z > \mathfrak{D}(P, Q)_{k+1}$ ($z = 0$ if no such index exists). Now let $\langle x_1, \ldots, x_{n+m} \rangle$ denote the ordered list of points from P and Q corresponding to the values $\langle \mathfrak{D}(P, Q)_1, \ldots, \mathfrak{D}(P, Q)_{n+m} \rangle$, and let $\hat{P} \subseteq P$ and $\hat{Q} \subseteq Q$ be the sets resulting from deleting $x_1, \ldots, x_z$ (no points are deleted if $z = 0$). We now argue that $\mathfrak{D}(P, Q)_{k+1} = \mathfrak{D}(\hat{P}, \hat{Q})_1$, which, since we already argued that $\mathfrak{D}(P, Q)_{k+1}$ was a lower bound on any possible deletion of k points, implies $d^{\mathtt{D}}_{\mathcal{H}}(P, Q, k) = \mathfrak{D}(P, Q)_{k+1}$.

Observe that for any $i > z$, no point in $x_1, \ldots, x_z$ can be the nearest neighbor of x_i from the other set (i.e. realizing $\mathfrak{D}(P, Q)_i$), as this would imply x_z has a neighbor from the other set in distance less than $\mathfrak{D}(P, Q)_z$, a contradiction. This implies $\mathfrak{D}(\hat{P}, \hat{Q}) = \langle \mathfrak{D}(P, Q)_{z+1}, \ldots, \mathfrak{D}(P, Q)_{n+m} \rangle$. Thus we have $\mathfrak{D}(P, Q)_{k+1} = \mathfrak{D}(\hat{P}, \hat{Q})_1$ as claimed, since by the definition of z we have that $\mathfrak{D}(P, Q)_{z+1} = \mathfrak{D}(P, Q)_{k+1}$.

As for the running time, the multiset $D = D^P \uplus D^Q$ can be computed in $O(nm)$ time by looking at all pairwise distances. The return value $\mathfrak{D}_{k+1}$ is the $k + 1$ largest value in the multiset D, which can be computed in $O(|D|) = O(n + m)$ time using the standard linear time median selection algorithm [8], thus the total time is dominated by the time to compute D. ☐

In the above algorithm it takes $O(nm)$ to compute the multiset of all bi-chromatic nearest neighbor distances, namely $D = D^P \uplus D^Q$. In the plane, these distances can be computed more efficiently using Voronoi diagrams. Namely, it takes $O(n \log n)$ time to compute the Voronoi diagram of P, which allows for $O(\log n)$ time point location queries. Thus, by querying all of Q, we get D^Q in $O((n + m) \log n)$ time. Similarly, D^P can be computed in $O((n + m) \log m)$ time. As returning the $k + 1$ largest value in D takes $O(n + m)$ time, we have the following.

Corollary 11 *Let $P \subset \mathbb{R}^2$ be a set of n points, $Q \subset \mathbb{R}^2$ be a set of m points, and $k \geq 0$ be an integer. Let $N = \max\{m, n\}$, then one can compute $d^{\mathtt{D}}_{\mathcal{H}}(P, Q, k)$ in $O\big(N \log(N)\big)$ time.*

In Appendix A.3 we argue that by using Approximate Nearest Neighbor data structures one can achieve a fast $(1 + \varepsilon)$-approximation for points in $\mathbb{R}^d$, for constant d.

Corollary 12 *Let $P \subset \mathbb{R}^d$ be a set of n points, $Q \subset \mathbb{R}^d$ be a set of m points, and $k \geq 0$ be an integer, where d is a constant. Let $N = \max\{m, n\}$, then one can compute* *a* $(1+\varepsilon)$*-approximation to* $d^{\mathtt{D}}_{\mathcal{H}}(P,Q,k)$ *in* $O\big(N\log(N)+N/\varepsilon^d\big)$ *time.*

Appendix A.4 discusses how our above results for the deletion only and general Hausdorff Edit Distance can be extended to other natural variants.

References

[1] P. K. Agarwal, K. Fox, J. Pan, and R. Ying. Approximating dynamic time warping and edit distance for a pair of point sequences. In *Proceedings of the 32nd International Symposium on Computational Geometry*, pages 6:1–6:16, 2016.

[2] P. K. Agarwal, S. Har-Peled, M. Sharir, and Y. Wang. Hausdorff distance under translation for points and balls. *ACM Trans. Algorithms*, 6(4):71:1–71:26, 2010.

[3] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.

[4] R. B. Avraham, M. Henze, R. Jaume, B. Keszegh, O. E. Raz, M. Sharir, and I. Tubis. Partial-matching RMS distance under translation: Combinatorics and algorithms. *Algorithmica*, 80(8):2400–2421, 2018.

[5] L. Biabani, A. Hennes, M. Monemizadeh, and M. Schmidt. Faster query times for fully dynamic k-center clustering with outliers. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 9226–9247. Curran Associates, Inc., 2023.

[6] D. Chakrabarty, P. Goyal, and R. Krishnaswamy. The non-uniform k-center problem. *ACM Trans. Algorithms*, 16(4), June 2020.

[7] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, page 642–651, USA, 2001. Society for Industrial and Applied Mathematics.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[9] M. de Berg, M. Monemizadeh, and Y. Zhong. k-Center Clustering with Outliers in the Sliding-Window Model. In P. Mutzel, R. Pagh, and G. Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[10] H. Ding, H. Yu, and Z. Wang. Greedy Strategy Works for k-Center Clustering with Outliers and Coreset Construction. In M. A. Bender, O. Svensson, and G. Herman, editors, *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[11] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 434–444, New York, NY, USA, 1988. Association for Computing Machinery.

[12] A. Fischer, C. Y. Suen, V. Frinken, K. Riesen, and H. Bunke. Approximation of graph edit distance based on hausdorff matching. *Pattern Recognition*, 48(2):331–343, 2015.

[13] E. Fox, A. Nayyeri, J. J. Perry, and B. Raichel. Fréchet Edit Distance. In W. Mulzer and J. M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 58:1–58:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[14] K. Fox and X. Li. Approximating the geometric edit distance. *Algorithmica*, 84(9):2395–2413, 2022.

[15] O. Gold and M. Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. *ACM Transactions on Algorithms*, 14(4):50, 2018.

[16] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[17] S. Har-peled. *Geometric Approximation Algorithms.* American Mathematical Society, USA, 2011.

[18] W.-L. Hsu and G. L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979.

[19] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

A Hausdorff Edit Distance Continued

A.1 The Euclidean Case

Throughout, P and Q have been finite point sets in an arbitrary metric. If instead we restrict P and Q to be finite point sets in $\mathbb{R}^d$ with the standard Euclidean metric, then when d is a constant, the above result can be improved both in terms of running time and approximation quality (though the latter requires bounding k). We now sketch how to achieve this.

For the running time, the mn term in Theorem 7 arises due to two reasons. The first reason is on Line 1 in **HausEdit**(P,Q,k,r). Here the algorithm marks a set P_r of points which is defined as all $p \in P$ within distance r of some point $q \in Q$ (and similarly define Q_r). Analogously define the set $P_{\alpha r}$ consisting of all $p \in P$ within distance αr of some point $q \in Q$. Observe that the argument of correctness for the algorithm is identical if instead of precisely marking P_r we instead mark any subset $\hat{P} \subseteq P$ such that $P_r \subseteq \hat{P} \subseteq P_{\alpha r}$. When d is a constant, finding such a set $\hat{P}$ can be done in $O(n+m)$ time using standard grid based techniques, see for example [17].[6] Thus using grids, the run time of **HausEdit**(P,Q,k,r) becomes $O\big(m+n+k\cdot T(\max\{m,n\},k)\big) = O\big(k\cdot T(\max\{m,n\},k)\big)$.

The other reason for the mn term in Theorem 7 is from the use of the $O(mn)$ sized set R from Lemma 5, consisting of all pairwise distances in $P\cup Q$. However, if $P,Q\subset\mathbb{R}^d$ for any constant d, then we can use WSPD's (see for example [17]) to approximate the values from the set R in Lemma 5. In particular, in $O((m+n)\log(m+n)+(m+n)/\varepsilon^d)$ time one can construct a set R' of size $O((m+n)/\varepsilon^d)$ such that for any value $r\in R$ there exists a value $r'\in R'$ where $r\le r'\le(1+\varepsilon)r$. Thus in Lemma 6 rather than binary searching over R, we can binary search over R'. Recall that there the binary search over R was used to construct a constant spread interval containing $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$. If instead we binary search over R' this will produce an interval containing $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k)$ that is a $(1+\varepsilon)$ factor larger, and as all we require is that the interval has constant spread, we can in fact set $\varepsilon=1$. The rest of the proof, which then searches over this constant spread interval, remains the same and the running time then becomes $O\big((m+n+k^2\max\{m,n\})\log(mn)+k\cdot T(\max\{m,n\},k)\big) = O\big(k^2\max\{m,n\}\log(mn)+k\cdot T(\max\{m,n\},k)\big)$.

Using the 2-approximation of Gonzalez for **kcen** we thus get a $O\big(k^2\max\{m,n\}\log(mn)\big)$ time $(2+\varepsilon)$-approximation. On the other hand, for a point set $P\subset\mathbb{R}^d$ for constant d, [3] gave an $O(n\log k)+(k/\varepsilon)^{O(k^{1-1/d})}$ time $(1+\varepsilon)$-approximation to $kcenter(P,k)$. Thus instead using this for **kcen** gives an $O(k^2\max\{m,n\}\log(mn))+(k/\varepsilon)^{O(k^{1-1/d})}$ time $(1+\varepsilon)$-approximation. Thus in summary we have the following.

Theorem 8 (Restated) *Given a set $P\subset\mathbb{R}^d$ of n points, a set $Q\subset\mathbb{R}^d$ of m points, and an integer $k\ge0$, where d is a constant and $N=\max\{m,n\}$, then for any chosen constant $\varepsilon>0$, one can compute a:*

- *$(2+\varepsilon)$-approximation in $O\big(k^2N\log(N)\big)$ time.*
- *$(1+\varepsilon)$-approximation in $O\big(k^2N\log(N)\big)+(k/\varepsilon)^{O(k^{1-1/d})}$ time.*

A.2 Separate Budgets Continued

Here we give the full details from Section 3.1.

Let $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k,\ell)$ be the Hausdorff Separate Budget Edit Distance, which differs from the Hausdorff Edit Distance by separating the budgets for insertion and deletion, which are noted as k and ℓ, respectively. Despite Lemma 1 not holding for this problem, we will prove nearly equivalent results as those shown for Hausdorff Edit Distance.

First, observe that $d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k)=d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k,0)$. As Theorem 3 showed computing $d^{\mathtt{I}}_{\mathcal{H}}(P,Q,k)$ is APX-Hard, we immediately have the following corollary.

[6] As a rough sketch, consider the uniform grid of cell side length $(\alpha-1)r/\sqrt{d}$ and diameter $(\alpha-1)r$. Hash all points from P into the cells of this grid. For every point $q\in Q$, there are $O((2(\alpha-1)/\sqrt{d})^d)=O(1)$ cells intersecting a ball of radius r around q, and any point in these cells is at most $r+(\alpha-1)r=\alpha r$ away from q. Thus we simply mark all points from P contained in any such cell around a point of Q.

Corollary 13 *For any point sets P,Q and integers $k,\ell\ge0$, the problem of computing $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k,\ell)$ is APX-Hard. The problem remains APX-Hard even when $P,Q\subset\mathbb{R}^2$.*

Recall that $\mathtt{D}(P,\ell)$ denotes the set of all subsets of P resulting from deleting at most ℓ points from P. Then for the k-center clustering with outliers problem [6, 7] the objective is to compute $kcenterOut(P,k,\ell)=\min_{P^{\mathtt{D}}\in\mathtt{D}(P,\ell)}\big(kcenter(P^{\mathtt{D}},k)\big)$.

Lemma 14 *For any finite point set P and integers $k,\ell\ge0$, $d^{\mathtt{E}}_{\mathcal{H}}(P,\emptyset,k,\ell)=kcenterOut(P,k,\ell)$.*

Proof. $d^{\mathtt{E}}_{\mathcal{H}}(P,\emptyset,k,\ell)=\min_{P^{\mathtt{D}}\in\mathtt{D}(P,\ell)}\big(d^{\mathtt{I}}_{\mathcal{H}}(P^{\mathtt{D}},\emptyset,k)\big)$ and by definition $kcenterOut(P,k,\ell)=\min\limits_{P^{\mathtt{D}}\in\mathtt{D}(P,\ell)}\big(kcenter(P^{\mathtt{D}},k)\big)$. The lemma statement then follows by Lemma 1 and Lemma 2, which when combined say that $d^{\mathtt{I}}_{\mathcal{H}}(P,\emptyset,k)=kcenter(P,k)$. ☐

We are also able to construct an equivalent decision algorithm, **HausEdit**(P,Q,k,ℓ,r), which uses as a subroutine any α-approximation algorithm for $kcenterOut(P,k,\ell)$, which we denote by **kcen**(P,k,ℓ).

Algorithm 3: HausEdit(P,Q,k,ℓ,r)

```
1  Mark all $p \in P$ such that $\|p - Q\| \le r$.
     Mark all $q \in Q$ such that $\|q - P\| \le r$.
2  Create sets $P' \subseteq P$ and $Q' \subseteq Q$ by removing all
     marked points.
3  $\beta = \infty$
4  for $\ell' = 0$ to $\ell$ do
5      for $k' = 0$ to $k$ do
6          $\beta = \min\{\beta, \max\{\mathbf{kcen}(P', k', \ell'),$
                          $\mathbf{kcen}(Q', k - k', \ell - \ell')\}\}$
7  if $\beta \le \alpha r$ then        // kcen$(P,k,\ell)$ is an $\alpha$-approx.
8      return True
9  else
10     return False
```

Lemma 15 *Let P and Q be sets of n and m points. Let $k,\ell\ge0$ be integers, and* **kcen**(P,k,ℓ) *an algorithm returning an α-approximation to $kcenterOut(P,k,\ell)$ in $O(T(n,k,\ell))$ time.[7] Then if* **HausEdit**(P,Q,k,ℓ,r) *returns True then $r\ge d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k,\ell)/\alpha$, and if* **HausEdit**$(P,Q,k,\ell,r)$ *returns False then $r<d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k,\ell)$. The running time of* **HausEdit**(P,Q,k,ℓ,r) *is $O\big(mn+k\ell\cdot T(\max\{m,n\},k,\ell)\big)$.*

Proof. The proof follows the same logic as that of Lemma 4, though with some changes required. For clarity, we reproduce the entire proof with the relevant changes.

Let P',Q' be as defined in **HausEdit**(P,Q,k,ℓ,r). First, observe that $d^{\mathtt{E}}_{\mathcal{H}}(P,Q,k,\ell)\le r$ if and only if $d^{\mathtt{E}}_{\mathcal{H}}(P',Q',k,\ell)\le r$. This follows as any point in $P\setminus P'$ is already within distance r to a point in $Q\setminus Q'$, and vice versa. Thus points in $P\setminus P'$ and $Q\setminus Q'$ are r-covered without

[7] We assume $T(n,k,\ell)$ is an increasing function of n, k, and ℓ.

the need for any insertions or deletions, and any deletions from P' or Q' will not change this fact. Moreover, points in $P \setminus P'$ do not r-cover any point in Q' (nor do points in $Q \setminus Q'$ r-cover points in P'), and any point added to Q' in $\mathsf{real}(d^{\mathtt{I}}_{\mathcal{H}}(P', Q', k, \ell))$ to r-cover a point in P' that is not deleted, will then itself be r-covered and hence does not require points in $P \setminus P'$ to cover it.

Next we argue that $d^{\mathtt{E}}_{\mathcal{H}}(P', Q', k, \ell) \leq r$ if and only if $\min_{0\leq \ell'\leq \ell, 0\leq k'\leq k} \max\{\mathit{kcenterOut}(P', k', \ell'), \mathit{kcenterOut}(Q', k-k', \ell-\ell')\} \leq r$. Let $\mathcal{X}$ represent the set of tuples $(P^{\mathtt{D}}, Q^{\mathtt{D}}, \mathcal{C}_{P^{\mathtt{D}}}, \mathcal{C}_{Q^{\mathtt{D}}})$ such that $P^{\mathtt{D}} \in \mathtt{D}(P', \ell')$ and $Q^{\mathtt{D}} \in \mathtt{D}(Q', \ell-\ell')$ for some integer $0 \leq \ell' \leq \ell$, and such that $|\mathcal{C}_{P^{\mathtt{D}}}| + |\mathcal{C}_{Q^{\mathtt{D}}}| \leq k$ where $||c - Q^{\mathtt{D}}|| \leq r$ for all $c \in \mathcal{C}_{P^{\mathtt{D}}}$ and $||c - P^{\mathtt{D}}|| \leq r$ for all $c \in \mathcal{C}_{Q^{\mathtt{D}}}$. (Without loss of generality, we view all deletions as occurring before any insertion, thus first producing the sets $P^{\mathtt{D}}$ and $Q^{\mathtt{D}}$. Note the set $\mathcal{C}_{P^{\mathtt{D}}}$ will then be the subset of points we are inserting into $P^{\mathtt{D}}$ to r-cover points in $Q^{\mathtt{D}}$, so it suffices to restrict to subsets such that every point in $\mathcal{C}_{P^{\mathtt{D}}}$ is within distance r of a point in $Q^{\mathtt{D}}$.) For readability, we write "$\exists$ s.t. " as shorthand for "$\exists (P^{\mathtt{D}}, Q^{\mathtt{D}}, \mathcal{C}_{P^{\mathtt{D}}}, \mathcal{C}_{Q^{\mathtt{D}}}) \in \mathcal{X}$ *s.t.* " below.

$$\begin{aligned}
& d^{\mathtt{E}}_{\mathcal{H}}(P', Q', k, \ell) \leq r \\
\Leftrightarrow\ & \exists \text{ s.t. } \ d_{\mathcal{H}}(P^{\mathtt{D}} \cup \mathcal{C}_{P^{\mathtt{D}}}, Q^{\mathtt{D}} \cup \mathcal{C}_{Q^{\mathtt{D}}}) \leq r \\
\Leftrightarrow\ & \exists \text{ s.t. } \ \max\{d_h(P^{\mathtt{D}} \cup \mathcal{C}_{P^{\mathtt{D}}}, Q^{\mathtt{D}} \cup \mathcal{C}_{Q^{\mathtt{D}}}), \\
& \qquad d_h(Q^{\mathtt{D}} \cup \mathcal{C}_{Q^{\mathtt{D}}}, P^{\mathtt{D}} \cup \mathcal{C}_{P^{\mathtt{D}}})\} \leq r \\
\Leftrightarrow\ & \exists \text{ s.t. } \ \max\{d_h(P^{\mathtt{D}}, Q^{\mathtt{D}} \cup \mathcal{C}_{Q^{\mathtt{D}}}), d_h(\mathcal{C}_{P^{\mathtt{D}}}, Q^{\mathtt{D}} \cup \mathcal{C}_{Q^{\mathtt{D}}}), \\
& \qquad d_h(Q^{\mathtt{D}}, P^{\mathtt{D}} \cup \mathcal{C}_{P^{\mathtt{D}}}), d_h(\mathcal{C}_{Q^{\mathtt{D}}}, P^{\mathtt{D}} \cup \mathcal{C}_{P^{\mathtt{D}}})\} \leq r \\
\Leftrightarrow\ & \exists \text{ s.t. } \ \max\{d_h(P^{\mathtt{D}}, \mathcal{C}_{Q^{\mathtt{D}}}), d_h(\mathcal{C}_{P^{\mathtt{D}}}, Q^{\mathtt{D}} \cup \mathcal{C}_{Q^{\mathtt{D}}}), \\
& \qquad d_h(Q^{\mathtt{D}}, \mathcal{C}_{P^{\mathtt{D}}}), d_h(\mathcal{C}_{Q^{\mathtt{D}}}, P^{\mathtt{D}} \cup \mathcal{C}_{P^{\mathtt{D}}})\} \leq r \\
\Leftrightarrow\ & \exists \text{ s.t. } \ \max\{d_h(P^{\mathtt{D}}, \mathcal{C}_{Q^{\mathtt{D}}}), d_h(Q^{\mathtt{D}}, \mathcal{C}_{P^{\mathtt{D}}})\} \leq r \\
& \qquad \text{since } \mathcal{C}_{Q^{\mathtt{D}}} \text{ and } \mathcal{C}_{P^{\mathtt{D}}} \text{ must be } r\text{-covered} \\
\Leftrightarrow\ & \exists \text{ s.t. } \ \max\{d_{\mathcal{H}}(P^{\mathtt{D}}, \mathcal{C}_{Q^{\mathtt{D}}}), d_{\mathcal{H}}(Q^{\mathtt{D}}, \mathcal{C}_{P^{\mathtt{D}}})\} \leq r \\
& \qquad \text{since the other direction must be } \leq r \\
\Leftrightarrow\ & \min_{0\leq \ell'\leq \ell, 0\leq k'\leq k} \max\{d^{\mathtt{E}}_{\mathcal{H}}(P', \emptyset, k', \ell'), \\
& \qquad\qquad d^{\mathtt{E}}_{\mathcal{H}}(\emptyset, Q', k-k', \ell-\ell')\} \leq r
\end{aligned}$$

Recall that by Lemma 14, for any point set S and integers a and b, $d^{\mathtt{E}}_{\mathcal{H}}(S, \emptyset, a, b) = \mathit{kcenterOut}(S, a, b)$. Thus putting everything together we have that

$$\begin{aligned}
& d^{\mathtt{E}}_{\mathcal{H}}(P, Q, k, \ell) \leq r \ \text{ if and only if} \qquad (A.1)\\
& \min_{\substack{0\leq \ell'\leq \ell,\\ 0\leq k'\leq k}} \max\big\{\mathit{kcenterOut}(P', k', \ell'), \\
& \qquad\qquad \mathit{kcenterOut}(Q', k-k', \ell-\ell')\big\} \leq r.
\end{aligned}$$

Now at the end of **HausEdit**(P, Q, k, r), $\beta = \min_{0\leq \ell'\leq \ell, 0\leq k'\leq k} \max\{\mathbf{kcen}(P', k', \ell'), \mathbf{kcen}(Q', k-k', \ell-\ell')\}$. As $\mathbf{kcen}(S, a, b)$ is an α-approximation to $\mathit{kcenterOut}(S, a, b)$ for any point set S and integers $a, b \geq 0$, Eq. (A.1) then implies that if $d^{\mathtt{E}}_{\mathcal{H}}(P, Q, k, \ell) \leq r$ then $\beta \leq \alpha r$ and so the algorithm returns True. Conversely, if $d^{\mathtt{E}}_{\mathcal{H}}(P, Q, k, \ell) > \alpha r$ then again by Eq. (A.1) we have $\beta > \alpha r$ as $\mathbf{kcen}(S, a, b) \geq \mathit{kcenterOut}(S, a, b)$ for any set S and integers $a, b \geq 0$, and thus the algorithm returns False.

As for the running time, the sets P' and Q' can be computed in $O(mn)$ time. The running times of $\mathbf{kcen}(P', k', \ell')$ and $\mathbf{kcen}(Q', k-k', \ell-\ell')$ are respectively bounded by $T(n, k, \ell)$ and $T(m, k, \ell)$, and there are $\ell + 1$ possible values for ℓ' and $k+1$ possible values for k'. Thus the overall running time is $O\big(mn + k\ell \cdot T(\max\{m, n\}, k, \ell)\big)$. $\square$

It is easy to see that Lemma 5 applies to the separate budgets problem as well. Additionally, Lemma 6 also still applies as it is simply searching using the decision procedure. The analogous lemma for the separate budget case is given below as a theorem. Recall Lemma 6 first used the Gonzalez algorithm to get a constant spread interval, which was then searched over using an α-approximation. Here we assume the same α-approximation is used in both parts for simplicity, and as there is not a clear best algorithm to choose now. (For simplicity, Lemma 6 assumed $\alpha \leq 2$, though the proof extends to any constant α.)

Theorem 9 (Restated) *Let P be a set of n points, Q be a set of m points, and let $k, \ell \geq 0$ be integers. Let* $\mathbf{kcen}(P, k, \ell)$ *be an algorithm which returns an α-approximation to $\mathit{kcenterOut}(P, k, \ell)$ in $O(T(n, k, \ell))$ time, for some constant $\alpha > 1$. Then for any constant $\varepsilon > 0$, one can compute an $(\alpha + \varepsilon)$-approximation to $d^{\mathtt{E}}_{\mathcal{H}}(P, Q, k, \ell)$ in $O\big((mn + k\ell \cdot T(\max\{m, n\}, k, \ell)) \log(mn)\big)$ time.*

As discussed in the introduction, there are various known algorithms for k-center clustering with outliers, with different trade-offs. For example, if we use the simple greedy constant factor approximation of [7] in the above theorem, then we get a polynomial time constant factor approximation for $d^{\mathtt{E}}_{\mathcal{H}}(P, Q, k, \ell)$.

A.3 Argument for Corollary 12

Given a set $P \subset \mathbb{R}^d$ of n points, for a query point q, we refer to a point $x \in P$ as a $(1+\varepsilon)$ approximate nearest neighbor (ANN) of q if $\|x - q\| \leq (1+\varepsilon) \min_{p\in P} \|p - q\|$. It is known that for any point set $P \subset \mathbb{R}^d$ for constant d, in $O(n \log n)$ time one can construct a data structure such that given any query point q, in $O(\log n + 1/\varepsilon^d)$ time it returns a $(1+\varepsilon)$-ANN [17].

We can replace the multiset D of exact bi-chromatic nearest neighbor distances from **HausDel**(P, Q, k) (Algorithm 2) with a multiset D^{ε} of approximate bi-chromatic nearest neighbor distances as follows. For the point set P, in $O(n \log n)$ time we construct a $(1+\varepsilon)$-ANN data structure, which supports $O(\log n + 1/\varepsilon^d)$ time nearest neighbor queries, and then query all points in Q. Then we build such a structure for Q and query all points of P. Thus the multiset of all distances between queried points and their $(1+\varepsilon)$-ANN is the desired multiset D^{ε}, which took $O(n \log n + m(\log n + 1/\varepsilon^d) + m \log m + n(\log m + 1/\varepsilon^d)) = O(N \log N + N/\varepsilon^d)$ time to compute, where $N = \max\{m, n\}$.

Recall that **HausDel**(P, Q, k) returned the $k+1$ largest value in D, denoted $\mathfrak{D}_{k+1}$. Let $\mathfrak{D}^{\varepsilon}_{k+1}$ denote the $k+1$ largest value in D^{ε}. Clearly $\mathfrak{D}_{k+1} \leq \mathfrak{D}^{\varepsilon}_{k+1}$ as our $(1+\varepsilon)$-ANN distances are never smaller than the true nearest neighbor distances. Moreover, $\mathfrak{D}^{\varepsilon}_{k+1} \leq (1+\varepsilon)\mathfrak{D}_{k+1}$, as clearly the largest $\mathfrak{D}^{\varepsilon}_{k+1}$ could possibly be is if all ANN distances were exactly a $(1+\varepsilon)$ factor larger (the maximum allowed), in which case the value at any rank would be exactly a $(1+\varepsilon)$

factor larger. Note, the actual point realizing $\mathfrak{D}^{\varepsilon}_{k+1}$ may not correspond to the one realizing $\mathfrak{D}_{k+1}$, though it does not matter.

A.4 Additional Applications

Here we observe that the above results easily extend to other natural variants.

Set-Partitioned Budgets. In Section 3.1 we considered the version where we had separate insertion and deletion budgets k and ℓ. Now consider the version where these insertion and deletion budgets are further partitioned among P and Q, i.e. as part of the input we are given k_P, k_Q, ℓ_P, and ℓ_Q. Algorithm 3 immediately extends to this case if we simply remove the for loops and directly compute $\max\{\mathbf{kcen}(P', k_P, \ell_P), \mathbf{kcen}(Q', k_Q, \ell_Q)\}$, as indeed the purpose of the loops was to guess the partition of k and ℓ into k_P, k_Q, ℓ_P, and ℓ_Q. Thus we get an analogous version of Theorem 9, though without the $k\ell$ term in the running time.

For the problem where edits of either type are allowed though the edits are partitioned among P and Q, i.e. we are given k_P and k_Q, we can no longer assume there are no deletions, i.e. Lemma 1 does not hold. However, in this case we can simply modify the for loops in Algorithm 3 to guess the partition of k_P and k_Q into deletions and insertions, yielding an analogous version of Theorem 9, though where the $k\ell$ term in the running time is replaced with $k_P k_Q$.

Note that the version of the problem where edits are only allowed to one side, say P, is now a special case of the problem where $k_Q = 0$ (or $k_Q = 0$ and $\ell_Q = 0$). However, in this case, since we are considering the r decision problem, we must delete all points in P'. Thus we can use k-center rather than k-center with outliers (i.e. modify Algorithm 1 not Algorithm 3), yielding analogous versions of Theorem 7 and Theorem 8, where the k^2 terms in the running times improve to k.

Deletion Only Set-Partitioned Budgets. We can also consider the variant of the deletion only problem where, rather than being given a single deletion budget k, we are given separate deletion budgets k_P for P and k_Q for Q. This case is easily handled by modifying Algorithm 2. Specifically, there we had multisets D^P and D^Q, and we returned the $k+1$ largest element in $D^P \uplus D^Q$. Instead, we simply return the maximum of the $k_P + 1$ largest element in D^P and the $k_Q + 1$ largest element in D^Q. The running time and correctness of Lemma 10 and Corollary 11 still apply with only superficial changes in the arguments.

As discussed in the introduction, [19] defined the Partial Hausdorff Distance, where given parameters k and ℓ, we want the maximum of the kth smallest value in $\{\|p-Q\| \mid p \in P\}$ and the ℓth smallest value in $\{\|q-P\| \mid q \in Q\}$. This is precisely the deletion only Hausdorff distance for parameters $n-k$ and $m-\ell$, and in fact our proof of Lemma 10 formally argues the equivalence of finding the value of a given rank and the deletion only variant.

Substitutions. In this paper we allowed insertions and deletions for our edits. One could also consider substitutions of one point for another. However, this introduces many possible variants, as then one must decide whether they are shared or split edit budgets among P and Q, and among the three possible operations.

Observe that a substitution can be viewed as a deletion plus an insertion,[8] which suggests that some of these variants may be amenable to approximations using k-center clustering with outliers as done above. However, there are cases where one may wish to perform an insertion (with no deletion) over a substitution. In particular, this can occur when any further substitution would require moving (i.e. deleting) a point from $P \setminus P'$ or $Q \setminus Q'$. Thus depending on how the budgets are shared or not the problem may become more challenging, and may be a good direction to pursue for future research.

[8]We assume the insertion and deletion both occur within P or both within Q, though you could create even more variants, where the insertion and deletion can occur in different sets.

Minimum Anchored Steiner Trees with Removable Obstacles[*]

Stephane Durocher[†] Arman Heydari[‡] J. Mark Keil[‡] Debajyoti Mondal[‡]

Abstract

Given a set P of point sites and a set S of polygonal obstacles in the plane, where $P \cap S = \varnothing$, an *anchored Steiner tree (AST)* of P is a tree with line segments as edges that spans all the sites while avoiding the interiors of the obstacles, possibly including Steiner points at vertices of the obstacles. A *minimum anchored Steiner tree (MAST)* of P minimizes the sum of edge lengths in T. We are interested in the k-MAST (respectively, k-AST) problem that seeks to find a MAST (respectively, AST) where at most k obstacles are removed from S. If obstacles in S are not required to be pairwise disjoint, then k-AST is known to be NP-hard. We prove that finding a k-AST is NP-hard when obstacles may touch at their boundaries, but have disjoint interiors. Here we restrict the k-AST from passing through two touching obstacles, as otherwise the k-AST problem always admits an affirmative answer. We also show that finding a k-MAST is NP-hard when obstacles are disjoint and non-touching, and give a polynomial-time algorithm to solve k-MAST exactly when obstacles and sites lie on a 1.5D terrain.

1 Introduction

A *Steiner tree* of a set P of points (equivalently, *sites* or *terminals*) in the plane is a tree T drawn in the plane with line segment edges that connects all sites in P. We refer to vertices of T that are points of P as *sites*, and to vertices of T that are not in P as *Steiner points*. A *minimum Steiner tree* minimizes the sum of the edge lengths of T. In the presence of a set S obstacles, constructing such a tree may require the removal of some obstacles, e.g., when a site is separated from others by a set of obstacles. The problem of finding a minimum Steiner tree and many of its variants are well studied in computational geometry (e.g., [5, 14, 19, 21]), including showing that the problem is NP-hard in the plane [12] and that a PTAS exists [3]. These problems have applications in path planning, VLSI design, and network routing [15,18]. Given a set P of points in the plane and a set S of polygonal obstacles in the plane, an *anchored Steiner tree* is a tree drawn in the plane whose vertices correspond to all sites in P and possibly some vertices of obstacles in S, whose edges are line segments, and whose edges do not pass through the interior of any obstacle in S. Figures 1(a)–(c) illustrate a Steiner tree, a Steiner tree among obstacles, and an anchored Steiner tree, respectively. The Steiner points of an anchored Steiner tree are vertices of obstacles in S, and are called *anchor points.*

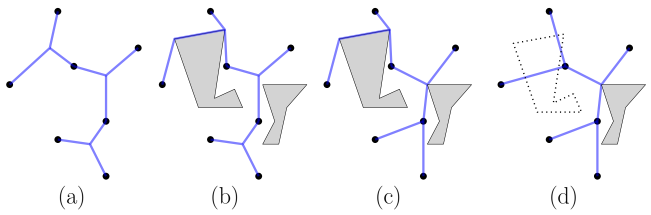


Figure 1: (a) A set of sites P and a Steiner tree for P. (b) A Steiner tree for P among two obstacles. (c) An anchored Steiner tree. (d) An anchored Steiner tree with one obstacle removed.

In this paper we examine anchored Steiner trees with removable obstacles, where two obstacle boundaries may touch, but must be interior-disjoint. Given a set P of sites, a set S of obstacles, and an integer k, we consider two variants. In *k-AST*, the goal is to decide whether an AST exists after removing at most k obstacles from S. In *k-MAST*, the goal is to find a minimum anchored Steiner tree by removing at most k obstacles from S. If the obstacles are disjoint and non-touching, then k-AST always admits an affirmative answer. Throughout the paper, while discussing k-AST, we assume the obstacles in S are allowed to touch. For k-MAST, we assume that the obstacles in S are disjoint and non-touching.

A rich body of research that examines the problem of finding a path from a given point s to a given point t while avoiding a given set of obstacles in the plane relates to our problem when $|P| = 2$ [1,4,6]. Given a set S of disjoint (non-touching) convex obstacles, a shortest s–t path that removes at most k obstacles (i.e., k-MAST with $|P| = 2$) can be computed in polynomial time [13]. However, if the obstacles have associated removal costs, then the problem becomes weakly NP-hard [1]. Agarwal, Kumar, Sintos, and Suri [1] showed how to obtain a

[*]This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) through Discovery and Alliance grants.

[†]University of Manitoba, Winnipeg, Canada, stephane.durocher@umanitoba.ca

[‡]University of Saskatchewan, Saskatoon, Canada, {arman.heydari,mark.keil,d.mondal}@usask.ca

fully-polynomial-time approximation scheme (FPTAS) for the case of convex polygons; i.e., one can compute a $(1+\epsilon)$-approximate shortest path in polynomial time with removal cost at most $(1+\epsilon)C$, where $\epsilon \in (0,1)$ and $C > 0$ is a cost budget. However, these results do not generalize to the case when $|P| \geq 2$ and the obstacles can be non-convex.

If the obstacle boundaries are permitted to cross, then finding an s–t path by removing k obstacles is known to be NP-hard [2], and remains hard even for obstacles that are unit line segments [22]. This has also been studied under barrier resilience problems, where for unit-disk obstacles, Chan and Kirkpatrick [7, 8] gave an $O(1)$-approximation algorithm and asked whether an $O(1)$-approximation can be computed for general disks. An $O(\sqrt{n})$-approximation can be obtained for this and for a more general constraint-removal problem [4], where the goal is to compute an s–t path by removing the minimum number of obstacles that are represented as simple closed curves in the plane.

Kumar, Lokshtanov, Saurahb, and Suri [16] gave the first $O(1)$-approximation algorithm for the problem of computing an s–t path with minimum obstacle removal (equivalently, for the minimum constraint removal problem [4]). They reduced the problem into a minimum color path problem in a color-connected planar graph and gave a polynomial-time $O(1)$-approximation. They also generalized this to minimum-color Steiner forests, where given a set of pairs of vertices, the goal is to minimize the total number of colors used in all paths of the forest. By setting all pairs of vertices as the input, this gives an $O(1)$-approximation for the minimum color Steiner tree problem. Consequently, this yields an $O(1)$-approximation algorithm for the AST problem. Recently, Kumar, Lokshtanov, Saurabh, Suri, and Xue [17] gave an alternate proof for the constant-factor approximation to find an s–t path with minimum obstacle removal. They also gave a $(2.3146^k n^{O(1)})$-time exact algorithm to determine whether removing k obstacles suffices to give an s–t path, improving over a previous $(k^{O(k^3)} n^{O(1)})$-time algorithm [11].

In this paper we examine the k-AST and k-MAST problems. Considering minimum anchored trees (as opposed to minimum Steiner trees) discretizes the solution space, and allows us to approximate the minimum Steiner tree within a constant factor [21]. If the obstacles are not required to be pairwise disjoint, then finding an s–t path by removing at most k obstacles (and thus a k-AST) is known to be NP-hard [16]. Hence it is natural to examine cases when obstacles are possibly touching, but interior-disjoint, and when they are disjoint and non-touching.

Contributions. We first show that if obstacles are interior-disjoint but can touch, then an s–t path with at most k obstacles removed can be computed in polynomial time (i.e., k-AST is solvable when $|P| = 2$). This follows from the observation that an s–t path connects a set of faces in the final configuration, and thus finding an s–t path can be reduced to a shortest path problem in a node-weighted graph (Section 2.1, Theorem 2). We show that the general k-AST problem is NP-hard by a reduction from monotone planar 3-SAT (Section 2.2, Theorem 3).

Next we examine disjoint and non-touching obstacles. Since a 0-AST always exists in this setting, we examine the optimization problem that minimizes the total edge length, i.e., finding a k-MAST. We show that k-MAST is NP-hard, as computing a 0-MAST within any given degree of approximation would imply a fully polynomial-time approximation scheme (FPTAS) for the Steiner tree problem (Section 3, Theorem 5).

Finally, we show that k-MAST can be solved exactly in polynomial time when obstacles and sites lie on a 1.5D terrain (Section 4). This contrasts our hardness results for k-AST and k-MAST when obstacles and sites can be intersected by a line (i.e., Theorem 4 and Theorem 6). Here we use a dynamic program that evaluates a triangulation of the solution tree and grows the tree by trying edges as we decompose the problem into subproblems. The algorithm maintains a bound k on the number of obstacles to be removed and distributes this bound to the subproblems.

2 Finding k-AST When Obstacles Can Touch But Are Interior-Disjoint

In this section we consider the k-AST problem with obstacles that are interior-disjoint but may touch. Here we restrict the k-AST from passing through two touching obstacles, as otherwise the k-AST problem always admits an affirmative answer.

2.1 Finding k-AST in Polynomial Time for $|P|=2$

We now give a polynomial-time algorithm for finding a k-AST when the input consists of two sites s, t, and a set S of interior-disjoint (but possibly touching) obstacles. The idea is to construct a graph G so that a shortest s–t path in G corresponds to the solution. Let R be the subdivision of the plane obtained by the obstacles in S (Figure 2(a)). We refer to each connected component in the subdivision as a face. The vertices of G correspond to the obstacles in S and the faces in R. An obstacle vertex v is adjacent to a face vertex w if and only if the face is adjacent to the obstacle. An obstacle vertex v is adjacent to an obstacle vertex w if and only if they touch. Figure 2(b) shows such a graph G.

We assign a weight of 1 to each obstacle vertex and a weight of 0 to each face vertex. Let f_s and f_t be the face vertices that contain s, t respectively. We now claim that a shortest path from f_s to f_t in G determines the

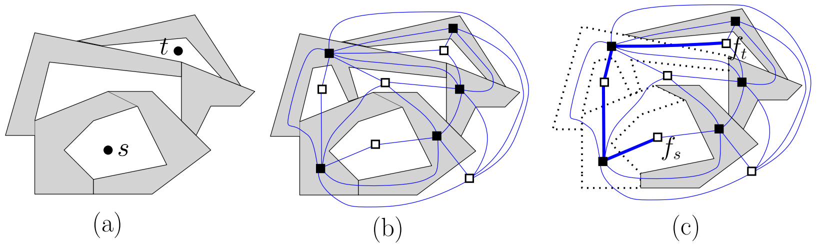


Figure 2: Illustration for (a) R and (b) G, where the obstacle and face vertices are shown as black and white squares, respectively. (c) A path P in G (shown in thick blue) that corresponds to the removal of two obstacles (shown in dotted lines).

minimum number of obstacles that must be removed to connect s and t using an anchored path.

Lemma 1 *If there exists an anchored path from s to t that removes r obstacles, then there exists a path from f_s to f_t in G of weight r and vice versa.*

Proof. Consider first the case when there exists an anchored path P from s to t that removes r obstacles from S. P intersects a sequence of obstacles and faces. The corresponding sequence of vertices in G gives us a path P' in G. If P' contains one obstacle twice, then we can shorten P' to get a path with lower cost. By repeating the process we obtain a path in G that does not contain more than r obstacle vertices.

Consider now the case when G contains a path P from f_s to f_t with r obstacle vertices (Figure 2(c)). We construct a corresponding path from s to t using a polygonal path going through the sequence of obstacles and faces determined by P. We then move the vertices of the path to obstacle vertices to obtain an anchored path (possibly adding more vertices to the path as its edges hit new obstacles). Since this transformation does not properly intersect any new obstacles, we obtain the required anchored path. □

One can find a shortest path by leveraging Dijkstra's algorithm in a vertex weighted graph. Hence we obtain the following theorem.

Theorem 2 *A k-AST with $|P| = 2$ can be computed in polynomial time.*

2.2 Hardness Results

We now show that finding a k-AST is NP-hard. Our reduction is from the NP-complete problem monotone planar 3-SAT [9]. A *monotone planar 3-SAT* consists of a set of clauses where each clause contains at most three positive (i.e., a positive clause) or at most three negated variables (i.e., a negative clause). Furthermore, the graph obtained by considering the clauses and variables as vertices and the variable-clause incidences as edges admits a planar drawing such that the following hold. (a) The variables and clauses are represented with distinct rectangles and the edges are drawn as vertical lines. (b) The variable rectangles lie on a horizontal line ℓ. (c) All the positive clause rectangles lie above ℓ and all negative clause rectangles lie below ℓ.

Given an instance of monotone planar 3-SAT I with variable set X and clause set C, we create a corresponding instance P_I of the k-AST problem, where $k = |X|$ and all obstacles are rectangles. Figure 3(a)–(b) illustrates such a transformation of I into an instance of k-AST. We now describe the details.

For each variable x_i, where $1 \le i \le |X|$, we place a *variable point* p_i inside the variable rectangle R_i. We enclose p_i with some thin rectangular obstacles as follows. The left and right sides of R_i are replaced with blocks of $k(= |X|)$ adjacent obstacles, respectively. We refer to them as *k-blocks*. The top and bottom sides of R_i are replaced with a single rectangle each. Figure 3(b) shows these obstacles in gray. For each clause c_i, where $1 \le j \le |C|$, we place a *clause point* p'_j inside the clause rectangle R'_j. We enclose p'_j with some thin rectangular obstacles leaving a narrow opening for each edge incident to R'_j. Finally, we replace each variable-clause incidence edge with two narrow rectangles which creates a narrow channel from the variable gadget to the clause gadget. Figure 3(b) shows the obstacles in orange. To construct the gadgets in polynomial time, start with integer coordinates for all elements of I, then scale I horizontally by a factor of $4|X|$ and vertically by a factor of 4. This gives sufficient area to place the rectangular obstacles with integer grid points.

We obtain the instance $\mathcal{P}$ of k-AST by setting $k = |X|$, taking the variable and clause points as the sites (i.e., set P), and taking the rectangles that we used to construct the gadgets as the set S of obstacles.

Theorem 3 *The k-AST problem is NP-hard.*

Proof. We reduce from monotone planar 3-SAT. Let I be an instance of monotone planar 3-SAT and let $\mathcal{P}$ be the corresponding instance of k-AST. Without loss of generality assume that $|X| \ge 2$. Consider first the case when I admits an affirmative solution. If a variable is

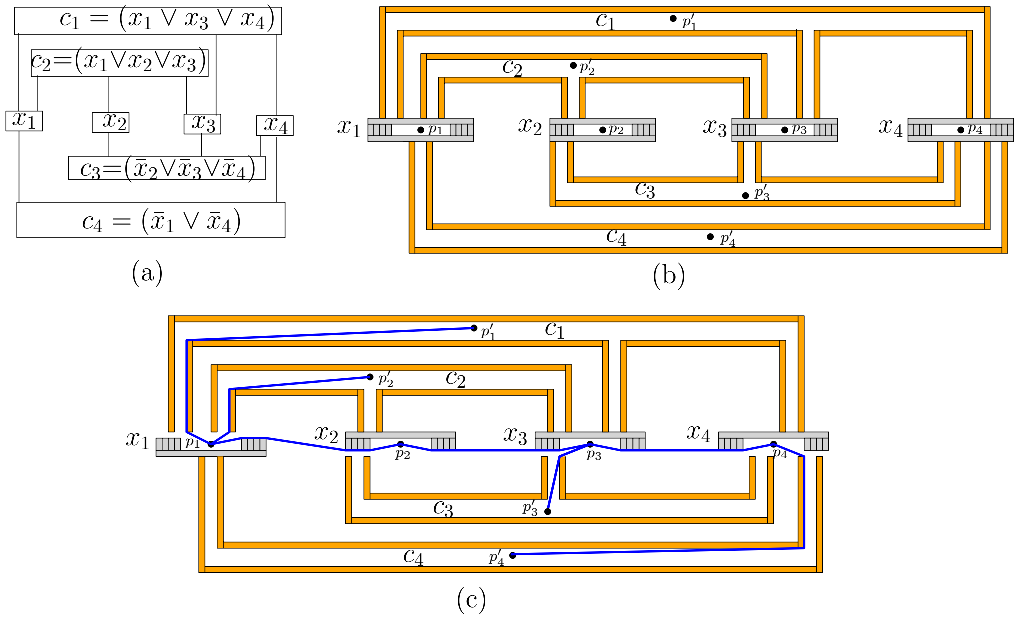

$c_1 = (x_1 \vee x_3 \vee x_4)$
$c_2=(x_1 \vee x_2 \vee x_3)$
x_1
x_2
x_3
x_4
$c_3=(\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4)$
$c_4 = (\bar{x}_1 \vee \bar{x}_4)$
(a)
c_1
p'_1
c_2
p'_2
p_1
p_2
p_3
p_4
c_3
p'_3
c_4
p'_4
(b)
(c)

Figure 3: (a) An instance of monotone planar 3-SAT I. (b) An instance of k-AST. (c) A solution to I where x_1 is set to True and x_2, x_3, x_4 are set to False.

set to True (respectively, False), then we remove the top (respectively, bottom) obstacle of its gadget. We then connect the variable points with a monotone anchored path, which must exist by the construction of the gadgets. Since each clause c is satisfied, it has an incident variable gadget whose variable point is reachable to c via an anchored path. We take all these anchored paths and together with the first anchored path that connects all the variable points, we obtain an anchored tree that connects all the variable and clause points. Since we removed only k-obstacles, the tree determines a solution to $\mathcal{P}$. Consider now the case that $\mathcal{P}$ has an affirmative solution T. We first observe that for each variable gadget, exactly one of its top or bottom obstacles must be removed. Otherwise, we have the following cases.

Case 1. Consider a variable x where neither the top nor the bottom obstacles have been removed. To connect x to T, the only other option is to remove a k-block. However, since we are only allowed to remove at most k obstacles, the other variable points remain blocked and T cannot be a solution.

Case 2. Consider a variable x where both the top and the bottom obstacles have been removed. We can remove at most $k-2$ obstacles from the remaining $k-1$ variable gadgets. Hence there must be one variable gadget where neither the top nor the bottom obstacles have been removed. Hence we can apply Case 1.

For each variable gadget, if its top (bottom) obstacle is removed, then we set the variable to True (False). Assume that this truth assignment does not satisfy a clause c. Then the clause point is entirely enclosed by a set of obstacles, which contradicts that T is an AST. $\square$

We use the concept of monotone topological book embedding to show that k-AST problem remains NP-hard in a restricted setting as stated in the following theorem (see Appendix A).

Theorem 4 *The k-AST problem is NP-hard even when the sites and obstacles can be intersected by a line.*

3 k-MAST with Disjoint Non-Touching Obstacles

In this section we show that the k-MAST problem when obstacles are interior-disjoint and non-touching is NP-hard. We reduce the NP-hard problem of finding a minimum Steiner tree [10, 12] by using a discretization method [20] to the problem of finding a 0-MAST.

Theorem 5 *The k-MAST problem is NP-hard even when $k = 0$ and the obstacles are convex and disjoint.*

Proof. Garey, Graham, and Johnson [12] showed that given an instance P (i.e., n points) of the minimum Steiner tree problem and a positive integer Q, it is NP-hard to distinguish whether $OPT < Q$ and $OPT > Q + 2M$, where $Q > n$, $1 < M < Q$, and Q, M are polynomial in $|P| = n$. Rao and Smith [20] showed that given a set P of n points and $\varepsilon > 0$, one can construct another set P' of poly$(n, 1/\varepsilon)$ points in poly$(n, 1/\varepsilon)$ time such that the length of the minimum Steiner tree T' of P using Steiner points from P' is at most $(1+\varepsilon)$ times of the length L of the minimum Steiner tree T for P. Figure 4(a) illustrates P and P' in black dot

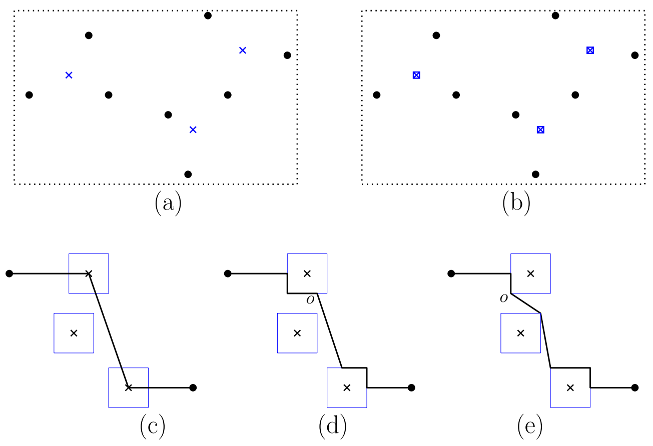


Figure 4: Illustration for the proof of Theorem 5.

and blue cross, respectively. The length of T' is at most $(1+\varepsilon)OPT$, where OPT is the length of T. If we set $\varepsilon = 1/Q$, then we have $|P'| = \text{poly}(n)$ and P' is computed in $\text{poly}(n)$ time. If $OPT < Q$, then $(1+\varepsilon)OPT < Q+1$. If $OPT > Q + 2M$, then $(1 + \varepsilon)OPT > Q + 2M$. Since $M > 1$, it is NP-hard to distinguish whether the length of T' is smaller than $Q + 1$ or larger than $Q + 2M$.

For each point $p \in P'$, we now construct an axis-aligned square $\square_p$ (centered at p) of perimeter $z = \frac{\min\{1,\beta\}}{Q|P \cup P'|^2}$, where β is the closest pair distance in $(P \cup P')$ (Figure 4(b)). We now claim that if T' has length OPT', then the 0-MAST instance that treats P as the set of sites and the boxes as obstacles has a solution of length at most $OPT' + 1/Q$. There are at most $|P \cup P'|$ edges in T' and each edge can be replaced by a path going around the boxes using an additional cost of z. We can then move the Steiner points that lies interior to an obstacle edge to one of its vertices to obtain an anchored Steiner tree. Figure 4(c)–(e) illustrates such a transformation where the Steiner point o has been moved. Since this may hit $O(P')$ other obstacles, the final cost remains within $OPT' + z|P \cup P'|^2 \leq OPT' + 1/Q$.

If $OPT' < Q + 1$, then $OPT' + 1/Q = Q + 1/Q + 1$. If $OPT > Q+2M$, then $OPT' + 1/Q = Q+1/Q+2M$. Hence it is NP-hard distinguish whether the length of a 0-MAST is smaller than $Q + 1/Q + 1$ or larger than $Q + 1/Q + 2M$. □

We modify the proof of Theorem 4 to show that the k-MAST problem remains NP-hard even in the restricted setting when the sites and obstacles can be intersected by a line (see Appendix B). The idea is to transform the shared boundaries among the touching obstacles using narrow zigzag channels so that the obstacles become disjoint but taking a path through the zigzag channel imposes a very large cost. This keeps the functionalities of the clause and variable gadgets the same, and the hardness result follows.

Theorem 6 *The k-MAST problem is NP-hard even when the sites and obstacles can be intersected by a line.*

4 Finding a k-MAST on a 1.5D Terrain

A 1.5D terrain T is an x-monotone polygonal chain in $\mathbb{R}^2$. In this section, we consider the case when sites in P are points on T and the obstacles in S are above and touching T (e.g., Figure 5(a)). We design a polynomial-time dynamic programming algorithm to solve the k-MAST problem in this setting.

The idea is to find the optimal tree along with a triangulation of the region within the tree and the obstacles that remain in the solution. Specifically, consider two dummy obstacles to the left and right sides of the terrain with a height larger than any obstacle or terrain vertices. Then the solution anchored Steiner tree, and the obstacles that are not removed can be triangulated within the region bounded by the obstacles and the terrain to obtain a straight-line plane graph, as shown in Figure 5(b). Our dynamic programming creates the subproblems by guessing the triangles of such a triangulation and ensures that a subproblem can be encoded concisely.

We now provide the details. Let $p_1, \ldots, p_n$ be the sites that appear on T in this left-to-right order. Assume that the obstacle and terrain vertices are labeled as $c_0, c_1, \ldots, c_m$ in the order one encounters them when walking along the terrain from left to right.

We refer to $M(k, a, b, \mathcal{Q}, I)$ as the problem of computing k-MAST inside the region R determined by $a, b, \mathcal{Q}$, where I is set to Φ or $\{a, b\}$. Here a and b can be sites or obstacle vertices, and $\mathcal{Q}$ contains the open segment ab and its corresponding end vertices, each with a label indicating whether it must be included in the solution or not. If I is set to $\{a, b\}$, then the Steiner tree must connect all the sites as well as a and b. If I is set to Φ, then the Steiner tree reachable from a is disjoint from the Steiner tree reachable from b.

We will write an element $q \in \mathcal{Q}$ as $\underline{q}$ if it must be included, but as $\overline{q}$ if it must be excluded. Specifically, the instance is defined with the sites and obstacles we encounter while walking from a to b from left to right. If a (similarly, b) is an obstacle vertex, then R contains the corresponding obstacle on its boundary. If $\mathcal{Q}$ contains an edge or vertex that must be included in the solution, then they also appear on the boundary of R. If the segment $q \in \mathcal{Q}$ properly intersects an obstacle O, where q is an edge on the triangulation that we guess, then we ignore the obstacle O.

Figure 5(a) illustrates two examples. Here $M(k, q, r, \{\underline{q}, \overline{r}, \overline{qr}\}, \Phi)$ denotes that R is bounded by the segment qr, obstacles O_3 and O_6, and the terrain between O_3 and O_6. Since qr intersects O_4, O_4 is excluded from the problem. The segment qr must not be included in the solution Steiner tree, but q should be included as stated in $\{\underline{q}, \overline{r}, \overline{qr}\}$. Here I is set to Φ, i.e., the sites connected to q must be disjoint from r. However, since r is excluded from the solution, all the sites must be connected to q. In the other example, the problem $M(k, p_2, q, \{\underline{q}, \underline{p_2}, \underline{qp_2}\}, \Phi)$ denotes that R is bounded by the edge qp_2, the obstacle O_3 and the terrain between p_2 to O_3. Since qp_2 intersects O_2, O_2 is excluded from the problem. Since qp_2 is already in the solution, I is set to Φ. In other words, we want the sites connected to q to be disjoint from the sites connected to p_2. Otherwise, the edge qp_2 will form a cycle in the resulting tree.

To decompose a problem $M(k, u, v, \mathcal{Q}, I)$, we first determine the region R, which is straightforward to compute in polynomial time. We next exclude the obstacles that are intersected by the segment uv. We then guess the triangle below uv by choosing a point w (which is a site or a vertex) inside R such that the segments uw and vw do not properly cross the boundary of R. This decomposes R into two subregions, one below uw and the other below vw, and we define the subproblems considering $\mathcal{Q}$. There are only a polynomial number of choices to grow the tree along vw and uw based on $\mathcal{Q}$ and I. For example, if $I = \Phi$ and $\mathcal{Q} = \{\underline{u}, \overline{v}, \overline{uv}\}$, as shown in Figure 5(b), then we must exclude O_4. If we choose w as the top corner of the obstacle O_6, then we have the following three options to define the subproblems for uw: $\{\underline{u}, \overline{w}, \overline{uw}\}$, $\{\underline{u}, \underline{w}, \underline{uw}\}$ and $\{\underline{u}, \underline{w}, \overline{uw}\}$. However, since the subproblem under vw contains sites that must be connected, we can safely exclude $\{\underline{u}, \overline{w}, \overline{uw}\}$. The options that exclude u (e.g., $\{\overline{u}, \ldots\}$) are not considered as u was already included. To define the subproblem for wv, we only have the option $\{\underline{w}, \overline{v}, \overline{uw}\}$ because v is not in the solution (based on $\mathcal{Q}$) and w must be in the solution to connect the sites.

When decomposing the problem, the parameter I needs to be set consistently. For example, the option $\{\underline{u}, \underline{w}, \overline{uw}\}$ requires I to be set to $\{u, w\}$ because w was not a part of the Steiner tree before but we are including w without taking the segment uw. We decompose this example instance $M(k, u, v, \{\underline{u}, \overline{v}, \overline{uv}\}, \Phi)$, as follows.

$$M(k, u, v, \mathcal{Q}, I) = \min_{\substack{i+j \le k-1 \\ i,j \ge 0}} \begin{cases} M(i, u, w, \{\underline{u}, \underline{w}, \underline{uw}\}, \Phi) + \\ \qquad M(j, w, v, \{\underline{w}, \overline{v}, \overline{wv}\}) + |uw| \\ M(i, u, w, \{\underline{u}, \underline{w}, \overline{uw}\}, \{u, w\}) + \\ \qquad M(j, w, v, \{\underline{w}, \overline{v}, \overline{vw}\}, \Phi) \end{cases}$$

Here we consider only $k-1$ obstacles to remove because the segment wv properly intersects the object O_7 which was not intersected by uv already. In contrast, O_4 is intersected by uv, and hence it is considered removed and we do not decrease the number of obstacles that can be removed for uw.

We solve the base cases using brute force, we consider

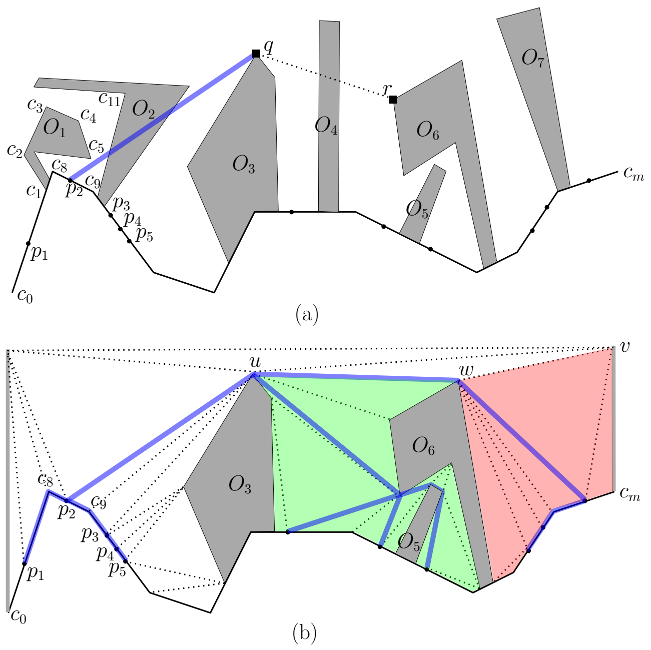


Figure 5: (a) Illustration for $M(k, a, b, \mathcal{Q}, I)$. (b) An anchored Steiner tree is shown in thick blue.

all possible instances $M(r, a, b, \mathcal{Q}, I)$, where $0 \le r \le k$ and ab is a segment on the terrain or obstacle boundary. In a general recurrence relation, the number of cases to consider is bounded by a polynomial in k and the number of choices for the vertex inside R. A subproblem either returns a connected tree spanning all the sites in it or two disjoint trees connecting all the sites that become connected when combining the solution. Therefore, the final result is a connected subgraph that spans all sites. Let a and b be the top vertices of the dummy obstacles placed at c_0 and c_m, respectively. Then the solution to the input problem is obtained by invoking $M(i, a, c, \{\overline{a}, \underline{c}, \overline{ac}\}, \Phi) + M(j, c, b, \{\underline{c}, \overline{b}, \overline{cb}\}, \Phi)$ for each $i + j \le k$, where c is a point inside the problem region, and taking the minimum over all options. We thus obtain the following theorem.

Theorem 7 *Given a set of sites P and a set of obstacles on a 1.5D terrain, a k-MAST for P can be computed in polynomial time.*

5 Discussion

This paper examines the k-AST and k-MAST problems, providing hardness results and polynomial-time algorithms in restricted settings. A natural open problem is to design a polynomial-time algorithm that, given obstacles in S, produces a k-MAST of P whose total length is within a constant factor of the minimum possible. Or, prove the hardness of approximating such an optimal obstacle removal.

References

[1] P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri. Computing shortest paths in the plane with removable obstacles. In *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018, June 18-20, 2018, Malmö, Sweden*, pages 5:1–5:15, 2018.

[2] H. Alt, S. Cabello, P. Giannopoulos, and C. Knauer. On some connection problems in straight-line segment arrangements. In *Proc. of the 24th European Workshop on Computational Geometry (EuroCG)*, 2008.

[3] S. Arora. Nearly linear time approximation schemes for euclidean TSP and other geometric problems. In *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 554–563. IEEE Computer Society, 1997.

[4] S. Bandyapadhyay, N. Kumar, S. Suri, and K. R. Varadarajan. Improved approximation bounds for the minimum constraint removal problem. *Comput. Geom.*, 2020.

[5] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. An improved LP-based approximation for Steiner tree. In *Proc. Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 583–592, 2010.

[6] S. Cabello, K. Jain, A. Lubiw, and D. Mondal. Minimum shared-power edge cut. *Networks*, 75(3):321–333, 2020.

[7] D. Y. C. Chan and D. Kirkpatrick. Approximating barrier resilience for arrangements of non-identical disk sensors. In *Proc. of the International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 42–53. Springer, 2012.

[8] D. Y. C. Chan and D. Kirkpatrick. Multi-path algorithms for minimum-colour path problems with applications to approximating barrier resilience. *Theoretical Computer Science*, 553:74–90, 2014.

[9] M. de Berg and A. Khosravi. Optimal binary space partitions in the plane. In *Proc. Computing and Combinatorics*, pages 216–225, 2010.

[10] A. Dhar, S. Hait, and S. Kolay. Efficient algorithms for Euclidean Steiner minimal tree on near-convex terminal sets. In *Proc. of the 34th International Symposium on Algorithms and Computation (ISAAC)*, volume 283 of *LIPIcs*, pages 25:1–25:17, 2023.

[11] E. Eiben and D. Lokshtanov. Removing connected obstacles in the plane is FPT. In *Proc. of the 36th International Symposium on Computational Geometry (SoCG)*, volume 164 of *LIPIcs*, pages 39:1–39:14, 2020.

[12] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP complete. *SIAM Journal of Applied Mathematics*, pages 826–834, 1977.

[13] J. Hershberger, N. Kumar, and S. Suri. Shortest paths in the plane with obstacle violations. *Algorithmica*, 82(7):1813–1832, 2020.

[14] F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

[15] E. Ihler, G. Reich, and P. Widmayer. Class Steiner trees and VLSI-design. *Discrete Applied Mathematics*, 90(1-3):173–194, 1999.

[16] N. Kumar, D. Lokshtanov, S. Saurabh, and S. Suri. A constant factor approximation for navigating through connected obstacles in the plane. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 822–839, 2021.

[17] N. Kumar, D. Lokshtanov, S. Saurabh, S. Suri, and J. Xue. Point separation and obstacle removal by finding and hitting odd cycles. In *Proc. 38th International Symposium on Computational Geometry (SoCG)*, pages 1–14, 2022.

[18] M. Leitner, I. Ljubić, M. Luipersbeck, M. Prossegger, and M. Resch. New real-world instances for the Steiner tree problem in graphs. Technical report, ISOR, University of Vienna, 2014.

[19] H. Prömel and A. Steger. *The Steiner Tree Problem: A Tour through Graphs, Algorithms, and Complexity*. Advanced Lectures in Mathematics. Vieweg+Teubner Verlag, 2002.

[20] S. B. Rao and W. D. Smith. Improved approximation schemes for geometrical graphs via spanners and banyans. *Proc. of the 13th ACM Symposium on the Theory of Computing (SoCG)*, pages 540 – 550, 1998.

[21] M. Razaghpour and A. Lubiw. The Steiner ratio for obstacle-avoiding rectilinear Steiner trees. In *Proc. of the 20th Annual Canadian Conference on Computational Geometry (CCCG)*, 2008.

[22] K. R. Tseng and D. G. Kirkpatrick. On barrier resilience of sensor networks. In *Proc. Algorithms for Sensor Systems - 7th International Symposium on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities*, pages 130–144, 2011.

Appendix

A Finding k-AST with Sites and Obstacles Intersected by a Line

We now show that k-AST problem remains NP-hard even in the restricted setting when the sites and obstacles can be intersected by a line. To this end we use a monotone topological book embedding of the graph corresponding to the SAT instance. A *monotone topological book embedding* of G is a planar drawing Γ of G with the following properties. (a) The vertices of G lie along a horizontal line ℓ. (b) Each edge (u, v) of G is drawn as a x-monotone polyline, where (u, v) either lies on one side of ℓ, or crosses ℓ at most once. (c) Let (u, v) be an edge that crosses ℓ at point d, where u appears before v on ℓ. Let $u, \ldots, d, \ldots, v$ be the corresponding polyline. Then the polyline $u, \ldots, d$ lies above ℓ, and the polyline $d, \ldots, v$ lies below ℓ. Figure 6(a) illustrates a monotone topological book embedding corresponding to the SAT instance of Figure 3(a).

It is straightforward to construct an orthogonal drawing similar to Γ, where edges are drawn using axis-aligned x-monotone polygon. If G has n vertices then the area of Γ is

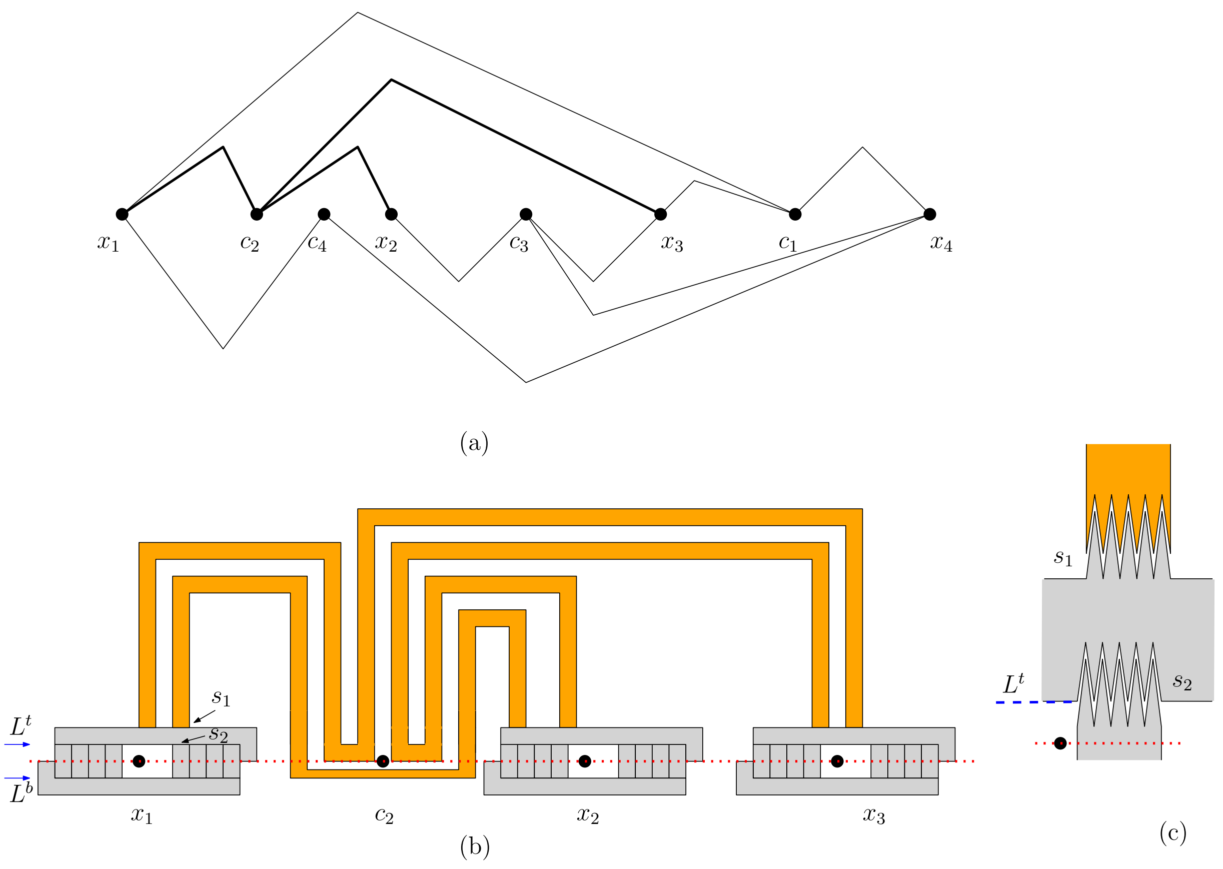


Figure 6: (a) A monotone topological book embedding. (b) Illustration for a clause gadget. The line L that intersects the sites and obstacles is shown in a red dotted line. (c) Making the obstacles disjoint, where the horizontal adjacencies s_1 and s_2 are replaced with narrow zigzag channels.

$O(n) \times O(n)$ and the length of the edges are also bounded by $O(n)$. We slightly modify the drawing at each vertex such that the connections from the positive and negative clauses comes from above and below, respectively. Let L be the line through the clause and variable points. We now build the vertex gadgets as in Section 2.2, but add an extension to the top and bottom rectangles to their right and left respectively, so that they touch the line L. We then use two obstacles to build the clause gadget, where we create a narrow channel along the edge connecting the clause to its variables, as shown in Figure 6(b). The functionalities of these gadgets remains the same, i.e., a block of k vertices must be removed to reach a variable point, whereas each clause point must remove a block of the variable gadget to reach a variable point. Therefore, we can use the same proof as in Theorem 3 to obtain the hardness in this restricted setting.

B k-MAST with Sites and Obstacles Intersected by a Line

In Section A we showed that the k-AST problem is NP-hard even when the sites and obstacles can be intersected by a straight line. Here we show how to modify the gadgets to obtain the hardness result for k-MAST in this setting where the obstacles are disjoint.

Let I be an instance of monotone planar 3-SAT with n variables and m clauses and let $\mathcal{P}$ be the corresponding instance of k-AST (Figure 6(a)–(b)). Let w and h be the width and height of the axis-aligned bounding box containing the obstacles of $\mathcal{P}$. Let L^t be the horizontal line passing through the top side of the k-block. Similarly, let L^b be the horizontal line passing through the bottom side of the k-block. We ensure that the clause gadget for the positive and negative clauses stay above L^b and below L^t, respectively. If a pair of obstacles touch in the k-AST instance, then they satisfy the following properties.

(I) If both the obstacles belong to a k-block, then they share a vertical segment.

(II) If at most one obstacle belongs to a k-block then the other belongs to a clause gadget, and they share a horizontal segment.

We first scale the width of $\mathcal{P}$ by a factor ζ, where $\zeta = \max\{w, h\}^3$. We then vertically scale the area above L^t and below L^b by a factor of ζ. Let the resulting scaled instance be $\mathcal{P}_s$. We shift the obstacles in the k-block horizontally towards the variable point creating small gaps in between such that the no vertical adjacencies (described in (I)) remains. Since each horizontal adjacencies (described in (II)) are of length at least ζ, we can create a narrow zigzag channel of length $\zeta \times \zeta$ by creating sharp sawtooth shapes on the corresponding polygons as illustrated in Figure 6(c) Let the resulting instance $\mathcal{P}''$ be the instance of k-MAST where $k = |C|$. We now show that $\mathcal{P}$ has an affirmative solution that removes at most k obstacles if and only if $\mathcal{P}''$ has an anchored Steiner tree of length at most $O(\zeta \max\{h, w\}^2)$ that removes at most k obstacles.

Assume that an affirmative solution exists for $\mathcal{P}$. By the property of a monotone topological book embedding, the shortest path length between a clause point to any of its variable gadget is $O(w + h)$, and the shortest monotone path $\mathcal{Q}$ that connects all the variable points (when for each variable gadget, either its top or bottom obstacle is removed) is of length $O(w)$. In the scaled version $\mathcal{P}_s$, these lengths become $O(w\zeta + h\zeta)$ and $O(w\zeta + |X|\zeta)$, respectively. Here the term $O(|X|\zeta)$ corresponds to the scenarios when the path $\mathcal{Q}$ needs to traverse $O(\zeta)$ vertical length to go over the k-blocks between two variable points. The corresponding anchored Steiner tree in $\mathcal{P}''$ thus have a length of $O(|C|\zeta(w + h) + \zeta(w + |X|)) = O(\zeta(w^2 + hw) + \zeta w) = O(\zeta \max\{h, w\}^2)$ since $|C|, |X| \leq w$.

If there is an anchored Steiner tree of length at most $O(\zeta \max\{h, w\}^2)$ for $\mathcal{P}'$, then the tree did not take any zigzag channel because the length of a zigzag channel is $\zeta^2 = O(\max\{w, h\}^6)$, whereas $O(\zeta \max\{w, h\}^2)$ is at most $O(\max\{w, h\}^5)$. Hence we can remove the corresponding obstacles in $\mathcal{P}$ to construct an affirmative solution for the k-AST problem.

Approximate and Exact Geometric Generalized Minimum Spanning Trees

Majid Mirzanezhad[*] Arash Rafiey [†]

Abstract

We revisit the geometric variant of the generalized minimum spanning tree (GGMST) problem: we are given a set of n points in $\mathbb{R}^d$ inside an integer grid with N non-empty clusters where each cluster is a unit hypercube, the aim is to find a minimum spanning tree that contains exactly one point from each cluster while respecting the adjacencies of the clusters. For any $\epsilon > 0$, we give an $(1+\epsilon)$ approximation algorithm whose running time is $O^*\big(N \cdot (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})} + N^{d+1} + n\log n\big)$, where O^* hides factors polynomial in $1/\epsilon$. We also give an exact FPT algorithm that runs in $O\big(N^2\rho^2\big(2^{O(\kappa^4)}\big)\big)$, where κ is an implicit parameter that relates to the input integer grid containing the non-empty clusters. Here, ρ is the maximum number of points across all clusters. When a GGMST instance is given in the plane, we present $(1+\epsilon)$-approximation with running time $O\big(N\rho^2 280^{(\frac{1}{\epsilon})}\big)$. In the case that for any non-empty cluster, either the row or column it belongs to, consists of at most $\gamma < N$ many non-empty clusters, we give a faster FPT algorithm that runs in $O\big(N\rho^2 560^{\gamma}\big)$ time.

1 Introduction

In *Generalized Minimum Spanning Tree* (GMST) problem, we are given an undirected abstract graph $G = (V, E)$ with vertex set V and edge set $E \subseteq V \times V$ together with a weight function $w : E \rightarrow \mathbb{R}^+$ and a partition of vertices V into non-intersecting clusters. The goal is to find a minimum weight tree containing exactly one vertex from each cluster. This problem is a generalization of the well-known *Minimum Spanning Tree* (MST) Problem where each vertex of the graph is a cluster itself. In the geometric variant of the problem, we are given N non-empty d-dimensional axis-aligned unit hypercubes, which we refer to as *clusters*, and a set of n points in $\mathbb{R}^d$ spread across the N clusters. Each cluster contains at least two points. The aim is to pick exactly one point from each cluster and connect the chosen N points by straight-line segments (with Euclidean weights) to form an MST. We only allow edges between points chosen from *adjacent* clusters, where adjacency means the clusters share at least a corner. This problem is also known as *Geometric Generalized Minimum Spanning Tree* (GGMST).

The GGMST appears in telecommunications network planning, where a network of node clusters needs to be connected via a tree architecture using exactly one node per cluster [7] and has applications in energy distribution and agricultural irrigation [9]. The GMST problem was first introduced by Myung et al. (1995) [13] and known to be NP-hard [6]. While there is no constant factor approximation unless P = NP, some heuristic algorithms have been developed [7, 9]. Pop et al. [15] lay out a $2\rho-$approximation algorithm for GGMST where the size of each cluster is bounded by ρ. They take advantage of Integer programming and LP relaxation to tackle the problem. Feremans, Grigoriev, and Sitters consider GGMST [5] and show a strong NP-hardness, even if we restrict to instances in which all non-empty clusters are connected and each cluster contains at most two points (Theorem 1 in [5]). They design a dynamic programming algorithm that solves the problem with connected non-empty clusters in time $O(l\rho^{6k}2^{34k^2}k^2)$ where the grid has size $l \times k$ and it is polynomial when k is fixed (Theorem 2 in [5]).

Bhattacharya et al. designed a polynomial-time approximation algorithm with a guaranteed ratio of $1 + 4\sqrt{2} + \epsilon$ [4]. Later, Kachooei et al. [10] showed that the problem is still NP-hard when there are at most two points in each cluster even with equal y-coordinates, and it is unlikely to have a fully polynomial time approximation scheme (FPTAS) unless P = NP. It is also known that the GGMST is NP-hard even if the cluster graph forms a tree [14].

Our results. We obtained two primary results in higher dimensions and then we show how they can be improved when the instance is given in the plane. We propose two approximation and FPT algorithms for each case as follows:

(1) We give a $(1+\epsilon)$-approximation algorithm that runs in $O\big(n\log n + n/\epsilon^d + (N/\epsilon)^{d+1} + N(\frac{1}{\epsilon})^{O(1/\epsilon)}\big)$, in any constant dimension d (Thm. 5). Our approach utilizes Arora's PTAS technique for TSP in [2]. Arora's technique requires portal *pairings* (entering and exiting a quadtree cell in matching pairs) to form a tour. Our GGMST variant tracks *connected* subsets of portals (partitions) to encode how GGMST edges unite bound-

[*]School of Electrical Engineering and Computer Science, Ohio University, USA, miirza@ohio.edu

[†]Department of Computer Science, Indiana State University, IN, USA, arash.rafiey@indstate.edu

ary portals to form connected components.

(2) We also give an exact FPT algorithm that runs in $O\big(\rho^2 N^2 \big(2^{O(k^4)}\big)\big)$, where κ is the treewidth of the hypergraph induced by the non-empty clusters of the integer grid in $\mathbb{R}^d$, for any constant dimnesion d (Thm. 7). In telecom networks, especially large-scale ones, each local subnetwork (e.g., LAN, building, or regional cluster) designates exactly one node (its gateway or hub) to connect to external links. This is precisely the setting where each cluster has to select one hub.

(3) We show that the optimal GGMST in $\mathbb{R}^2$ can be computed in time $O(N\rho^2 280^k)$ where the instance $(\mathcal{G}, \mathcal{N}, P)$ lies in a grid of size $k \times l$ and ρ is the maximum number of points inside each cluster (Thm. 8). As a byproduct of this result, we can obtain the following PTAS. Using a similar approximation argument that appeared in [5], one can obtain a $(1+\epsilon)$-approximation for instance $(\mathcal{G}, \mathcal{N}, P)$ in time $O(N\rho^2 280^{\frac{1}{\epsilon}})$ (Corollary 9) which significantly improves the $(1+\epsilon)$-approximation algorithm with running time $O\big(\frac{l}{\epsilon^2}\rho^{\frac{6}{\epsilon}} 2^{(\frac{34}{\epsilon^2})}\big)$ in [5].

(4) In the case that for any non-empty cluster, either the row or column it belongs to, consists of at most $\gamma < N$ many non-empty clusters, we give another FPT algorithm in $\mathbb{R}^2$ that runs in $O\big(N\rho^2 560^\gamma\big)$ which is much faster than our FPT algorithm in $\mathbb{R}^d$ (Thm. 10).

2 Preliminaries

Given two points $x, y \in \mathbb{R}^d$, the Euclidean distance between them is denoted by $\|x-y\|$. Computing minimum spanning trees requires setting a fixed point as the root of a tree. We denote a subtree rooted at a point x by T_x, consisting of all nodes and edges descending from x, encapsulating the hierarchical structure induced by the root. The tree's weight $W(T)$ represents the total edge lengths of T. When dealing with substructures, we use $W(T_x)$ to specify the weight of the subtree rooted at x.

A *"Hypergraph"* of an integer grid is a graph whose vertices are the non-empty clusters in the grid and two vertices share an edge iff the clusters are adjacent. Let $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ be an instance of the GGMST where $\mathcal{G}$ is an HG, $\mathcal{N}$ is the set of clusters with $|\mathcal{N}| = N$, and P is the set of points spread across the clusters in $\mathcal{N}$, with $|P| = n$. An HG $\mathcal{G}$ in $\mathcal{I}$ can attain a spanning tree whose nodes span all the clusters in $\mathcal{I}$. The said spanning tree is defined as follows: A *"Backbone Spanning Tree* (BBST)*"* w.r.t. some integer grid clusters is a spanning tree if (1) each vertex corresponds to a cluster, (2) all vertices span the entire non-empty clusters, and (3) two vertices share an edge if their corresponding clusters are adjacent to each other.

Let $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ be an instance of the GGMST problem. For a given BBST T, let $\mathsf{OptMST}(T)$ denote the optimal tree corresponding to the BBST T. Note that every backbone spanning tree can induce a GGMST whose adjacency of nodes follows the adjacency of nodes in the BBST. Let $\mathcal{B}$ be the set of all possible BBSTs. Then: $T_{opt} := \min_{T \in \mathcal{B}} \mathsf{OptMST}(T)$. In other words, a branching tree of the BBST T describes the adjacency of the vertices of a GGMST T^* where T^* selects the points of every cluster marked by T and connects them with respect to the adjacency of the clusters in T. While most of the notions are utilized in the main presentation, we included the omitted proofs in Appendix for the sake of brevity.

3 Approximation algorithm for GGMST

As mentioned earlier, the GGMST problem is NP-hard in general, but some underlying geometric structural properties can be utilized to achieve approximation schemes. Our goal is to provide a PTAS, albeit potentially with a high dependence on $1/\epsilon$.

3.1 Algorithmic sketch

We begin with discretizing each cluster into a finite set of candidate points independent of n. By choosing a sufficiently fine grid inside each cluster, we can guarantee that picking the nearest candidate point to the optimal solution's chosen point will not increase the weight (W) of optimal GGMST by more than $\epsilon \cdot W(T_{\text{opt}})$. Next, we construct a *WSPD-based* $(1+\epsilon)$*-spanner* to reduce the potential $O(n^2)$ edge set down to $O(n)$ edges without significantly increasing GGMST cost.

We then build a *quadtree* on the clusters, placing $O(1/\epsilon)$ *portals* on each cell boundary to constrain how GGMST edges may cross those boundaries. Applying a bottom-up *dynamic programming* (DP) merges partial GGMST solutions from child cells by matching their portal-connectivity patterns, similar to Arora's technique for TSP that appeared in [2]. By carefully bounding the "overhead", i.e., inflated weight added to the solution due to the approximation via the portal placement introduced at each scale (from forcing edges to go through portals) and leveraging the spanner's limited set of edges, the algorithm achieves a total GGMST cost within $(1+\epsilon)$ of the true optimum, all in polynomial time for a fixed ϵ. Let $\delta > 0$, we partition each cluster uniformly into grid cells each of side length δ. This yields $O(\frac{1}{\delta^d})$ candidate points per cluster. Let P be the resulting set of all cluster grid points. The naïve algorithm below is a dynamic programming algorithm that can compute the exact GGMST given a BBST T.

Lemma 1 *For a given* BBST T*, one can compute the* $\mathsf{OptMST}(T)$ *in polynomial time.*

Lemma 2 (Discretization) *Let* $\epsilon > 0$*. We can choose* $\delta = \epsilon/(\sqrt{d}(N-1))$ *sufficiently small so that* *there exists a $(1+\epsilon)$-approximate GGMST solution selecting from these candidate points. Moreover, the number of grid points overall is $O(N^d/\epsilon^d)$*

3.2 The dynamic program

We present the algorithm and analysis in four main steps: (1) quadtree construction and cluster isolation, (2) building a $(1+\epsilon)$-Spanner using WSPD, (3) portal placement and the DP scheme, and finally, (4) analysis of approximation guarantee and running time.

The first two steps are relatively straightforward to handle as follows: our first goal is to spatially organize the clusters and points so that we can apply a dynamic program to propagate the minimum weight of a spanning forest from each "zone" to the adjacent ones. We use a *quadtree* to recursively partition the plane until each leaf cell of the quadtree contains at most one cluster (thus fully containing that cluster). By ensuring each cluster resides in precisely one leaf cell, picking a representative grid point from that cluster is a local decision at the leaf. We utilize further to ensure a constant number of points per leaf limits state explosion in the DP state. Without further techniques, considering an MST among n points might require looking at $O(n^2)$ potential edges. To make the problem tractable, we reduce complexity by constructing a $(1+\epsilon)$-spanner. Given n points in the plane and $\epsilon > 0$, it is known that one can construct a $(1+\epsilon)$-spanner G with $O(n/\epsilon^d)$ edges in $O(n\log n + n/\epsilon^d)$ time [8]. Furthermore: $\mathrm{MST}(G) \leq (1+\epsilon)\mathrm{MST}(P)$, where P is the set of points and $\mathrm{MST}(P)$ is the optimal MST cost on these n points.

Portal placement and the DP scheme. Even with a spanner, we face the challenge of ensuring we pick exactly one point per cluster to form a GGMST. We approach this as follows: In the Arora-style PTAS algorithm, portals are used to discretize how solutions cross cell boundaries. Place $p = O(1/\epsilon)$ portals uniformly along each side of every quadtree cell. Any MST edge crossing the boundary must "snap" to the nearest portal. By introducing portals: i) we limit the complexity of boundary interactions in the DP states and ii) we ensure that any additional cost from snapping edges to portals is at most $\epsilon \cdot W(T_{\mathrm{opt}})$ after appropriate scaling.

Lemma 3 *Let T_{opt} be an GGMST of cost $W(T_{opt})$ in a bounding box of side-length M. Suppose at quadtree level i, each cell has side length $\frac{M}{2^i}$. Let X_i be the number of GGMST edges that cross cell boundaries at level i. If every crossing edge is at least $\alpha\,\frac{M}{2^i}$ in length, then: $X_i \leq \frac{\beta\, W(T_{opt})}{\alpha\,(M/2^i)}$, for some constant $\alpha > 0$ and $\beta > 0$.*

Lemma 4 (Portal Approximation) *There is a way to choose $O(1/\epsilon)$ portals per cell boundary so that rerouting GGMST edges through these portals increases the GGMST cost by at most $\epsilon \cdot W(T_{opt})$.*

DP state definition and encoding connectivity. Consider a cell C in the quadtree. The DP state $D(C, S, B)$ might encode: (i) C: The cell for which a GGMST is being computed. (ii) S: Which clusters fully contained in C have chosen their representative point. This controls the GGMST to be a forest and then connects to other forests at a higher-level cell. Since each leaf cell has at most one cluster, and these sets merge as we go up, S is manageable. (iii) B: A description of how GGMST connections that leave C through its boundary portals are arranged. This is a pattern representing how portals are connected internally.

- *Portal Partitions:* To represent how portals connect inside C, we consider a partition of the set of portals on C's boundary into connected components. Each connected component represents a partial GGMST structure inside C that connects some subset of these boundary portals. The number of partitions of a set of m elements is given by the mth Bell number B_m, which grows super-exponentially in k. For $m = p = O(1/\epsilon)$, we know that Bell number B_m is: $B_m = \Theta\big(m^m/(e^m \cdot m^{3/2})\big)$. Hence $B_m \leq m^m = O(1/\epsilon)^{O(1/\epsilon)}$. This encoding is crucial. By storing a "minimal structure" that indicates which portals are connected, we can later merge these configurations at higher-level cells.

- *DP Recurrence:* For a leaf cell C, we know exactly which points it contains (at most a constant number). We try all possible ways of picking one point for the cluster (if there is one) and then determine how this affects portal connectivity. In particular, for each choice, we compute the partial GGMST cost inside C (often trivial since it is just one cluster plus some boundary edges to portals). Then, we encode the resulting portal connectivity in a partition B.

$$D(C,S,B) = \underbrace{\mathrm{PortalsCost}(p,\,B)}_{\text{how } p \text{ connects to boundary portals}}$$

where $\mathrm{PortalsCost}(p,B)$ is the cost of hooking up p to the boundary portals in a way that yields the partition B. In other words, how does p connect to (some or all) of the portals in B? This ensures that the internal MST structure is consistent with B. For an internal cell C with children C_1, C_2, C_3, C_4, we combine their solutions:

$$D(C,S,B) = \min_{\substack{S_1\cup S_2\cup S_3\cup S_4=S\\ B_1,B_2,B_3,B_4}} \Bigg(\sum_{i=1}^{4} D(C_i,S_i,B_i) + \textsc{MergeCost}(B_1,B_2,B_3,B_4,B) \Bigg)$$

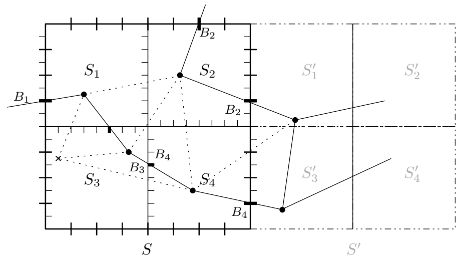


Figure 1: Connected components in cell S have formed through a detour through the adjacent cell S' reckoned by MergeCost. The dotted lines display the edges of the underlying spanner. A partition of active (bold) portals on the boundary can be stored in B.

where MergeCost involves adding edges from the spanner G between portals to ensure the combined configuration at C respects partition B, see Figure 1.

Analysis: approximation and running time. The approximation factors set to $\epsilon/3$ for various steps such as cluster discretization, spanner construction, and portal partitioning combine to yield a $(1+\epsilon)$-approximation overall. We now give a detailed view of the running time analysis, including how the complexity of handling portals is controlled. The runtime analysis is as follows:

(1) We discretize each of N cell into $O(N^d/\epsilon^d)$ cells, hence it takes $O(N^{d+1}/\epsilon^d)$ time. (3) We build a $(1+\epsilon/3)$-spanner G in time of $O(n\log n + n/\epsilon^d)$. (2) We build a quadtree on N cells in time $O(N\log N)$. The quadtree has $O(N)$ cells since we stop subdividing when each cell has at most a constant number of points and at most one cluster. Each leaf cell is thus simple to handle. Suppose each leaf has $l = O\big(\frac{N^d}{\epsilon^d}\big)$ candidate points. For each candidate grid point, we check edges to up to $p = O\big(\frac{1}{\epsilon}\big)$ portals. That's $O(l\cdot k) = O\big(\frac{N^d}{\epsilon^d}\cdot\frac{1}{\epsilon}\big)$. Summation across all leaves, since we have $O(N)$ leaves, the total is $N \times O\big(\frac{N^d}{\epsilon^d}\cdot\frac{1}{\epsilon}\big) = O\big(\frac{N^{d+1}}{\epsilon^{d+1}}\big)$. (4) At internal cells, we use a dynamic programming (DP) approach. The DP states consider: (4a) Which of the four subcells are represented in the cell (chosen at lower levels). (4b) How the GGMST inside the subcells connects to one another outside via a set of boundary portals. The challenge is that at each cell, we must consider different ways to connect portals. The number of ways to partition a set of m elements into connected subsets is at most the mth Bell number $B_m = (1/\epsilon)^{O(1/\epsilon)}$. Since the number of the cells in the quadtree is $O(N)$, the total would be $O(N(1/\epsilon)^{O(1/\epsilon)})$. Hence, the total runtime is $O(N^{d+1}/\epsilon^d) + O(n\log n + n/\epsilon^d) + O(N\log N) + O((N/\epsilon)^{d+1}) + O(N(1/\epsilon)^{O(1/\epsilon)})$, which is: $O\big(n\log n + n/\epsilon^d + (N/\epsilon)^{d+1} + N(1/\epsilon)^{O(1/\epsilon)}\big)$.

Theorem 5 *Given an instance $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ with $|\mathcal{N}| = N$ and $|P| = n$, where clusters are axis-aligned unit hypersquares in $\mathbb{R}^d$, for any $\epsilon > 0$, one can compute a $(1+\epsilon)-$GGMST of $\mathcal{I}$ in $O^*\big((1/\epsilon)^{O(1/\epsilon)}N + N^{d+1} + n\log n\big)$ time, where O^* hides factors polynomial in $1/\epsilon$.*

4 An FPT algorithm for exact GGMST

Our algorithm breaks down into two levels by mixing a coarse dynamic program to enumerate all BBSTs like T and a fine-grained dynamic programming algorithm to obtain OptMST(T) (using the algorithm in Lemma 1). However, the number of backbone spanning trees can grow exponentially with the size of the graph, making enumeration computationally challenging. Given an instance $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ with $|\mathcal{N}| = N$ and $|P| = n$, we present an algorithm that efficiently enumerates all spanning trees of our $\mathcal{G} = (V, E)$ with bounded treewidth κ, exploiting the structure provided by a nice tree decomposition, where V is the set of clusters and E is the set of edges representing the adjacency between the hypernode in V. Computing the exact treewidth of a graph is NP-complete [1]. However, Korhonen [12] presented an algorithm that computes a tree decomposition of width k' satisfying $\kappa' \leq 2\kappa + 1$, where κ is the optimal treewidth of $\mathcal{G}$. Given a graph G with treewidth κ, one can compute a tree decomposition of $\mathcal{G}$ with width $k' \leq 2k+1$ in time $O(2^{O(k)}\cdot N)$ [12].

In our algorithm, we use the tree decomposition obtained from Korhonen's algorithm and κ' will be used as the treewidth parameter in our analysis. Every tree decomposition can be transformed into a "nice" tree decomposition with at most $O(N)$ nodes [11]. Our goal is to enumerate all combinatorially different spanning trees of $\mathcal{G}$. We achieve this by performing dynamic programming over the nice tree decomposition of $\mathcal{G}$. The key idea is to represent partial solutions at each node of the decomposition and combine them according to specific rules that ensure correctness and avoid redundancy.

Our algorithm consists of several components that update the state of connectivity of vertices in each bag and the acyclicity of the spanning tree induced by the vertices in the bag.

State Representation: At each node t of the nice tree decomposition, we maintain a set of DP states. Each state represents a set of partial spanning trees consistent with the subgraph induced by the vertices seen so far.

Definition 6 (DP State) *A DP state at node t is a tuple (σ, δ) where:*

(1) σ is a partition of the vertices in X_t, representing the connected components among them in the partial spanning tree.

(2) $\delta \subseteq E_t$ is the set of edges included in the partial spanning tree, where E_t is the set of edges between vertices in X_t.

Note that each state must satisfy the following invariants: (1) *Acyclicity:* The edges in δ do not form cycles, and (2) *Consistency:* The partition σ accurately reflects the connectivity induced by δ. Let DP_t denote the set of DP states at node t. We define recursive formulas for computing DP_t based on the type of node. We have the following breakdowns for our DP recurrences:

- *Leaf Nodes:* $\mathrm{DP}_t = \{(\emptyset, \emptyset)\}$.

- *Introduce Nodes:* Let t be some node which is an introduce node introducing vertex v, with child t'. The DP states are computed as: $\mathrm{DP}_t = \bigcup_{(\sigma',\delta')\in \mathrm{DP}_{t'}} \bigcup_{\delta_v \subseteq E_v} \{(\sigma_v, \delta_v \cup \delta') \mid \mathrm{Valid}(\sigma_v, \delta_v \cup \delta')\}$,

where: E_v is the set of edges between v and vertices in $X_{t'}$, δ_v is a subset of E_v (possible ways v can connect to $X_{t'}$), σ_v is the updated partition obtained by adding v to σ' and merging parts if v is connected to vertices in δ_v, and $\mathrm{Valid}(\sigma_v, \delta_v \cup \delta')$ ensures acyclicity and consistency.

- *Forget Nodes:* Let t be a forget node forgetting vertex v, with child t'. The DP states are computed as: $\mathrm{DP}_t = \{(\sigma|_{X_t}, \delta') \mid (\sigma', \delta') \in \mathrm{DP}_{t'}\}$, where $\sigma|_{X_t}$ is the restriction of the partition σ' to X_t.

- *Join Nodes:* Let t be a join node with children t_1 and t_2. The DP states are computed as:

$$\mathrm{DP}_t = \Big\{(\sigma, \delta_1 \cup \delta_2) \;\Big|\; (\sigma, \delta_1) \in \mathrm{DP}_{t_1}, (\sigma, \delta_2) \in \mathrm{DP}_{t_2}, \mathrm{Valid}(\sigma, \delta_1 \cup \delta_2)\Big\}.$$

The function $\mathrm{Valid}(\sigma, \delta)$ returns true if the edges in δ do not form cycles. The partition σ correctly represents the connected components induced by δ. See Appendix for further explanation on the recursive formulas above.

To analyze the runtime, let $N = |V|$ be the number of vertices in $\mathcal{G}$ and κ' be the width of the tree decomposition obtained from Korhonen's algorithm. At each node t, the size of the bag is $|X_t| \leq \kappa' + 1$. The number of possible partitions σ of X_t is given by the Bell number $B_{k'+1}$ and the number of possible edge subsets δ among vertices in X_t is $2^{\binom{\kappa'+1}{2}}$. Since for any $m > 0$, the mth Bell number is: $B_m = \left((m/e)^m m^{3/2}\right)$, the total number of states at each node, and hence the required space is: $S(N, \kappa) = B_{\kappa'+1} \cdot 2^{\binom{\kappa'+1}{2}} = O\Big(2^{O(\kappa^2)}\Big(O\big(\kappa/e\big)^{2\kappa+2} O(\kappa)^{3/2}\Big)\Big) = O\left(2^{O(\kappa^2)}\right)$, Since $\kappa' \leq 2\kappa + 1$. Regarding time complexity, we need to explore each:

(I) *Introduce Nodes*: For each state in $\mathrm{DP}_{t'}$, we consider $2^{|E_v|}$ subsets δ_v. Note that $|E_v| \leq \kappa'$, thus we have $2^{\kappa'}$ possibilities. The runtime per introduce node: $O(S \cdot 2^{\kappa'}) = O(S \cdot 2^{O(\kappa)})$. (ii) *Forget Nodes*: For each state in $\mathrm{DP}_{t'}$, we perform operations in $O(1)$ time. Hence The runtime per forget node: $O(S)$. (iii) *Join Nodes*: For each pair of states from the two children, we check for compatibility and merge. Number of state pairs is $O(S^2)$. To obtain the total runtime: The number of nodes in the nice tree decomposition is $O(N)$. Hence, the total time complexity is: $T(N, \kappa) = O\big(N(S^2 + S \cdot 2^{O(\kappa)})\big) = O\left(N\big(2^{O(\kappa^4)}\big)\right)$.

Theorem 7 *Given $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ with $|\mathcal{N}| = N$, $|P| = n$, and κ as the treewidth of $\mathcal{G}$, one can compute the exact* GGMST *of $\mathcal{I}$ in time $O\left(\rho^2 N^2 \big(2^{O(\kappa^4)}\big)\right)$, where ρ is the maximum number of points in across all clusters in $\mathcal{N}$.*

5 Approximation and FPT algorithms in $\mathbb{R}^2$

We view the grid column-by-column. For any two consecutive columns only the cells that appear in those columns can be joined, so a *backbone forest* needs to be chosen inside every $k \times 2$ window. A direct enumeration shows that at most 280^k distinct forests can occur in a $k \times 3$ strip; hence we can exhaustively list every candidate backbone for the first three columns, and for each of them run a dynamic program that selects the best point from every non–empty cell and wires the choices together. It takes $O(\rho^2\, N_3)$ time on a single forest, where ρ bounds the points per cluster and N_3 is the number of non-empty cells in the first three columns. We store the optimal value together with the induced forest on the last two columns and then slide the $k \times 2$ window one column to the right. At step i we match every stored forest on columns $(i-2, i-1)$ against each of the at most 280^k possibilities on $(i-1, i)$; the same dynamic program, now charged to the non-empty cells in $(i-2, i-1, i)$, updates the table. After the last column is processed the cheapest entry whose backbone is a tree is the optimum. Overall running time: $O(N\,\rho^2\, 280^k)$.

Theorem 8 *Given $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ in $\mathbb{R}^2$ with $|\mathcal{N}| = N$ and $|P| = n$ and with k fixed rows. The optimal* GGMST *of $\mathcal{I}$ can be computed in $O(N\rho^2 280^k)$ time.*

Replacing the exact dynamic programme inside the sliding window by the PTAS of [5] yields a $(1+\varepsilon)$-approximation in $O\big(N\,\rho^2\, 280^{1/\varepsilon}\big)$ time.

Corollary 9 *There is $(1+\epsilon)$-approximation algorithm for* GGMST *of $\mathcal{I}$ in $\mathbb{R}^2$ that runs in $O(N\rho^2 280^{\frac{1}{\epsilon}})$ time.*

We say $(\mathcal{G}, \mathcal{N}, P)$ is *γ-bounded* if every non-empty cluster (i, j) has at most γ non-empty clusters in row i or column j. For a γ-bounded instance every column holds at most γ non–empty cells, so within any 3 consecutive columns we must decide how the vertices of a graph on $\leq 3\gamma$ labelled cells are joined. Lemma 14 (see Appendix) shows that the number of legal patterns is bounded by 280^γ (the constant 280 appears exactly as in Lemma 12, but raised only to γ instead of k). During the sweep we must remember, for every cell that sits on the border between the two columns of the sliding window, whether its incident backbone edge continues left or right; this doubles the local state space, whence the factor $(2 \cdot 280)^{\gamma} = 560^{\gamma}$. Apart from this bookkeeping tweak, the algorithm is identical to the one sketched above: enumerate, run the same $O(\rho^2)$ dynamic program per pattern, and slide the window. The result is an FPT–algorithm as state in the following theorem:

Theorem 10 *Given $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ in $\mathbb{R}^2$ that is γ-bounded with $|\mathcal{N}| = N$ and $|P| = n$. The* GGMST *of $\mathcal{I}$ can be computed in $O(N\rho^2 560^{\gamma})$ time.*

6 Concluding remarks

In this paper we proposed some exact and approximation algorithms improving the runtime on the latest results on the GGMST problem. Our algorithms work in any dimension d and make use of some known techniques, the one by Arora appeared in [2]. Our main contribution is indeed the $(1 + \epsilon)$-approximation for GGMST in $\mathbb{R}^d$. Arora's original PTAS for TSP relies on portal pairings—ensuring entry and exit in matching pairs—to construct a tour, whereas our GMST variant instead tracks connected subsets (partitions) of portals to capture how GMST edges merge boundary portals into forests, achieving an approximation error of $\epsilon \cdot W(T_{opt})$ via a geometric series and Bell number analysis. Unlike TSP, where every point must appear in the tour, our formulation requires exactly one point per cluster; these activations are encoded at the leaf cells of our dynamic program to prevent redundancy at higher levels. Furthermore, since edges can only connect adjacent unit squares, the complexity associated with long-range edges is reduced. While Arora's method typically places $O\left(\frac{\log n}{\epsilon}\right)$ portals per cell side—resulting in runtime factors such as $n^{\mathcal{O}(1/\epsilon)}$—our approach utilizes only $O\left(\frac{1}{\epsilon}\right)$ portals, distributing overhead hierarchically across the quadtree levels, thereby avoiding the extra $\log n$ factor in portal spacing. As with Arora's technique, the overall runtime depends super-polynomially on $\frac{1}{\epsilon}$, confirming that our method is a PTAS: it is polynomial in n for any fixed ϵ, but not fully polynomial in $\frac{1}{\epsilon}$. Our other algorithms are also improving on some known results by [5] in $\mathbb{R}^2$ and provide fixed parameter tractablility for the case where the GMST instances behave naturally such as having bounded treewidth and/or γ-bounded as we introduced in this paper.

References

[1] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987.

[2] S. Arora. Polynomial-time approximation schemes for euclidean tsp and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[3] B. Bhattacharya, A. Ćustić, A. Rafiey, A. Rafiey, and V. Sokol. Approximation algorithms for generalized mst and tsp in grid clusters. In Z. Lu, D. Kim, W. Wu, W. Li, and D. Z. Du, editors, *Combinatorial Optimization and Applications*, volume 9486 of *Lecture Notes in Computer Science*, pages 117–129. Springer, Cham, 2015.

[4] B. K. Bhattacharya, A. Custic, A. Rafiey, A. Rafiey, and V. Sokol. Approximation algorithms for generalized MST and TSP in grid clusters. In *Combinatorial Optimization and Applications - 9th International Conference, COCOA 2015, Houston, TX, USA*, volume 9486 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2015.

[5] C. Feremans, A. Grigoriev, and R. Sitters. The geometric generalized minimum spanning tree problem with grid clustering. *4OR*, 4:319–329, 2006.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[7] B. L. Golden, S. Raghavan, and D. Stanojevic. Heuristic search for the generalized minimum spanning tree problem. *INFORMS Journal on Computing*, 17(3):290–304, 2005.

[8] S. Har-peled. *Geometric Approximation Algorithms*. American Mathematical Society, USA, 2011.

[9] H. Jiang and Y. Chen. An efficient algorithm for generalized minimum spanning tree problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 217–224, 2010.

[10] H. A. Kachooei, M. Davoodi, and D. Tayebi. On the generalized minimum spanning tree in the euclidean plane. In *Proceedings of the 1st Iranian Conference on Computational Geometry (ICCG)*, pages 19–23, 2018.

[11] T. Kloks. *Treewidth: computations and approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994.

[12] T. Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 2021.

[13] Y.-S. Myung, C. ho Lee, and D. wan Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

[14] P. C. Pop. The generalized minimum spanning tree problem. 2002.

[15] P. C. Pop, G. Still, and W. Kern. An approximation algorithm for the generalized minimum spanning tree problem with bounded cluster size. In *Proceedings of the Algorithms and Complexity in Discrete Mathematics (ACiD)*, pages 115–121, 2005.

Appendix

This appendix discloses the omitted proofs and algorithmic details skipped in each dedicated section.

Proof of lemmas and theorems in Section 3

Lemma 1 *For a given* BBST T*, one can compute the* OptMST(T) *in polynomial time.*

Proof. Select an internal vertex of T corresponding to a cluster C as the root. For any leaf cluster $l \in T$, and for any point $p \in l$, set $W(T_p) = 0$. For any internal node $I \in T$, and any point $q \in I$, let $\mathsf{Chd}(I)$ denote the set of children of I in BBST T. Then we have: $W(T_q) = \sum_{c \in \mathsf{Chd}(I)} \min_{p \in c} \Big\{ W(T_p) + \|p - q\| \Big\}$. The algorithm runs in $O(N\rho^2)$ time using a forward and then backward breadth-first search on T since the number of children per node is $O(2^{d+1}) = O(1)$, where $d = O(1)$ and $\rho < n$ is the maximum number of points in a cluster. □

Lemma 2 (Discretization) *Let $\epsilon > 0$. We can choose $\delta = \epsilon/(\sqrt{d}(N-1))$ sufficiently small so that there exists a $(1+\epsilon)$-approximate GGMST solution selecting from these candidate points. Moreover, the number of grid points overall is $O(N^d/\epsilon^d)$*

Proof. Consider an optimal GGMST solution T_{opt} that selects a point q_i in each cluster C_i. By construction of the grid inside C_i, there is a grid point p_i no more than $\delta\sqrt{d}$ away from q_i (since the cluster is unit size and we have a δ-spaced lattice).

Replace each q_i by p_i to obtain a new (approximate) solution T'. The increase in the length of any edge (q_i, q_j) in T_{opt} is at most proportional to δ because shifting points can increase distances by at most $\sqrt{d}\delta$. Thus, the total additional cost incurred by this rounding is at most $\sqrt{d}\delta N$, since the MST has at most $N-1$ edges and each edge length changes by at most $\sqrt{d}\delta$.

In the case that $N \le 2^d$ and assuming $d = O(1)$, one can enumerate all $N^{N-2} = 2^{(d2^d)} = O(1)$ many BBSTs and compute GGMST with a minimum total weight in polynomial time in an exact manner utilizing the algorithm described in Lemma 1. For all $N > 2^d$, following the packing argument demonstrated in Lemma 1 in [3], we derive that $W(T_{\text{opt}}) > 1$. Hence: $W(T') \le W(T_{\text{opt}}) + (N-1) \cdot \sqrt{d}\delta = W(T_{\text{opt}}) + (N-1) \cdot \frac{\epsilon}{N-1} = W(T_{\text{opt}}) + \epsilon \le W(T_{\text{opt}}) + \epsilon \cdot W(T_{\text{opt}}) = (1+\epsilon) \cdot W(T_{\text{opt}})$. □

Lemma 3 *Let T_{opt} be an GGMST of cost $W(T_{opt})$ in a bounding box of side-length M. Suppose at quadtree level i, each cell has side length $\frac{M}{2^i}$. Let X_i be the number of GGMST edges that cross cell boundaries at level i. If every crossing edge is at least $\alpha\,\frac{M}{2^i}$ in length, then: $X_i \le \frac{\beta\, W(T_{opt})}{\alpha\,(M/2^i)}$, for some constant $\alpha > 0$ and $\beta > 0$.*

Proof. Any GGMST edge e crossing a cell boundary at level i has length at least $\alpha\,\frac{M}{2^i}$, because it spans distinct cells of side $\frac{M}{2^i}$, for some $\alpha > 0$. Suppose there are X_i such crossing edges. Then these X_i edges alone have total length

$$\sum_{e \text{ crosses at level } i} \|e\| \;\ge\; X_i\Big(\alpha\,\frac{M}{2^i}\Big).$$

Applying a standard geometric pigeonhole argument, the GGMST cannot afford a set of edges whose combined length exceeds $\beta \cdot W(T_{\text{opt}})$, for some constant $\beta > 0$. Concretely, if the total length of all crossing edges at level i were allowed to surpass $\beta \cdot W(T_{\text{opt}})$, one could reconstruct a substructure heavier than $W(T_{\text{opt}})$, contradicting T_{opt}'s minimality.

Consider the MST edges crossing cell boundaries at level i, each having length at least $\alpha\,\frac{M}{2^i}$. The *pigeons* here are precisely those boundary-crossing edges themselves, while the *holes* represent "slots" in the MST's total cost $W(T_{\text{opt}})$. In other words, if each crossing edge (pigeon) demands at least an $\alpha\,\frac{M}{2^i}$ share of the MST length budget (a hole), and the MST cannot allocate more than $W(T_{\text{opt}})$ total, then having "too many" such edges (pigeons) would exceed the MST's entire weight (holes). Hence:

$$\sum_{e \text{ crosses at level } i} \|e\| \;\le\; \beta W(T_{\text{opt}}).$$

Combining both inequalities yields:

$$X_i\big(\alpha\,\tfrac{M}{2^i}\big) \le \beta\, W(T_{\text{opt}}),$$

which rearranges to

$$X_i \le \frac{\beta\, W(T_{\text{opt}})}{\alpha\,\big(\frac{M}{2^i}\big)} = \frac{2^i\,\beta\, W(T_{\text{opt}})}{\alpha\, M}.$$

If X_i were any larger, the total cost of those crossing edges alone would exceed $W(T_{\text{opt}})$, which is a contradiction. □

Lemma 4 (Portal Approximation) *There is a way to choose $O(1/\epsilon)$ portals per cell boundary so that rerouting GGMST edges through these portals increases the GGMST cost by at most $\epsilon \cdot W(T_{opt})$.*

Proof. Following the geometric scaling argument from Arora's PTAS, at each level of the quadtree, the cost added by snapping edges to portals is a small fraction of the GGMST's cost *at that scale*. Summing these fractions over all levels (cells at different scales) yields at most $\epsilon \cdot W(T_{\text{opt}})$ in total overhead. Let the bounding box of the entire cluster set have side length M. The quadtree is built by recursively splitting into four equal squares. At level 0 (the root), cells have side length M. At level i, cells have side length $\frac{M}{2^i}$. We place $p = O\big(\frac{1}{\epsilon}\big)$ portals on each cell boundary, so the portal spacing at level i is on the order of $\frac{M}{2^i \cdot p} = O\big(\epsilon\,\frac{M}{2^i}\big)$. The cost analysis unfolds as follows:

- *Overhead per Crossing at Level i:* Consider an GGMST edge e that crosses a boundary at level i. Snapping e from its original crossing point to the nearest portal adds at most $O\big(\epsilon\,\frac{M}{2^i}\big)$ extra length, since the portal spacing is $O\big(\epsilon\,\frac{M}{2^i}\big)$.

- *Bounding the Number of Crossings:* We now focus on all GGMST edges crossing cell boundaries at level i. Each such crossing indicates an edge of length at least $\Theta\big(\frac{M}{2^i}\big)$, since it spans across different cells at that scale. Following Lemma 3, if there were too many such "long" edges, their combined length would exceed $W(T_{\text{opt}})$, contradicting the GGMST's minimality. Hence, we conclude that the *total length* of edges crossing boundaries at level i is bounded by some constant factor of $W(T_{\text{opt}})$. Since each crossing can add up to $O\big(\epsilon\,\frac{M}{2^i}\big)$ extra length, the total *overhead* at this level is at most $(\epsilon \cdot c) \cdot W(T_{\text{opt}})$ for some universal constant $c > 0$.

- *Summation Over All Levels:* The quadtree may have $O(\log N)$ levels (or $O(\log M)$). If we allocated a full $\epsilon\, W(T_{\text{opt}})$ overhead *per level*, we could end up with $O(\log N) \cdot \epsilon\, W(T_{\text{opt}})$. To avoid this, we use a *geometric distribution* of overhead: $\text{Overhead}_i \leq \frac{\epsilon}{2^i} W(T_{\text{opt}})$, at each level i. By placing portals more densely at coarser levels, the detour at scale i is forced to be only $\frac{\epsilon}{2^i} W(T_{\text{opt}})$. Summing these over $i = 0$ to $\log N$ gives Total Overhead $= \sum_{i=0}^{\log N} \text{Overhead}_i \leq \epsilon\, W(T_{\text{opt}}) \sum_{i=0}^{\log N} \frac{1}{2^i} < 2\,\epsilon\, W(T_{\text{opt}})$. Adjusting constants or using $\epsilon' = \frac{\epsilon}{2}$ in the construction ensures the total overhead does not exceed $\epsilon \cdot W(T_{\text{opt}})$. This completes the proof. □

Proof of lemmas and theorems in Section 4

The key idea in Section 4 is to represent partial solutions at each node of the decomposition and combine them according to specific rules that ensure correctness and avoid redundancy.

DP formulation

Our algorithm consists of several components that update the state of connectivity of vertices in each bag and the acyclicity of the spanning tree induced by the vertices in the bag.

- *State Representation:* At each node t of the nice tree decomposition, we maintain a set of DP states. Each state represents a set of partial spanning trees consistent with the subgraph induced by the vertices seen so far.

Definition 11 (DP State) *A DP state at node t is a tuple (σ, δ) where:*

1. *σ is a partition of the vertices in X_t, representing the connected components among them in the partial spanning tree.*
2. *$\delta \subseteq E_t$ is the set of edges included in the partial spanning tree, where E_t is the set of edges between vertices in X_t.*

Note that each state must satisfy the following invariants: (1) *Acyclicity:* The edges in δ do not form cycles, and (2) *Consistency:* The partition σ accurately reflects the connectivity induced by δ. Let DP_t denote the set of DP states at node t. We define recursive formulas for computing DP_t based on the type of node. We have the following breakdowns for our DP recurrences:

- *Leaf Nodes:* $\text{DP}_t = \{(\emptyset, \emptyset)\}$.
- *Introduce Nodes:* Let t be an introduce node introducing vertex v, with child t'. The DP states are computed as:

$$\text{DP}_t = \bigcup_{(\sigma',\delta')\in \text{DP}_{t'}} \; \bigcup_{\delta_v \subseteq E_v} \left\{(\sigma_v, \delta_v \cup \delta') \mid \text{Valid}(\sigma_v, \delta_v \cup \delta')\right\}$$

  where: E_v is the set of edges between v and vertices in $X_{t'}$, δ_v is a subset of E_v (possible ways v can connect to $X_{t'}$), σ_v is the updated partition obtained by adding v to σ' and merging parts if v is connected to vertices in δ_v, and $\text{Valid}(\sigma_v, \delta_v \cup \delta')$ ensures acyclicity and consistency.

- *Forget Nodes:*
  Let t be a forget node forgetting vertex v, with child t'. The DP states are computed as:

$$\text{DP}_t = \left\{(\sigma|_{X_t}, \delta') \mid (\sigma', \delta') \in \text{DP}_{t'}\right\}$$

  where $\sigma|_{X_t}$ is the restriction of the partition σ' to X_t.

- *Join Nodes:* Let t be a join node with children t_1 and t_2. The DP states are computed as:

$$\text{DP}_t = \big\{ (\sigma, \delta_1 \cup \delta_2) \;\mid\; (\sigma, \delta_1) \in \text{DP}_{t_1}, \\ (\sigma, \delta_2) \in \text{DP}_{t_2},\; \text{Valid}(\sigma, \delta_1 \cup \delta_2) \big\}$$

  The function $\text{Valid}(\sigma, \delta)$ returns true if: The edges in δ do not form cycles. The partition σ correctly represents the connected components induced by δ.

Runtime and space analysis

Let $N = |V|$ be the number of vertices in $\mathcal{G}$ and κ' be the width of the tree decomposition obtained from Korhonen's algorithm. At each node t, the size of the bag is $|X_t| \leq \kappa'+1$. The number of possible partitions σ of X_t is given by the Bell number $B_{k'+1}$ and the number of possible edge subsets δ among vertices in X_t is $2^{\binom{\kappa'+1}{2}}$. Since for any $m > 0$, the mth Bell number is: $B_m = \left((m/e)^m m^{3/2}\right)$, the total number of states at each node, and hence the required space is:

$$\begin{aligned} S(N, \kappa) &= B_{\kappa'+1} \cdot 2^{\binom{\kappa'+1}{2}} \\ &= O\left(2^{O(\kappa^2)} \cdot \left(O\left(\frac{\kappa}{e}\right)^{2\kappa+2} \cdot O(\kappa)^{3/2}\right)\right) \\ &= O\left(2^{O(\kappa^2)}\right) \end{aligned}$$

Since $\kappa' \leq 2\kappa + 1$. Regarding time complexity, we need to explore each:

- *Introduce Nodes*: For each state in $\text{DP}_{t'}$, we consider $2^{|E_v|}$ subsets δ_v. Note that $|E_v| \leq \kappa'$, thus we have $2^{\kappa'}$ possibilities. The runtime per introduce node: $O(S \cdot 2^{\kappa'}) = O(S \cdot 2^{O(\kappa)})$.
- *Forget Nodes*: For each state in $\text{DP}_{t'}$, we perform operations in $O(1)$ time. Hence The runtime per forget node: $O(S)$.
- *Join Nodes*: For each pair of states from the two children, we check for compatibility and merge. Number of state pairs is $O(S^2)$.

To obtain the total runtime: The number of nodes in the nice tree decomposition is $O(N)$. Hence, the total time complexity is:

$$T(N, \kappa) = O\big(N(S^2 + S \cdot 2^{O(\kappa)})\big) = O\left(N\left(2^{O(\kappa^4)}\right)\right)$$

Theorem 7 *Given $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ with $|\mathcal{N}| = N$, $|P| = n$, and κ as the treewidth of $\mathcal{G}$, one can compute the exact* GGMST *of $\mathcal{I}$ in time $O\left(\rho^2 N^2 \left(2^{O(\kappa^4)}\right)\right)$, where ρ is the maximum number of points in across all clusters in $\mathcal{N}$.*

Proof. As shown above, given an instance $\mathcal{I}$, with κ as the treewidth of $\mathcal{G}$, one can enumerate all possible BBSTs in time $T(N,\kappa) = O\left(N\left(2^{O(\kappa^4)}\right)\right)$. Once all BBSTs like T are obtained, we can explicitly pick points from the selected clusters that result in the optimal GGMST, i.e., $\mathsf{OptMST}(T)$ using Lemma 1. During enumeration of all BBSTs, we fix each $T \in \mathsf{BBST}$ and utilize the algorithm described in Lemma 1 that runs in $O(N\rho^2)$. Therefore, the total runtime would be: $T(N,n,\kappa) = O\left(\rho^2 N^2 \left(2^{O(k^4)}\right)\right)$. □

Further skipped details in Section 5

Let $(\mathcal{G},\mathcal{N},P)$ be an example of the GGMST problem with N non-empty cells where the non-empty cells are connected. Let ρ be the maximum number of points inside each cluster. We break the algorithms down into a separate lemma below:

Lemma 12 *The number of backbone forests in a grid $k \times 3$ (with some non-empty cells that) is at most 280^k.*

Proof. Let $F(k)$ denote the number of such forests. Let C_1, C_2, C_3 denote the clusters in row $k-1$ and C_4, C_5, C_6 be the clusters in row k. Let H be a graph with vertices $C_1, \cdots, C_6$ and edges $(C_1,C_4), (C_1,C_5), (C_2,C_4), (C_2,C_5), (C_2,C_6), (C_3,C_5)$, $(C_3,C_6), (C_4,C_5), (C_5,C_6)$. Here, the set of edges of H is the adjacency of each of the clusters $C_1, \cdots, C_6$ without considering the adjacency between C_1, C_2 and C_2, C_3 cells. $F(k) \leq h \times F(k-1)$ where h is the number of forests that are sub-graphs of G. It is not difficult to design a computer simulation and verify that $h = 280$. Therefore, $F(k) \leq 280^k$. □

Theorem 8 *Given $\mathcal{I} = (\mathcal{G},\mathcal{N},P)$ in $\mathbb{R}^2$ with $|\mathcal{N}| = N$ and $|P| = n$ and with k fixed rows. The optimal* GGMST *of $\mathcal{I}$ can be computed in $O(N\rho^2 280^k)$ time.*

Proof. Let $\mathcal{F}(i,j)$, where $i \leq j$, denote the set of possible forests restricted to the columns from i to j, covering all k rows of $\mathcal{G}$. By Lemma 12, there exist at most 280^k trees from $\mathcal{F}(1,3)$ that can serve as the backbone of $\mathcal{G}$. For each such tree T, we apply the dynamic programming approach described in Lemma 1 to obtain $\mathsf{OptMST}(T)$. This requires $O(N_3\rho^2 280^k)$ time to determine the optimal forest, where N_3 represents the number of non-empty clusters up to column 3. We store the backbone forest restricted to columns c_2 and c_3 along with the weight of the minimum spanning forest, computed using the dynamic programming approach from Lemma 1. Notice that no BBST connect two clusters which are two columns (rows) apart from each other. Thus, we only need to look at two consecutive columns and enumerate possible forests as backbones.

Next, we apply a sliding window technique, moving from left to right with a window of size $k \times 2$. At step $i-1$, we assume that the minimum spanning forest whose backbone belongs to $\mathcal{F}(i-2,i-1)$ has already been computed. To expand to column i, we must consider all possible forests in $\mathcal{F}(i-1,i)$ that overlap with $\mathcal{F}(i-2,i-1)$. This requires evaluating forests in $\mathcal{F}(i-2,i-1,i)$ to identify the valid overlaps. The selection of actual points from each cluster is again determined using the dynamic programming approach from Lemma 1, which runs in $O(N_i\rho^2)$ time for each given forest, where N_i represents the number of non-empty cells across the three consecutive columns c_{i-2}, c_{i-1}, c_i. Since there are at most 280^k such forests, the runtime at step i is $O(N_i\rho^2 280^k)$. Thus, upon reaching the last column, the overall running time is $O(N\rho^2 280^k)$. Notice that at the end of the last row, we need to make sure we end up with a tree, eliminating the forests that are not tree after all. □

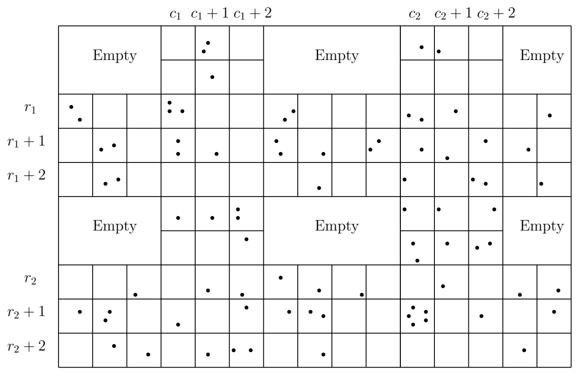


Figure 2: Example of the γ-bounded instance. Note that there could be non-empty cluster in row c_1 as well as in row r_1, etc. The empty text means the clusters without any point in them.

Corollary 9 *There is $(1+\epsilon)$-approximation algorithm for* GGMST *of $\mathcal{I}$ in $\mathbb{R}^2$ that runs in $O(N\rho^2 280^{\frac{1}{\epsilon}})$ time.*

Proof. We apply the approximation algorithm from [5] to obtain an $(1+\epsilon)$-approximation. Their approach includes a subroutine that computes OptMST within a grid of size $k \times l$, running in time $O(l\rho^{6k}2^{34k^2}k^2)$. However, as shown in Theorem 8, this step can be performed in $O(\rho^2 N 280^k)$ time. By carefully analyzing their PTAS, we conclude that a GMST T can be computed such that its weight is at most $(1+\epsilon) \cdot \mathsf{OptMST}$ while maintaining an overall runtime of $O(N\rho^2 280^{\frac{1}{\epsilon}})$. □

Definition 13 (γ-bounded) *Let $(\mathcal{G},\mathcal{N},P)$ be an instance of the* GGMST. *We say $(\mathcal{G},\mathcal{N},P)$ is γ-bounded if for every non-empty cluster (i,j), (in row i and column j), the number of non-empty clusters in row i is at most γ or the number of non-empty clusters in column j is at most γ.*

By a simple adaptation of the argument of the Lemma 12 we have the following lemma (proof is straightforward):

Lemma 14 *The number of backbone forests for instance $(\mathcal{G},\mathcal{N},P)$ which lies in a $l \times 3$ grids and for each column $1,2,3$ have at most γ non-empty cluster is at most 280^γ.*

By a simple adaptation of the argument of the Theorem 8 and Lemma 14, we have the following lemma.

Lemma 15 *Let $(\mathcal{G},\mathcal{N},P)$ be an instance of the* GGMST *problem so that the number of non-empty clusters in every column is at most γ. Then optimal* GGMST *can be computed with running time $O(N\rho^2 280^\gamma)$.*

We are now ready to conclude our main theorem:

Theorem 10 *Given $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ in $\mathbb{R}^2$ that is γ-bounded with $|\mathcal{N}| = N$ and $|P| = n$. The* GGMST *of $\mathcal{I}$ can be computed in $O(N\rho^2 560^\gamma)$ time.*

Proof. Consider that $\mathcal{G}$ is embedded within a grid of dimensions $\alpha \times \beta$. We partition this grid as follows : Let $c_1, c_1 + 1, \ldots, c_1 + \ell_1$ denote a sequence of consecutive columns where each column contains at most γ non-empty clusters. Similarly, let $c_2, c_2 + 1, \ldots, c_2 + \ell_2$ be the next sequence of consecutive columns satisfying where each of them has at most γ non-empty clusters. We continue this partitioning until reaching the last column β (refer to Figure 2 for visualization). Likewise, define $r_1, r_1 + 1, \ldots, r_1 + s_1$ as the first set of consecutive rows where each row contains at most γ non-empty clusters. Define $r_2, r_2 + 1, \ldots, r_2 + s_2$ as the next set of consecutive rows where each also contains at most γ non-empty clusters. We continue this partitioning until reaching the last row α.

Using Lemma 15, we can compute the minimum spanning forest for sub-grid restricted to columns from c_1 to $c_1 + \ell_1$ and all the α rows. Next, we compute the optimal minimum forest within the rows $r_1, r_1 + 1, \ldots, r_1 + s_1$ up to column $c_1 + 1$, denoted as $\mathsf{OptMST}(F_{r_1})$. Observe that the number of possible forests in these rows and the corresponding column c_1 is bounded by 280^γ. Since the dynamic programming approach processes the rows sequentially, we ensure that the constructed forest is compatible with the existing forest on column c_1. This requires $O(N'\rho^2 280^\gamma \times 2^\gamma)$ time, where N' represents the number of non-empty clusters in the rows from r_1 to $r_1 + s_1$. Observe that there are at most 2^γ possible forests on column c_1. A similar procedure is applied to the subsequent set of rows, $r_2, r_2 + 1, \ldots, r_2 + s_2$, and the next block of columns $c_2, c_2 + 1, \ldots, c_2 + s_2$. To track all possible forests restricted to any row and up to column $c_1 + 1$, we spend $O(N_1\rho^2 280^\gamma \times 2^\gamma)$ time, where N_1 represents the number of non-empty clusters in this restricted section. Finally, as we process the columns $c_1 + 1, c_1 + 2, \ldots, c_1 + \ell_1$, we maintain a sliding window of size $\alpha \times 2$. Note that for each of these forests there is another super-forest $\mathcal{S}$ that is a potential part of the OptMST. We compute the $\mathsf{OptMST}(\mathcal{S})$ using the dynamic programming approach in Lemma 1. The approach proceeds iteratively:

1. Process the grid portion between columns $c_1 + \ell_1$ and c_2 row by row until reaching c_2.
2. Continue processing column by column.
3. Repeat 1 and 2 until we reach to the last row and last column in $\mathcal{G}$.

Thus, the overall runtime complexity of the algorithm is bounded by $O(N\rho^2 560^\gamma)$, as required. □

Fault-Tolerant Euclidean k-Centres*

Stephane Durocher† Sahar Lamey† Pak Ching Li†

Abstract

Given a set P of n points in $\mathbb{R}^d$, the Euclidean k-centre problem seeks to select a set F of k points in $\mathbb{R}^d$ such that the maximum distance between any point in P and its nearest neighbour in F is minimized, i.e., cover P with k balls of minimum radius. Upon introducing a parameter ℓ, the ℓ-fault-tolerant Euclidean k-centre problem seeks to minimize the maximum distance from any point in P to its ℓth nearest neighbour in F, i.e., select k balls of minimum radius such that each point in P is contained in ℓ balls. We give an $O(n \log n)$-time algorithm that solves the ℓ-fault-tolerant k-centre problem exactly in $\mathbb{R}$. We show the problem is NP-hard in $\mathbb{R}^2$, and give an $O(nk/\ell)$-time algorithm that finds a 2-approximate solution.

1 Introduction

1.1 Motivation

Given positions for a set of *clients*, a facility location problem seeks to identify positions for a multiset of *facilities* to serve the clients while optimizing a given *cost* function on the relative positions of clients to facilities. In the *k-centre problem*, the cost function is the maximum distance between any client and its nearest facility. E.g., given the locations of 1000 houses, select where to place 10 electric vehicle charging stations such that the maximum distance between any house and its nearest charger is minimized. Chargers are sometimes occupied or out of service, which suggests including *fault tolerance* in the cost function. E.g., select where to place 10 chargers such that the maximum distance between any house and its second-nearest charger is minimized.

1.2 Definitions

Let $\operatorname{dist}(u, v)$ denote the Euclidean (ℓ_2) distance between the points $u, v \in \mathbb{R}^d$. Equivalently, $\operatorname{dist}(u, v)$ is the radius of the smallest ball centred at u that contains v. Given a multiset S of points in $\mathbb{R}^d$ and a positive integer $\ell \leq |S|$, let $\operatorname{dist}_\ell(u, S)$ denote the distance from u to its ℓth nearest neighbour in S. Equivalently, $\operatorname{dist}_\ell(u, S)$ is the radius of the smallest ball centred at u that contains ℓ points of S. Let ℓ, k, and n denote integers such that $1 \leq \ell \leq k \leq n$, let P denote a set[1] of n points in $\mathbb{R}^d$, and let F denote a multiset of k points in $\mathbb{R}^d$. The *cost* of F serving P with fault tolerance ℓ is

$$\operatorname{cost}_\ell(P, F) = \max_{p \in P} \operatorname{dist}_\ell(p, F). \tag{1}$$

Expressed in the terminology of facility location, P is a set of *clients* served by a multiset F of *facilities*. Interpreted geometrically, each point in P is covered by at least ℓ balls in the multiset of k balls of *radius* $r = \operatorname{cost}_\ell(P, F)$, centred on points in F. We examine the problem of finding a multiset F that minimizes (1) for given P, k, and ℓ.

Definition 1 (ℓ-fault-tolerant Euclidean k-centre) *An (optimal)* ℓ-fault-tolerant Euclidean k-centre *of P is a multiset*

$$F^* = \arg\min_{|F|=k} \operatorname{cost}_\ell(P, F). \tag{2}$$

A multiset F' is an *α-approximate* ℓ-fault-tolerant Euclidean k-centre of P if $\operatorname{cost}_\ell(P, F') \leq \alpha \cdot OPT$, where $OPT = \operatorname{cost}_\ell(P, F^*)$. Interpreted geometrically, covering each point in P by ℓ balls centred on points in F' requires a radius at most α times larger than the radius of balls centred on points in an optimal solution F^*.

1.3 Contributions

In Section 4, we describe an $O(n \log n)$-time algorithm that solves the ℓ-fault-tolerant k-centre problem exactly in $\mathbb{R}$. In Section 5, we show the problem is NP-hard in $\mathbb{R}^2$. In Section 6, we describe an $O(nk/\ell)$-time algorithm that finds a 2-approximate ℓ-fault-tolerant k-centre in $\mathbb{R}^2$.

2 Related Work

When $\ell = 1$, Definition 1 corresponds to the *Euclidean k-centre* problem. I.e., the objective is to cover P with k balls of minimum radius. The Euclidean k-centre problem is NP-hard when k is an arbitrary input parameter [11] and $d \geq 2$, remains NP-hard to approximate

*This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

†University of Manitoba, Canada,
stephane.durocher@umanitoba.ca, rahimzas@myumanitoba.ca, ben.li@cs.umanitoba.ca

[1]Whether P is a set or multiset of clients makes no difference to the cost of a solution (1). Due to the fault-tolerance parameter ℓ, however, an optimal solution to (2) may require collocating points in F; therefore, unlike P, F is a multiset.

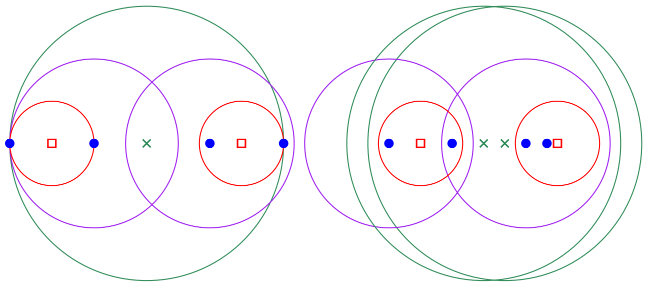


Figure 1: A set P of eight points (blue), a Euclidean 4-centre of P (red □), a discrete 4-centre of P (purple), and a 2-fault-tolerant Euclidean 4-centre of P (green ×) with two facilities collocated on the left ×.

within a factor of $(1+\sqrt{7})/2 \approx 1.8229$ [4], and has an $O(n \log k)$-time 2-approximation algorithm [4]. When $P \subseteq \mathbb{R}$ $(d=1)$, the Euclidean k-centre problem can be solved exactly in $O(n \log n)$ time [2].

The *discrete k-centre* problem examines the corresponding problem (when $\ell = 1$) in the graph setting, where the metric space is limited to the set of graph vertices. That is, facilities must be selected from the set of clients: $F \subseteq P$. Conversely, in the Euclidean k-centre problem, points in F may be positioned anywhere in $\mathbb{R}^d$. The corresponding radii of covering balls for these two versions of the k-centre problem can differ by a factor of two (see Figure 1).

The *ℓ-fault-tolerant discrete k-centre* problem for arbitrary $1 \leq \ell \leq k$ has been examined on graphs whose edge weights define a metric space, where the problem has been shown to be NP-hard, and for which no polynomial-time $(2-\epsilon)$-approximation algorithm is possible for any $\epsilon > 0$ unless P=NP [8]. Chaudhuri et al. [1] gave a polynomial-time 2-approximation algorithm.

As far as the authors are aware, nearly all previous work on fault-tolerant k-centre problems considers the *discrete* k-centre [1, 6, 8, 9]. The results in this paper consider the ℓ-fault-tolerant Euclidean k-centre problem, in which the facilities are not restricted to be a subset of the input point set, but can be any points in $\mathbb{R}^d$. Drezner [3] briefly examined the 2-fault-tolerant Euclidean 2-centre problem (k-centre for *unreliable facilities*) and analyzed the example in Figure 2.

3 Geometric Properties

As with k-centres, an ℓ-fault-tolerant Euclidean k-centre F^* is not unique in general, but $\text{cost}_\ell(P, F^*)$ (radius of the covering balls) is equal for all F^* that minimize (2).

When $\ell > 1$, an ℓ-fault-tolerant Euclidean k-centre is a multiset that sometimes requires collocating multiple facilities in F at a common point (see Figure 1). A natural strategy for identifying k points that approximate an ℓ-fault-tolerant k-centre of P is to place ℓ points on

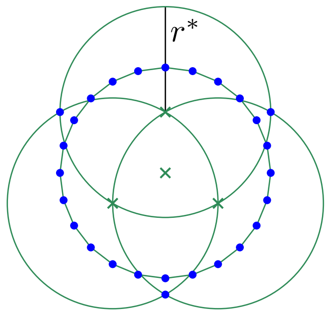


Figure 2: The set P of blue points has a 2-fault-tolerant Euclidean 4-centre (green ×) of radius r^*; all Euclidean 2-centres of P have radius strictly greater than r^* [3].

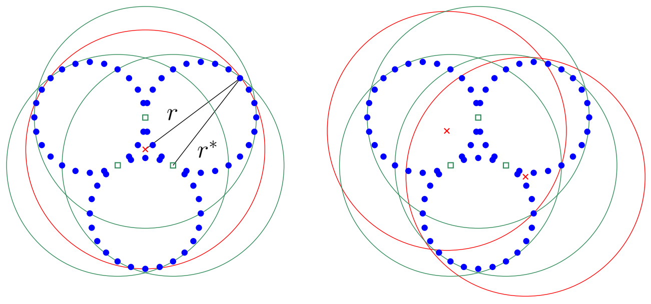


Figure 3: The set P of blue points has a 2-fault-tolerant Euclidean 6-centre (green □) of radius r^*, but all Euclidean 3-centres of P (e.g., red ×) have radius $r = (1/\sqrt{3} + 1/2)r^* \approx 1.0774r^*$.

each centre in a Euclidean $\lfloor k/\ell \rfloor$-centre of P. We examine this technique in Sections 4 and 6. As shown by Drezner [3], this strategy does not give an optimal ℓ-fault-tolerant Euclidean k-centre in $\mathbb{R}^2$ in general (see Figures 2 and 3).

Observation 1 (Drezner 1987 [3]) *$\exists P \subseteq \mathbb{R}^2$, $k \in \mathbb{R}^+$, $\ell \in \mathbb{R}^+$ such that no Euclidean $\lfloor k/\ell \rfloor$-centre of P is an ℓ-fault-tolerant Euclidean k centre of P.*

Finally, collocating facilities of F on a k-centre of P can result in $\text{cost}_\ell(P, F)$ that is arbitrarily larger than $r^* = \text{cost}_\ell(P, F^*)$ for an ℓ-fault-tolerant Euclidean k-centre F^* of P. That is, a Euclidean k-centre cannot guarantee any approximation of an ℓ-fault-tolerant Euclidean k-centre when $\ell > 1$ (see Figure 4).

Observation 2 *$\forall \alpha \in \mathbb{R}$, $\exists P \subseteq \mathbb{R}^2$ such that $|P| = 4$ and $\text{cost}_2(P, F) > \alpha \cdot OPT$, where F is a Euclidean 4-centre of P, F^* is a 2-fault-tolerant Euclidean 4-centre of P, and $OPT = \text{cost}_2(P, F^*)$.*

4 1D Algorithm

In this section, we give an $O(n \log n)$-time algorithm for computing an optimal ℓ-fault-tolerant Euclidean k-centre when the set P of clients is in $\mathbb{R}$. The algorithm

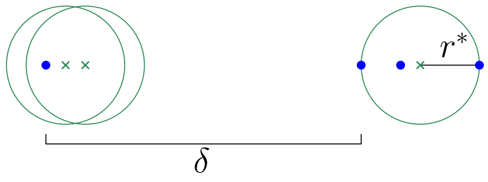


Figure 4: The set P of blue points has a 2-fault-tolerant Euclidean 4-centre of radius r^* (green $\times$) with two facilities collocated on the right $\times$. Setting F to a Euclidean 4-centre of P ($F = P$ in this case) gives $\text{cost}_2(P, F) = \delta$, which can be arbitrarily larger than r^*.

computes an $\lfloor k/\ell \rfloor$-centre of P and outputs ℓ copies of this solution.

Lemma 1 *Given a set P of n points in $\mathbb{R}$, and integers k, ℓ such that $1 \le \ell \le k \le n$, if Z is an ℓ-fault-tolerant Euclidean k-centre of P with $d_Z = \text{cost}_\ell(P, Z)$, then there exists a multiset Y that is a Euclidean k'-centre of P with $\text{cost}_1(P, Y) \le d_Z$, where $k' = \lfloor k/\ell \rfloor$.*

Proof. Let $Z = \{\{z_1, z_2, ..., z_k\}\}$ be a multiset, where $z_1 \le z_2 \le \cdots \le z_k$. Define $Y = \{\{y_1, y_2, ..., y_{k'}\}\}$ where $y_i = z_{i\ell}$, for $i = 1$ to k'. Let $p \in P$. We will now show that $\text{dist}_1(p, Y) \le d_Z$. Since Z is an ℓ-fault-tolerant k-centre of P with cost d_Z, there exist (at least) ℓ consecutive elements of Z that are within distance of d_Z from p. That is, there exists i such that $\text{dist}(p, z_j) \le d_Z$ for $j = i, i+1, ..., i+\ell$. By definition of Y, one of these z_j is in Y. Therefore, $\text{dist}_1(p, Y) \le d_Z$. □

Theorem 2 *Given a set P of n points in $\mathbb{R}$, and integers k, ℓ such that $1 \le \ell \le k \le n$, an optimal ℓ-fault-tolerant Euclidean k-centre can be computed in $O(n \log n)$ time.*

Proof. Let $k' = \lfloor k/\ell \rfloor$. The algorithm computes and returns ℓ copies of S, a Euclidean k'-centre of P.

Let ALG denote the cost of the solution returned by the algorithm and let OPT denote the cost of an optimal solution $\mathcal{O}$. Note that the cost of the Euclidean k'-centre solution S is ALG. By Lemma 1, we can obtain from $\mathcal{O}$ a k'-centre solution C with cost at most OPT. Since S is an optimal k'-centre with cost ALG, we see that $OPT \ge ALG$. Since $ALG \le OPT$ by definition, we conclude that $ALG = OPT$ and the solution computed by the algorithm is optimal.

The running time of the algorithm corresponds to the time required to compute a k'-centre of P. When $P \subseteq \mathbb{R}$, a Euclidean κ-centre of P can be computed in $O(n \log n)$ time for any κ, where $n = |P|$ [2]. Consequently, our algorithm computes an ℓ-fault-tolerant Euclidean k-centre of P in $O(n \log n)$ time. □

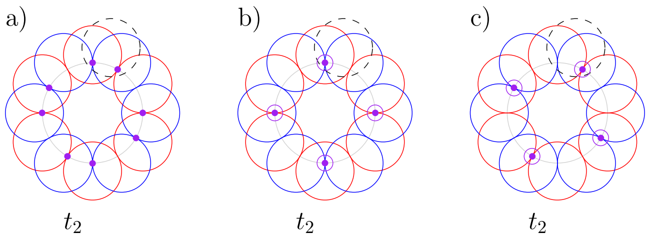


Figure 5: **Variable Gadget (true)**: These three configurations of eight purple points stab each red or blue disc with two points, as well as the dashed disc with two points. Pairs of purple points are collocated in the second and third configurations. The dashed disc is part of a clause gadget.

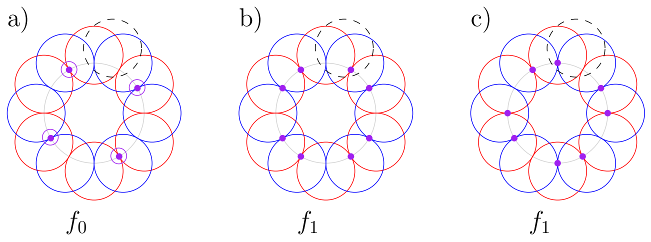


Figure 6: **Variable Gadget (false)**: These three configurations of eight purple points stab each red or blue disc with two points, but stab the dashed disc with only one (f_1) or zero (f_0) points. Pairs of purple points are collocated in the first configuration. The dashed disc is part of a clause gadget.

5 2D Hardness

We show that the 2-fault-tolerant Euclidean k-centre problem is NP-hard by describing a reduction from the Planar 3-SAT problem. An instance $\Phi = (U, E)$ of *3-SAT* consists of a set of variables $U = \{u_1, \ldots, u_n\}$ and a set of clauses $E = \{E_1, \ldots, E_s\}$. Each clause is the disjunction of three literals, each of which is either a variable in U or its negation, e.g., $E_1 = (u_1 \vee \neg u_2 \vee u_3)$. The objective is to determine if there exists an assignment of truth values to the variables in U such that all clauses in E are satisfied (evaluate to true). Every instance Φ of 3-SAT corresponds to a bipartite graph, G_Φ, with vertex set $U \cup E$, and an edge (u_i, E_j) if and only if the variable u_i is in the clause E_j. *Planar 3-SAT* is a restricted version of 3-SAT for which G_Φ is planar. Planar 3-SAT is NP-complete [10]. Furthermore, for every instance Φ of Planar 3-SAT, there exists a non-crossing rectilinear embedding of G_Φ in $\mathbb{R}^2$ [7].

We shall establish the NP-hardness of the Fault-tolerant Euclidean k-centre problem by proving that *Disc 2-Stabbing* is NP-hard. The input consists of a set P' of unit discs in $\mathbb{R}^2$ and an integer $k \ge 1$. The objective is to determine if there exists a multiset F of

k points such that each disc in P' contains (is stabbed by) at least two points of F. If so, we say F *2-stabs* P', which occurs if and only if F is a 2-fault-tolerant Euclidean k-centre of the set of centres of discs in P' with cost at most 1. That is, Disc 2-Stabbing reduces directly to ℓ-fault-tolerant Euclidean k-centre.

We now reduce Planar 3-SAT to Disc 2-Stabbing. Choose any instance $\Phi = (U, E)$ of Planar 3-SAT and a rectilinear planar embedding of G_Φ.

5.1 Variable Gadget

For each variable $u_i \in U$, we construct a cycle of $r_i = 3m_i$ overlapping discs that follows the edges adjacent to u_i in the drawing of G_Φ, where m_i is even (see Figures 5 and 6); let $C^i = \{C^i_0, C^i_1, ..., C^i_{r_{i-1}}\}$ denote[2] this set of discs. Position the discs in C^i such that $C^i_a \cap C^i_b \neq \varnothing$ if and only if $(a-b) \bmod r_i \in \{0,1,2\}$; that is, each disc in C^i intersects two discs ahead and two discs behind it in the cycle, but no other disc in C^i. Discs corresponding to u_i should not intersect discs corresponding to any other variable u_j (see Figure 11). That is,

$$\forall i,j,\ \forall C^i_x \in C^i,\ \forall C^j_y \in C^j,\ i \neq j \Rightarrow C^i_x \cap C^j_y = \varnothing.$$

A simple counting argument shows that if $|C^i| = 3m_i$, then at least $2m_i$ points are necessary to 2-stab C^i.

We begin by characterizing the possible 2-stabbing sets of C^i using $2m_i$ points and how these 2-stabbing sets correspond to truth values for variables in the reduction.

Observation 3 *If F' is a multiset of $2m_i$ points that 2-stabs C^i, then each $q \in F'$ must stab three discs in C_i; furthermore, $q \in C^i_j \cap C^i_{j+1} \cap C^i_{j+2}$ for some j.*

Lemma 3 *If F' is a multiset of $2m_i$ points that 2-stabs C^i, then $\forall j \in \{0,1,...,r_{i-1}\}$, $|C^i_j \cap F'| = 2$.*

Proof. By Observation 3, each point in F' stabs three discs in C^i. Since $|F'| = 2m_i$, there are $6m_i$ discs (counting multiplicities) stabbed by F'. Let D denote the multiset of discs stabbed by F'. Since F' is a 2-stabber of C^i, each disc of C^i must occur at least twice in D. As $|D|/|C^i| = 2$, we conclude that each disc of C^i appears exactly twice in D. That is, each disc of C^i is stabbed by exactly two points of F'. □

With Observation 3 and Lemma 3, Lemmas 4 and 5 below show that there are six combinatorially distinct configurations of $2m_i$ points that 2-stab C^i: three cases correspond to the truth value *True*, and the other three correspond to the truth value *False* (see Figures 5 and 6).

Lemma 4 *Suppose F' 2-stabs C^i and $|F'| = 2m_i$. Then for every j, $q_j \cap F' = \varnothing$, $q_{j+1} \cap F' = \varnothing$, or $q_{j+2} \cap F' = \varnothing$, where $q_x \in C^i_x \cap C^i_{x+1} \cap C^i_{x+2}$.*

[2]Indices are taken modulo r_i. E.g., C^i_j denotes $C^i_{j \bmod r_i}$.

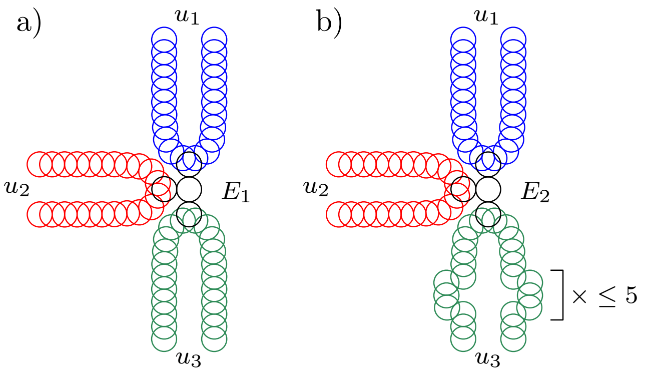


Figure 7: **Clause Gadget.** Each clause E_i is represented by a central disc tangent to three adjacent discs (black), each of which intersects the chain of discs corresponding to one of the three variables in the clause. This example illustrates two clauses: $E_1 = u_1 \vee u_2 \vee u_3$ and $E_2 = u_1 \vee u_2 \vee \neg u_3$. To negate the truth value of a variable, additional discs are added to the variable chain on either side of the clause gadget to switch the truth value of the variable where it intersects the black disc. In this example, one additional disc is added to each side of the green chain to negate u_3 in E_2. Since the number of discs in each variable disc chain must be a multiple of 6, additional discs may be required on the other side of the intersection with the black disc.

Lemma 4 implies that F' cannot contain three points that lie on consecutive intersections of three discs.

Proof. Suppose there is a j such that $\{q_j, q_{j+1}, q_{j+2}\} \subseteq F'$. Without loss of generality, assume $j = 0$. In this case, $|C^i_2 \cap F'| = 3$, contradicting Lemma 3. □

Lemma 5 *Suppose F' 2-stabs C^i, $|F'| = 2m_i$, and at least one point occurs twice in F'. Then every point occurs twice in F'.*

Proof. Suppose some point q_j occurs twice in F' and some point $q_{j'}$ occurs only once in F'. Without loss of generality, suppose $j = 0$, $j' > 0$, and $j' - j$ is minimum among all such q_j and $q_{j'}$, where $q_x \in C^i_x \cap C^i_{x+1} \cap C^i_{x+2}$. By Lemma 3, $j' \notin \{1,2\}$. In order to stab the disc C^i_3, F' must contain q_3. Therefore, $j' = 3$ and, by assumption, q_3 occurs only once in F'. Consequently, F' cannot 2-stab the disc C^i_3, contradicting the assumption that F' 2-stabs C^i. □

We now characterize all possible configurations F' of $2m_i$ points that 2-stab C^i. If F' contains some point twice, then by Lemma 5 F' has m distinct points, each of which appears twice in F'. By Lemma 3, F' must be the multiset $\{\{q_{0+j}, q_{0+j}, q_{3+j}, q_{3+j}, ..., q_{r_i-3+j}, q_{r_i-3+j}\}\}$, for some $j \in \{0,1,2\}$, corresponding to Figures 5(b), 5(c), and 6(a), respectively. In our reduction, we associate

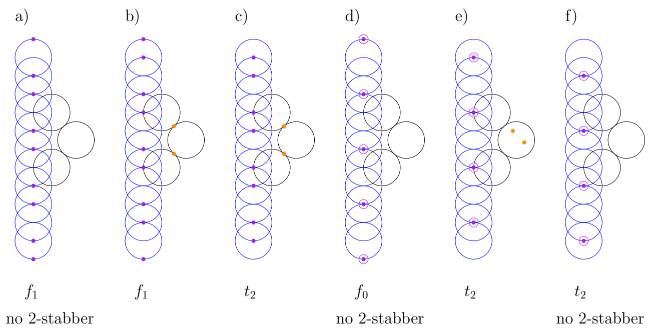


Figure 8: Adding three constraint discs (black) to each variable gadget reduces the number of possible configurations from six to three; the configurations b, c, and e can each be 2-stabbed with two additional points (orange), but a, d, and f cannot.

the 2-stabbing set for $j = 0, 1$ with *True* and the 2-stabbing set for $j = 2$ with *False*.

The only other possibility is that F' contains $2m$ distinct points. In this case, Lemmas 3 and 4 imply that F' is the set $\{q_{0+j}, q_{1+j}, q_{3+j}, q_{4+j}, ..., q_{r_i-3+j}, q_{r_i-2+j}\}$, for some $j \in \{0, 1, 2\}$, corresponding to Figures 5(a), 6(b), and 6(c), respectively. In our reduction, we associate the 2-stabbing set for $j = 0$ with *True* and the 2-stabbing set for $j = 1, 2$ with *False*.

Consequently, exactly six configurations are possible for F'. We further restrict the possible configurations by adding three *constraint discs* to each literal's chain of discs, as illustrated in Figure 8. In three cases, two of the constraint discs contain one or two points that stab C^i, and two additional points are necessary and sufficient to 2-stab all three constraint discs. In the remaining three cases, all three constraint discs are empty and cannot be 2-stabbed by any two additional points. Therefore, this reduces the number of possible configurations to three. Position the three constraint discs such that Figures 8c, e, and b correspond to Figures 5a, 5b, and 6c, respectively. This allows negating a variable before it meets a clause gadget (if the variable is negated in that clause) by adding discs (see Figure 7).

5.2 Clause Gadget and Reduction

We now describe the clause gadgets in our reduction. Each clause $E_h \in E$ is represented by four discs: one central disc that intersects three other discs, each in a point (see Figures 7 and 10). If the clause contains the literal u_i, then the corresponding clause gadget intersect a disc C^i_j such that $j \bmod 3 \in \{0, 1\}$; if the clause contains the literal $\neg u_i$, then $j \bmod 3 = 2$. The index j of the intersecting disc can be controlled by adding discs (see Figure 7). If the assignment of a truth value to u_i implies that some point $q \in C^i_{j-1} \cap C^i_j \cap C^i_{j+1}$ is selected for the solution F, then q may be selected so as to belong to the black disc as well. These correspond exactly to configurations of points for true literals (see Figure 5 and 6).

Observation 4 *A clause gadget can be 2-stabbed by two points if and only if at least one of its literals is true. Furthermore, at least two points are required to 2-stab the clause gadget for every combination of truth assignments to its literals.*

Let Ψ denote the set $\bigcup_{i=1}^n C^i$ of discs in variable gadgets, along with the $4s$ discs for the s clauses in E, and the $3n$ constraint discs for the n variables in U. Let

$$k = 2s + 2n + 2\sum_{i=1}^{n} m_i. \tag{3}$$

We now prove that E is satisfiable if and only if there exists a set F of k points that 2-stabs Ψ. First, assume that E is satisfied by a truth assignment Γ to U. For $i \in \{1, \ldots, n\}$, if $\Gamma(u_i) = $ *True*, then add to F $2m_i$ points that 2-stab C^i as in Figure 5a. Otherwise, $\Gamma(u_i) = $ *False*; add to F $2m_i$ points as in Figure 6b. So far, $|F| = 2\sum_{i=1}^n m_i$, and each set C^i of discs is 2-stabbed by F. By our assumption, each clause $E_h = u_x \vee u_y \vee u_z$ is satisfied; add to F two points per clause as in Figure 10 such that each clause's set of four discs is 2-stabbed by F. Finally, add to F two points per variable as in Figure 8, such that each variable's constraint discs are 2-stabbed by F. Therefore, $|F| = k$ and F 2-stabs Ψ.

To prove the converse, assume F is a set of k points that 2-stabs Ψ. We must show there exists a truth assignment Γ for U that satisfies E. By Observation 4, at least $2s$ points of F are required to 2-stab the clause gadgets and at least $2n$ points of F are required to 2-stab the constraint discs for variable gadgets (excluding points that 2-stab $\bigcup_{i=1}^n C^i$) by (3); this leaves at most $2\sum_{i=1}^n m_i$ points in F to 2-stab $\bigcup_{i=1}^n C^i$. By Observation 3 and Lemmas 4 and 5, $2m_i$ points are necessary in F to 2-stab each C^i; furthermore, these correspond to one of the configurations in Figures 5 or 6. That is, for each C^i, F uniquely determines the truth assignment $\Gamma(u_i)$ associated with that point configuration. By (3), this leaves $k - 2\sum_{i=1}^n m_i = 2s + 2n$ points with which to 2-stab the s clause gadgets (Figures 7 and 10) and the n sets of variable constraint discs (Figure 8). Since each clause gadget is 2-stabbed by F, by Observation 4, each clause in E is satisfied. Since a rectilinear embedding of G_Φ exists on a grid of size $3s \times 3s$, where $s = |E|$ denotes the number of clauses in Φ [10], and each unit length of each edge in the embedding can be represented by a chain of $O(1)$ unit discs, therefore the reduction has polynomial size. See Figure 11 for an example reduction. Given any set P' of unit discs and any set F of points in $\mathbb{R}^2$, it is straightforward to verify in polynomial time whether F 2-stabs P'.

Theorem 6 *Disc 2-Stabbing is NP-complete.*

Since every P' and F in $\mathbb{R}^2$ can be embedded in a flat in $\mathbb{R}^d$ for any $d \geq 2$, this gives the following theorem:

Theorem 7 *For all $d \geq 2$, the 2-fault-tolerant k-centre problem is NP-complete in $\mathbb{R}^d$.*

6 2D Approximation Algorithm

In this section, we give a simple 2-approximation algorithm based on the greedy k-centre approximation algorithm of Gonzalez [5].

Algorithm 1: Computing an approximate ℓ-fault-tolerant Euclidean k-centre solution

```
Input  : A set $P \subset \mathbb{R}^2$ of $n$ points, integers $k, \ell$
         satisfying $0 < \ell \leq k < n$.
Output: A multiset of (at most) $k$ points from
         $P$.
1  $k' \leftarrow \lfloor k/\ell \rfloor$
2  Select an arbitrary point $s_1 \in P$.
3  $S \leftarrow \{s_1\}$
4  $i \leftarrow 1$
5  while $|S| < k'$ do
6      $s_{i+1} \leftarrow \arg\max_{p \in P \setminus S} \mathrm{dist}_1(p, S)$
7      $r_i \leftarrow \mathrm{dist}_1(s_{i+1}, S)$
8      $S \leftarrow S \cup \{s_{i+1}\}$
9      $i \leftarrow i + 1$
10 end
11 return $\ell$ copies of $S$
```

When $\ell = 1$, Algorithm 1 is identical to the greedy 2-approximation algorithm of Gonzalez [5], which in each iteration locates a facility that is farthest from the facilities already selected. Line 7 of the algorithm is used only for the analysis of the algorithm.

Theorem 8 *Algorithm 1 is a 2-approximation for the ℓ-fault-tolerant Euclidean k-centre problem.*

Proof. Let $k' \leftarrow \lfloor k/\ell \rfloor$ and $S = \{s_1, s_2, ..., s_{k'}\}$ be the k' centres selected by Lines 1–10 of Algorithm 1, where s_i is the i^{th} facility selected. The algorithm returns ℓ copies of S and, therefore, the cost of the solution returned is equal to the cost of the Euclidean k'-centre solution S.

Let $r_{k'} = \mathrm{cost}_1(P, S)$ and let $s_{k'+1}$ be a point in P that realizes $r_{k'}$. The value $r_{k'}$ is the cost of the solution returned by (Line 11) of the algorithm as each point in S appears ℓ times in the solution. Since $k < n$, we have $k' < n$ and, therefore, the point $s_{k'+1}$ is distinct from the points in S.

The values $r_1, \ldots, r_{k'-1}$ computed in Line 7 and $r_{k'}$ defined above satisfy $r_1 \geq \cdots \geq r_{k'-1} \geq r_{k'}$. These inequalities imply that any two points in $S \cup \{s_{k'+1}\}$ are at least a distance of $r_{k'}$ apart.

Let OPT denote the cost of an optimal solution for the ℓ-fault-tolerant k-centre instance. It suffices to show that $r_{k'} \leq 2 \cdot OPT$, which we show by contradiction.

Suppose $r_{k'} > 2 \cdot OPT$ and let $\mathcal{O}$ be an optimal solution, where $|\mathcal{O}| = k$. Consider any point $p \in S \cup \{s_{k'+1}\}$. As $p \in P$, there must be at least ℓ facilities in $\mathcal{O}$ that are within distance OPT from p. Since $r_{k'} > 2 \cdot OPT$ and by the triangle inequality, none of these facilities are within a distance of OPT from any member of $S \cup \{s_{k'+1}\} \setminus \{p\}$. This implies that there must be at least $(k'+1)\ell > k$ facilities in $\mathcal{O}$, which is a contradiction. □

The running time of Algorithm 1 is dominated by the nested loops on Lines 5 and 6 that iterate k' and n times, respectively, giving a worst-case running time of $\Theta(nk') = \Theta(nk/\ell)$. Figure 9 gives an example showing that Algorithm 1 cannot guarantee any approximation factor better than 2 in general.

Theorem 8 holds for both the discrete and continuous versions of the ℓ-fault-tolerant k-centre problem over any metric space.

7 Discussion and Directions for Future Research

7.1 Approximation by a $\lfloor k/\ell \rfloor$-Centre

Our example in Figure 3 shows that placing ℓ points on each facility of a Euclidean $\lfloor k/\ell \rfloor$-centre of a set P of points cannot guarantee an approximation for the ℓ-fault-tolerant Euclidean k-centre better than $(1/\sqrt{3} + 1/2) \approx 1.0774$ in general. Theorem 8 shows that a 2-approximation is possible. It remains open to determine what approximation factors are possible in the range $[1/\sqrt{3} + 1/2, 2)$. Our 2-approximation algorithm takes $O(nk/\ell)$ time; it may be possible to apply techniques similar to those used by Feder and Greene [4] to reduce the running time to $O(n \log k)$.

7.2 Hardness of Approximation

We showed that the ℓ-fault-tolerant Euclidean k-centre problem is NP-hard in two or more dimensions. Feder and Greene [4] showed that the Euclidean k-centre problem is NP-hard to approximate within a factor of $(1+\sqrt{7})/2 \approx 1.8229$ in two or more dimensions. A similar hardness of approximation result likely holds for the ℓ-fault-tolerant Euclidean k-centre, but remains open at present. In fact, even without fault tolerance, it remains open whether any polynomial-time algorithm can guarantee an α-approximation of the Euclidean k-centre problem for any $\alpha \in ((1+\sqrt{7})/2, 2)$ in $\mathbb{R}^2$. No such gap exists for the discrete k-centre problem, for which a polynomial-time 2-approximation exists, and it is NP-hard to find a $(2-\epsilon)$-approximation for any $\epsilon > 0$ [5].

References

[1] S. Chaudhuri, N. Garg, and R. Ravi. The p-neighbor k-center problem. *Information Processing Letters*, 65:131–134, 1998.

[2] D. Z. Chen, J. Li, and H. Wang. Efficient algorithms for the one-dimensional k-center problem. *Theoretical Computer Science*, 592:135–142, 2015.

[3] Z. Drezner. Heuristic solution methods for two location problems with unreliable facilities. *Journal of the Operational Research Society*, 38(6):509–514, 1987.

[4] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proc. Symposium on Theory of Computing (STOC)*, 1988.

[5] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[6] S. Khuller, R. Pless, and Y. J. Sussmann. Fault tolerant K-center problems. *Theoretical Computer Science*, 242:237–245, 2000.

[7] D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM Journal on Discrete Mathematics*, 5:422–427, 1992.

[8] S. Krumke. On a generalization of the p-center problem. *Information Processing Letters*, 56:67–71, 1995.

[9] N. Kumar and B. Raichel. Fault tolerant clustering revisited. In *Proc. Canadian Conference on Computational Geometry (CGGG)*, pages 103–108, 2013.

[10] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[11] N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984.

A Appendix: Figures

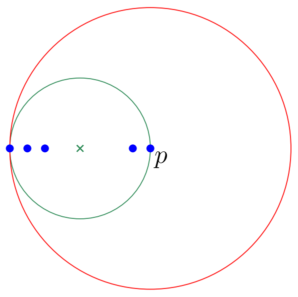


Figure 9: In this example, the unique optimal 2-fault-tolerant 2-centre has two facilities on $\times$, whereas Algorithm 1 could place two facilities on p.

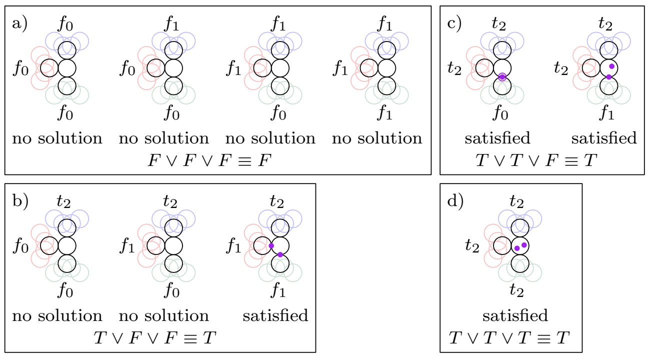


Figure 10: **Clause gadget configurations.** The black discs labelled f_0 and f_1 corresponds to false variables, and contain zero or one points, respectively. The black discs labelled t_2 correspond to true variables, and contain two points. Two additional points (purple) suffice to 2-stab clause gadgets that have one or more true literals (b–d). No configuration of two points can 2-stab a clause gadget with three false literals (a). Two purple points are collocated in the left configuration of (c).

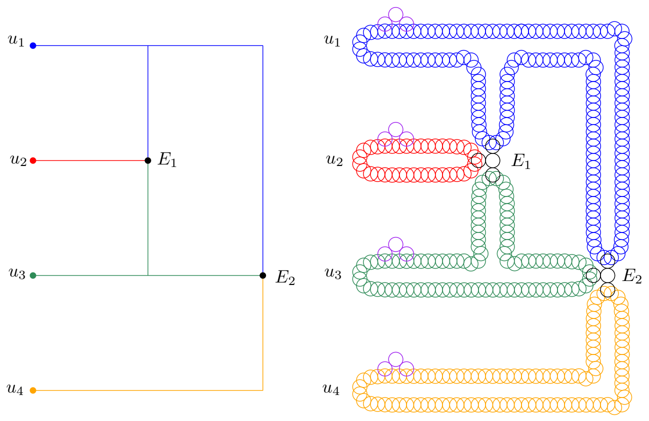


Figure 11: Given a rectilinear planar embedding of G_ϕ (left), we replace the edges adjacent to each variable in U with a cycle of overlapping discs (right) and each clause in E with four discs (black) that overlap the three cycles of discs corresponding to variables in that clause. Each variable also has one set of three constraint discs (purple). This example illustrates the reduction for an instance consisting of the clauses $E_1 = u_1 \vee u_2 \vee u_3$ and $E_2 = u_1 \vee u_3 \vee u_4$.

The Orthogonal Two-Line Center Problem[*]

Taehoon Ahn[†] Sang Won Bae[‡] Sang Duk Yoon[§]

Abstract

Given a set of n points in the plane, the *two-line center problem* asks to find two lines that minimize the maximum distance from each point to its closer line. The best known algorithm for the problem takes $O(n^2 \log^2 n)$ time, presented by Jaromczyk and Kowaluk in 1995. In this paper, we consider the *orthogonal* two-line center problem, a constrained version of the two-line center problem in which two resulting lines should be orthogonal. We present an $O(n^2 \log n)$ time algorithm using only $O(n)$ space.

1 Introduction

In the *two-line center problem*, we are given a set P of n points in $\mathbb{R}^2$ and want to find two lines that minimize the maximum distance to the closer line from each $p \in P$. The first subcubic $O(n^2 \log^5 n)$-time algorithm for the problem was presented in 1991 by Agarwal and Sharir [3]. Later in 1995, Jaromczyk and Kowaluk [15] improved it to $O(n^2 \log^2 n)$ time, and it still remains the best known algorithm until now. See also Katz and Sharir [16] and Glozman et al. [14] for other approaches to the problem.

Whereas no progress on the problem has been made for the last three decades, there has been effort to achieve better running times for variants of two-line center problem with additional constraints. Bae [8] presented an algorithm that runs in $O(n^2)$ time for the *parallel* two-line center problem, in which two resulting lines should be parallel. Very recently, Ahn and Bae [5] studied orientation-constrained versions of the two-line center problem, and presented efficient algorithms with running times: $O(n \log n)$ time when both resulting lines should have fixed orientations, $O(n \log^3 n)$ time when one of them should have a fixed orientation, and $O(n^2 \alpha(n) \log^2 n)$ time when the angle between two resulting lines is fixed, where $\alpha(n)$ denotes the inverse Ackermann function.

Along this line of research, in this paper, we study the *orthogonal* two-line center problem, in which we consider an additional constraint that the two resulting lines should be orthogonal. Our main contribution is a new algorithm for the orthogonal two-line center problem, which takes $O(n^2 \log n)$ time and $O(n)$ space. Note that the orthogonal two-line center problem is a special case of the *fixed-angle* two-line center problem, described above as the third variant of Ahn and Bae [5], whose algorithm makes use of the arrangement of n dual lines via the point-to-line duality and therefore requires $O(n^2)$ space. Thus, our algorithm improves the previous one both in time and space for the orthogonal case.

Related work. Generalizing two-line center problem, the *k-line center problem* is to find k lines that optimally cover a given set P of points in the plane. Notice that the k-line center problem is equivalent to the problem of finding k *strips* of minimum width whose union encloses input points P. The 1-line center problem is known as the *width* problem, and it can be solved in $O(n)$ time after computing the convex hull of P [18]. In $\mathbb{R}^3$ or higher dimensions, Agarwal and Sharir [4] showed that the width of n points in $\mathbb{R}^3$ can be computed in $O(n^{3/2+\epsilon})$ expected time, and Chan [10] discussed an $O(n^{\lceil d/2 \rceil})$-time algorithm in $\mathbb{R}^d$ for $d \geq 4$. For the two-line center problem in $\mathbb{R}^2$, the exact algorithms are known as mentioned earlier, and when the points are given in $\mathbb{R}^d$ for $d \geq 3$, an efficient approximation algorithm for two-line center is presented by Agarwal et al. [1]. The general k-line center problem is known to be NP-hard when k is a part of input [17] even in $\mathbb{R}^2$, while an efficient approximation algorithm is known [2].

In addition to the previous research [5, 8] introduced above, more results on variants of the k-line center problem and their extensions to higher dimensions have recently been studied. Das et al. [12] presented an approximation algorithm for the *axis-aligned* k-line center problem in which resulting lines should be either horizontal or vertical. Chung et al. [11] presented algorithms that compute k parallel strips of best separability, enclosing a given planar set of points. Ahn et al. [6] presented algorithms computing two parallel slabs enclosing points in $\mathbb{R}^d$ for $d \geq 3$.

[*]T. Ahn was supported by the National Institute for Mathematical Sciences (NIMS) Grant funded by the Korean government(MSIT) (No. NIMS-B25910000). S.W. Bae was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2023-00251168).

[†]National Institute for Mathematical Sciences, Daejeon, Korea. taehoon@nims.re.kr

[‡]Division of AI Computer Science and Engineering, Kyonggi University, Suwon, Korea. swbae@kgu.ac.kr

[§]Department of Service and Design Engineering, Sungshin Women's University, Seoul, Korea. sangduk.yoon@sungshin.ac.kr

Due to space limit, proofs are omitted.

2 Preliminaries

Let P be a given set of n points in the plane $\mathbb{R}^2$. We assume that the points in P are in general position; that is, no three points of P lie on a common line. We consider the standard Cartesian coordinate system of $\mathbb{R}^2$ with the *horizontal* x-axis and the *vertical* y-axis. For each point $p \in \mathbb{R}^2$, we use $x(p)$ and $y(p)$ to denote its x- and y-coordinates, respectively.

We say that the *orientation* of a line is $\theta \in [0, \pi)$ (modulo π) if θ is the counterclockwise angle from the x-axis to the line. A line of orientation θ may have a *direction* $\phi \in [0, 2\pi)$, where $\phi = \theta$ or $\phi = \theta + \pi$. For a point $p \in \mathbb{R}^2$, we use $\vec{\ell}_\phi(p)$ to denote the line in direction ϕ that goes through p. A half-line (or a ray) starting at p in direction ϕ is the subset of $\vec{\ell}_\phi(p)$ consisting of all points on the line but those prior to p in direction ϕ.

A *strip* σ is the closed region bounded by two parallel lines, and its orientation is determined as that of its bounding lines. The *width* of strip σ, denoted by $\mathrm{w}(\sigma)$, is the distance between its two bounding lines. For any subset $Q \subseteq P$, we denote by $\sigma_\theta(Q)$ the minimum-width strip of orientation θ that encloses Q.

Note that the orthogonal two-line center problem is equivalent to the problem of finding a pair of orthogonal strips that encloses P such that the width of the wider strip is minimized. For a pair of orthogonal strips σ_1 and σ_2, we call the union $C = \sigma_1 \cup \sigma_2$ of the strips a *cross*. The *orientation* of cross C is meant to be θ for $\theta \in [0, \pi/2)$ if the orientations of σ_1 and σ_2 are θ and $\theta + \pi/2$, respectively. The *width* of cross C is defined by $\mathrm{w}(C) = \max\{\mathrm{w}(\sigma_1), \mathrm{w}(\sigma_2)\}$. The four bounding lines of σ_1 and of σ_2 are also called bounding lines of C.

Consider any cross C of orientation θ. The complement $\mathbb{R}^2 \setminus C$ consists of four quadrants bounded by half-lines of orientation θ or $\theta + \pi/2$. We denote each of the four quadrants by $Q_i(C)$ for $i \in \{0, 1, 2, 3\}$ and two bounding half-lines of $Q_i(C)$ by $h_{2i}(C)$ and $h_{2i+1}(C)$ in direction $\theta + i\pi/2$ and $\theta + (i + 1)\pi/2$, respectively. We simply write Q_i or h_j for indices $0 \le i \le 3$ and $0 \le j \le 7$ instead of $Q_i(C)$ or $h_j(C)$, if it is understood from the context. See Figure 1(a).

3 Data structures

In this section, we describe data structures which will be maintained in our algorithm. We call a cross $C = \sigma_1 \cup \sigma_2$ of orientation θ *minimal* if $\sigma_1 = \sigma_\theta(P \setminus \sigma_2)$ and $\sigma_2 = \sigma_{\theta+\pi/2}(P \setminus \sigma_1)$. If C is minimal, then each bounding line of C contains a point of P unless $P \subset \sigma_1$ or $P \subset \sigma_2$. It is obvious that there exists a minimum-width cross enclosing P that is minimal.

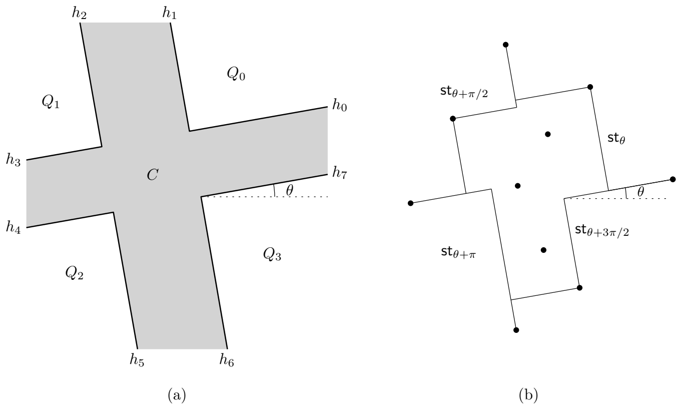
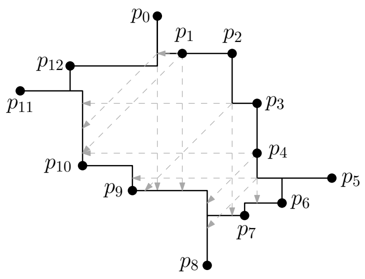


Figure 1: (a) A cross C of orientation θ and its four quadrants Q_0, Q_1, Q_2, and Q_3. Each quadrant Q_i is bounded by two half-lines h_{2i} and h_{2i+1}. (b) Illustration of OH_θ. It consists of four staircases: st_θ, $\mathsf{st}_{\theta+\pi/2}$, $\mathsf{st}_{\theta+\pi}$, and $\mathsf{st}_{\theta+3\pi/2}$.

Let K_p for $p \in \mathbb{R}^2$ be the quadrant with apex p bounded by two rays emanating from p, one horizontally to the left and the other vertically downwards. Consider the union of K_p over all $p \in P$, and the polygonal chain from the topmost point of P to the rightmost one of P along the boundary of the union of quadrants. This rectilinear chain, consisting only of horizontal and vertical segments, is called the *staircase* of P. Note that the staircase can be degenerated to a single point $p \in P$, when $x(p) > x(q)$ and $y(p) > y(q)$ for all other points q in P. Let st_θ for $\theta \in [0, 2\pi)$ denote the staircase of P with the axes rotated by θ in the counterclockwise direction.

For any $\theta \in [0, 2\pi)$, consider the staircase st_θ in orientation θ and let $p_0, \dots, p_k \in \mathsf{st}_\theta$ be the points of P lying on st_θ in this order. Note that p_0 is the topmost point of P and p_k is the rightmost point of P with the axes rotated by θ. A portion of rectilinear chain st_θ between two consecutive points p_{i-1} and p_i is called a *step*. In general, the i-th step between p_{i-1} and p_i consists of the vertical segment from p_{i-1} down to the point v_i with coordinates $x(v_i) = x(p_{i-1})$ and $y(v_i) = y(p_i)$ and the horizontal segment between v_i and p_i. Note that such a step can be degenerated to a segment, either horizontal or vertical, when p_{i-1} and p_i have the same x- or y-coordinate. We call v_i the *vertex* of the i-th step.

For each $\theta \in [0, \pi/2)$, the four staircases st_θ, $\mathsf{st}_{\theta+\pi/2}$, $\mathsf{st}_{\theta+\pi}$, and $\mathsf{st}_{\theta+3\pi/2}$ form the well-known *orthogonal convex hull* of P with the axes rotated by θ, denoted by OH_θ, and the set of points $p \in P$ lying on the four staircases is called the *maxima* of P in orientation θ. See Figure 1(b). By a slight abuse of notation, the staircase st_θ is also considered as a sequence of points $p \in P$ lying on it, and we also mean by OH_θ the union of the four staircases $\mathsf{st}_{\theta+i\pi/2}$ for $i \in \{0, 1, 2, 3\}$.



Step	p_u^1	p_u^2	p_u^3
p_0p_1	$p_{12}p_0$	$p_{10}p_{11}$	–
p_1p_2	$p_{12}p_0$	$p_{10}p_{11}$	–
p_2p_3	–	p_8p_9	p_7p_8
p_3p_4	–	p_8p_9	p_6p_7
p_4p_5	–	p_8p_9	p_6p_7

Figure 2: The orthogonal convex hull of the 13 points $p_0, ..., p_{12}$. For each step p_ip_{i+1} of st_0 where $\widehat{u}$ is p_ip_{i+1}, the gray arrows indicate the targets of p_u^1, p_u^2, and p_u^3. The information we maintain for each step of st_0 is summarized in the table on the right.

The following observation is immediate.

Lemma 1 *For a cross C of orientation θ that encloses P, any point of P lying on the boundary of $Q_i(C)$ lies on $\mathsf{st}_{\theta+i\pi/2}$ for $i \in \{0, 1, 2, 3\}$.*

This suggests a natural approach to tackle the problem by maintaining OH_θ while θ increases from 0 to $\pi/2$.

Description of data structures. Now, we introduce our data structures that maintain OH_θ over $\theta \in [0, \pi/2)$ and also that supports a specific type of queries for our later purpose. More precisely, let $\mathcal{T}_0, \ldots, \mathcal{T}_3$ be four balanced binary search trees such that $\mathcal{T}_i$ stores the steps of $\mathsf{st}_{\theta+i\pi/2}$ at its leaf nodes in the order along the staircase $\mathsf{st}_{\theta+i\pi/2}$. We add internal pointers between two consecutive leaf nodes so that we can visit the leaf node preceding or succeeding a given leaf node of $\mathcal{T}_i$ in $O(1)$ time. We denote by $\widehat{u}$ the step stored at a leaf node u.

Additionally, we maintain three external pointers p_u^j for $j \in \{1, 2, 3\}$ to a leaf node of $\mathcal{T}_{i+j}$ (the index of the trees is taken by modulo 4). Precisely, suppose the current $\mathcal{T}_i$'s represent OH_θ for $\theta \in [0, \pi/2)$. Let u be a leaf node of $\mathcal{T}_i$ and v be the vertex of the step $\widehat{u}$ stored at u. We then store three external pointers at node u.

1. p_u^1 points to the leaf node u_1 of $\mathcal{T}_{i+1}$ such that the ray from the vertex v of $\widehat{u}$ in direction $\theta + i\pi/2 + \pi$ hits $\widehat{u_1}$. If there is no such step (and no such leaf node u_1), then p_u^1 points to null.
2. p_u^2 points to the leaf node u_2 of $\mathcal{T}_{i+2}$ such that the ray from v in direction $\theta + i\pi/2 + 5\pi/4$ hits $\widehat{u_2}$. If there is no such step (and no such leaf node u_2), then p_u^2 points to null.
3. p_u^3 points to the leaf node u_3 of $\mathcal{T}_{i+3}$ such that the ray from v in direction $\theta + i\pi/2 + 3\pi/2$ hits $\widehat{u_3}$. If there is no such step (and no such leaf node u_3), then p_u^3 points to null.

Roughly speaking, these external pointers enable us in $O(1)$ time to shoot a ray of a specific direction from the vertex of a step of one staircase to the others at the current orientation θ. See Figure 2 as an example.

In the following, we show how to maintain the four tree structures $\mathcal{T}_i$ as θ increases from 0 to $\pi/2$.

Events. As θ increases, our trees $\mathcal{T}_i$, including the pointers, undergo changes in their structures and values storing in them. We distinguish such changes into two types of events:

- *Hull events* occur when the combinatorial structure of OH_θ changes. There are again two different types of changes: when a point is added to or removed from OH_θ or when an overlap between two opposite staircases is created or destroyed. Note that two opposite staircases st_θ and $\mathsf{st}_{\theta+\pi}$ may overlap and hence OH_θ introduce a self-crossing.
- *Pointer events* correspond to changes of pointers p_u^j. Such an event occurs exactly when the corresponding ray of p_u^j for some leaf node u of $\mathcal{T}_i$ and $j \in \{1, 2, 3\}$ hits a point on OH_θ, and thus its destination needs to be updated. A pointer event is associated with its occurring orientation $\phi \in [0, \pi/2)$, the corresponding leaf node u and pointer p_u^j.

From previous work on maintaining OH_θ, we can precompute all hull events.

Lemma 2 ([7, 9, 13]) *There are at most $O(n)$ hull events and all hull events can be computed in $O(n \log n)$ time using $O(n)$ space.*

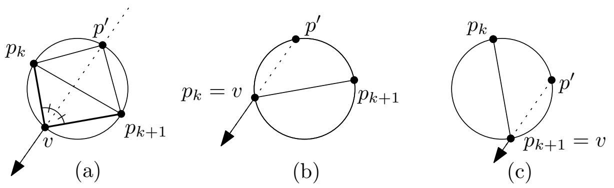


Figure 3: The line ℓ containing the ray from v in direction $\theta + \pi/4$ always goes through v regardless of θ. (a) $v \neq p_k$ and $v \neq p_{k+1}$, (b) $v = p_k$, (c) $v = p_{k+1}$.

Predicting pointer events can be done as follows. Suppose the tree structures $\mathcal{T}_i$ together with the pointers are correctly maintained until $\theta = \theta_0 \in [0, \pi/2)$. Fix a leaf node u of $\mathcal{T}_i$. For each $j \in \{1, 2, 3\}$, let u_j be the leaf node of $\mathcal{T}_{i+j}$ such that p_u^j points to u_j. At this moment, we can compute the possible pointer event corresponding to the next change of p_u^j after θ_0, which we will call the *candidate* event for p_u^j, denoted by $\mathrm{PEC}\textsc{and}_j(u, \theta_0)$. Below, we assume $i = 0$ without loss of generality, as the other cases $i = 1, 2, 3$ can be handled symmetrically.

1. For p_u^1, its corresponding ray γ hitting $\widehat{u_1}$ emanates from the vertex v of $\widehat{u}$ in direction $\theta_0 + \pi$, unless it points to null. Though the vertex v moves as θ

varies, observe that the line extending the corresponding ray γ from v always goes through one of the two endpoints of $\widehat{u}$, say $p \in P$. Hence, we can precompute when the ray γ hits one of the endpoints of $\widehat{u_1}$ and which leaf node neighboring u_1 will be the next target of p_u^1, unless there is a hull event beforehand. This constitutes the candidate event for p_u^1 after θ_0, that is, $\mathrm{PECAND}_1(u, \theta_0)$.

2. For p_u^2, its corresponding ray γ hitting $\widehat{u_2}$ emanates from the vertex v of $\widehat{u}$ in direction $\theta_0 + 5\pi/4$, unless it points to null. Consider the two endpoints p_1 and p_2 of $\widehat{u}$ and the circle O with diameter p_1p_2. Let p' be the midpoint of the circular arc between p_1 and p_2 that avoids v. Let ℓ be the directed line extending γ. As θ increases, γ and ℓ move. In this case, we observe that ℓ rotate around a fixed point p' as well: Since the step $\widehat{u}$ makes the right angle at v, v moves along the circle O as θ varies. Now observe that the line ℓ bisects the right angle at v, since ℓ is in direction $\theta + 5\pi/4$. So, ℓ always goes through p' by the inscribed angles of circles. See Figure 3. Hence, we can compute the candidate event $\mathrm{PECAND}_2(u, \theta_0)$ for p_u^2 after θ_0.
3. The case of p_u^3 is similar to that of p_u^1.

As described above, we can compute in $O(1)$ time the candidate pointer event $\mathrm{PECAND}_j(u, \theta_0)$ for each pointer p_u^j at a specific moment θ_0.

Our algorithm to maintain the trees $\mathcal{T}_i$ is run by properly handling events. As a preprocessing, we compute hull events by Lemma 2 and insert them into a priority queue, called the *event queue*. We then initialize the $\mathcal{T}_i$'s for $\theta = 0$ as follows:

Initialization. First, we compute the four staircases st_0, $\mathsf{st}_{\pi/2}$, st_π, and $\mathsf{st}_{3\pi/2}$, and so OH_0. We then build $\mathcal{T}_i$ for all $i \in \{1, 2, 3, 4\}$ with the internal pointers. Observe that when traversing the leaf nodes u of $\mathcal{T}_i$ along the internal pointers, the leaf nodes in $\mathcal{T}_{i+j}$ pointed by p_u^j also change along the internal pointers of $\mathcal{T}_{i+j}$ for $j \in \{1, 2, 3\}$. Thus, all external pointers can be computed in $O(n)$ additional time. This takes $O(n \log n)$ time using $O(n)$ space in total.

Next, we compute the candidate pointer event $\mathrm{PECAND}_j(u, 0)$ for each external pointer p_u^j, and insert it into the event queue. Since there are $O(n)$ leaf nodes in total, this can be done in additional $O(n \log n)$ time. Note that the number of (candidate) events stored in the event queue is bounded also by $O(n)$.

Handling the next event. Our algorithm then extract the next event from the event queue and handle it correctly to maintain the invariants. Let ϕ be the orientation of the next event extracted from the event queue.

If it is a pointer event $\mathrm{PECAND}_j(u, \theta_0)$, we perform the following. Note that the event was a candidate event for pointer p_u^j at the moment $\theta = \theta_0$. This pointer event is associated with the next target leaf that p_u^j should point after ϕ. Hence, updating p_u^j can be done in $O(1)$ time with the access to the node u. Then, we compute a new candidate event $\mathrm{PECAND}_j(u, \phi)$ for pointer p_u^j after ϕ as described above, and insert it into the event queue. This completes the procedure to handle a pointer event, spending $O(\log n)$ time.

For the total number of pointer events handled, we prove the following.

Lemma 3 *The total number of pointer events created and handled during the execution of the algorithm is bounded by $O(n^2)$.*

Therefore, all pointer-event-related operations can be done in $O(n^2 \log n)$ time.

Next, consider the case of a hull event. By Lemma 2, there are only $O(n)$ hull events. So, we can handle each hull event in a brute force way: reinitialize the structure for $\theta = \phi$ as described above. This spends $O(n \log n)$ time per hull event, and so $O(n^2 \log n)$ time in total.

Therefore, we conclude the following.

Lemma 4 *As θ increases from 0 to $\pi/2$, we can maintain the tree structures $\mathcal{T}_i$ for $0 \le i \le 3$ with the pointers in total $O(n^2 \log n)$ time using $O(n)$ space.*

Querying crosses. While the trees $\mathcal{T}_i$ are being maintained, we can test whether a given cross encloses P.

Lemma 5 *At any moment $\theta \in [0, \pi/2)$, while maintaining the $\mathcal{T}_i$'s as above, we can decide if any given cross of orientation θ encloses P in $O(\log n)$ time.*

4 Algorithm

Now we present an algorithm to compute a minimum-width minimal cross. First, we observe the following.

Lemma 6 *Let $C = \sigma_1 \cup \sigma_2$ be a minimum-width minimal cross. Exactly one of the following is true:*

- $\mathrm{w}(\sigma_1) > \mathrm{w}(\sigma_2)$ *and one of the bounding lines of σ_1 contains two points of P on the boundary of C.*
- $\mathrm{w}(\sigma_1) < \mathrm{w}(\sigma_2)$ *and one of the bounding lines of σ_2 contains two points of P on the boundary of C.*
- $\mathrm{w}(\sigma_1) = \mathrm{w}(\sigma_2)$.

Using Lemma 6, we present two different algorithms for computing a minimum-width cross, depending on whether one of the bounding lines of the cross contains two points of P on the boundary of the cross, or each of the bounding lines of the cross contains exactly one point of P on the boundary of the cross. Both of the algorithms work by increasing θ from 0 to $\pi/2$ and maintaining the trees $\mathcal{T}_i$.

4.1 When one of the bounding lines contains two points

In this section, we present an algorithm that computes a minimum-width cross when one of its bounding line contains two points of P on its boundary. Let $C = \sigma_1 \cup \sigma_2$ be the cross of orientation $\theta \in [0, \pi/2)$ such that the orientation of σ_1 is θ. We assume that the upper bounding line of σ_1 contains two points of P on the boundary of C and $\mathrm{w}(\sigma_1) \geq \mathrm{w}(\sigma_2)$, since the other cases can be handled symmetrically. By Lemma 1, both of the points on the bounding line appear on OH_θ.

Consider two points p_1 and p_2 of P such that both of the points are on OH_θ and the line ℓ going though p_1 and p_2 is of orientation θ for some $\theta \in [0, \pi/2)$. Let $C(p_1, p_2) = \sigma_1 \cup \sigma_2$ be the minimal cross with minimum width of orientation θ such that the upper bounding line of σ_1 is ℓ and $\mathrm{w}(\sigma_1) \geq \mathrm{w}(\sigma_2)$. Note that there is a point on the lower bounding line of σ_1 that appears in OH_θ by Lemma 1. We can compute $C(p_1, p_2)$ with our data structures at θ as follows. Let $q_1, \ldots, q_k$ be the points of P on OH_θ lying below ℓ, ordered in the decreasing order of y-coordinate in the coordinate system rotated by θ. For each index $1 \leq j \leq k$, we define $\sigma_1(j) = \sigma_\theta(\{p_1, q_j\})$ and $\sigma_2(j) = \sigma_{\theta+\pi/2}(P \setminus \sigma_1(j))$. Observe that as j increases, $\mathrm{w}(\sigma_1(j))$ increases while $\mathrm{w}(\sigma_2(j))$ is non-increasing. Therefore, by finding the smallest index j such that $\mathrm{w}(\sigma_1(j)) \geq \mathrm{w}(\sigma_2(j))$, we obtain $C(p_1, p_2) = \sigma_1(j) \cup \sigma_2(j)$.

For a fixed q_j, $\mathrm{w}(\sigma_1(j))$ can be computed trivially in $O(1)$ time. To compute $\mathrm{w}(\sigma_2(j))$, we must find the point q_M with the maximum x-coordinate and the point q_m with the minimum x-coordinate in $P \setminus \sigma_1(j)$, in the coordinate system rotated counterclockwise by θ. Let p_M be the point in P with the maximum x-coordinate. If $\sigma_1(j)$ does not contain p_M, then $q_M = p_M$. Otherwise, the value $x(q_M)$ is determined by the intersection $\mathsf{OH}_\theta \cap \ell$ or $\mathsf{OH}_\theta \cap \ell'$, where ℓ' is the line of orientation θ passing through q_j. The intersection $\mathsf{OH}_\theta \cap \ell$ can be found in $O(1)$ time via the step containing p_1 and the external pointer of the node storing that step. Likewise, the intersection $\mathsf{OH}_\theta \cap \ell'$ can be found in $O(1)$ time. Thus, $x(q_M)$ can be computed in $O(1)$ time, and by a similar argument, $x(q_m)$ can also be computed in $O(1)$ time. Hence, $\mathrm{w}(\sigma_2(j))$ can be computed in $O(1)$ time. We observe that the steps containing $q_1, \ldots, q_k$ are stored in the trees $\mathcal{T}_i$ for $i \in 0, 1, 2, 3$ at orientation θ, ordered in their y-coordinates. Therefore, by applying the binary search on the trees $\mathcal{T}_i$, we can compute $C(p_1, p_2)$ in $O(\log n)$ time.

As θ increases from 0 to $\pi/2$, the orientation when two points p_1 and p_2 of P on OH_θ lie on a line of orientation θ corresponds to either a hull event or a pointer event. More specifically, if p_1 and p_2 belong to the same staircase, the event is a hull event; otherwise, it is a pointer event. The total number of tree and pointer events is $O(n^2)$, and for each event, the corresponding cross $C(p_1, p_2)$ can be computed in $O(\log n)$ time. No additional data structures are required beyond the trees $\mathcal{T}_i$. Thus, by Lemma 4, we can conclude the following.

Lemma 7 *We can compute a minimum-width cross with one of the bounding line containing two points of P on the boundary of cross in $O(n^2 \log n)$ time and $O(n)$ space.*

4.2 When each of the bounding lines contains exactly one point

In this section, we present an algorithm that computes a minimum-width cross such that each of the bounding lines contains exactly one point of P on the boundary of the cross. Then it consists of the strips of the same width by Lemma 6. Let $C_\theta(p, q, r, s)$ be the cross of orientation $\theta \in [0, \pi/2)$ such that

- p and q lie on the upper and lower bounding line of the strip of orientation θ, respectively, and
- r and s lie on the left and right bounding line of the strip of orientation $\theta + \pi/2$

for $p, q, r, s \in P$. By definition, $C_\theta(p, q, r, s) = \sigma_\theta(\{p, q\}) \cup \sigma_{\theta+\pi/2}(\{r, s\})$. Since $\mathrm{w}(\sigma_\theta(\{p, q\}))$ and $\mathrm{w}(\sigma_{\theta+\pi/2}(\{r, s\}))$ are both sinusoidal functions of θ, they are equal for at most one or for all $\theta \in [0, \pi/2)$.

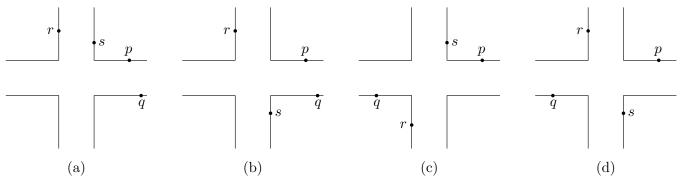


Figure 4: Four types of cross: (a) type 1, (b) type 2, (c) type 3, and (d) type 4.

Consider a cross $C = C_\theta(p, q, r, s)$ such that p, q, r, and s are the only points of P lying on the boundary of C. Then we can classify C into following four types under the symmetry group. See Figure 4.

1. $p \in h_0(C)$, $q \in h_7(C)$, $r \in h_2(C)$, and $s \in h_1(C)$.
2. $p \in h_0(C)$, $q \in h_7(C)$, $r \in h_2(C)$, and $s \in h_6(C)$.
3. $p \in h_0(C)$, $q \in h_4(C)$, $r \in h_5(C)$, and $s \in h_1(C)$.
4. $p \in h_0(C)$, $q \in h_4(C)$, $r \in h_2(C)$, and $s \in h_6(C)$.

There can be $2^4 = O(1)$ different configurations with respect to the position of p, q, r, and s on the boundary of C, but since each of them can be transformed into one of the following four types under the rotation and reflection, we present an algorithm computing a minimum-width cross belonging to one of the types above. Other case can be handled symmetrically.

Lemma 8 *There is no minimum-width cross of type 4.*

By Lemma 8, we only have to consider the crosses of type 1, 2, and 3. By Lemma 1, if a cross C of orientation θ and type $i \in \{1,2,3\}$ encloses P, then the point of P lying on $h_0(C)$ is on st_θ.

For each point $p \in P$ on st_θ and $i \in \{1,2,3\}$, we define $C^i_\theta(p)$ of orientation θ and type i such that p is the point of P on $h_0(C^i_\theta(p))$ which may be a minimum-width cross as follows.

Definition of $C^1_\theta(p)$. If p is the leftmost point of st_θ, $C^1_\theta(p)$ is not defined. Otherwise, let $s^1_\theta(p)$ be the point of P on st_θ making the step together with p on the left side of p. Let u be the leaf node of $\mathcal{T}_0$ such that $\widehat{u}$ is the step. We set $r^1_\theta(p)$ to be the point incident to the step pointed by p^1_u on the upper side. If p^1_u points to null, we set $r^1_\theta(u)$ to be the point of P at $\mathsf{st}_{\theta+\pi/2} \cap \mathsf{st}_{\theta+\pi}$. We also set $q^1_\theta(p)$ to be the point incident to the step pointed by p^3_u on the right side. If p^3_u points to null, we set $q^1_\theta(p)$ to be the point of P at $\mathsf{st}_{\theta+\pi} \cap \mathsf{st}_{\theta+3\pi/2}$. Then $C^1_\theta(p) := C_\theta(p, q^1_\theta(p), r^1_\theta(p), s^1_\theta(p))$.

Definition of $C^2_\theta(p)$. We first give following observation on a minimum-width cross of type 2.

Lemma 9 *If $C = C_\theta(p', q', r', s')$ is a minimum-width cross of orientation θ and type 2, then q' has the larger x-coordinate than p' in the coordinate system rotated by θ in counterclockwise direction.*

Note that if $C_\theta(p', q', r', s')$ is a cross of orientation θ and type 2 enclosing P, q' and s' makes a step in $\mathsf{st}_{\theta+3\pi/2}$. Moreover, Lemma 9 implies that the ray from p' in direction $\theta + 3\pi/2$ hits $\mathsf{st}_{\theta+3\pi/2}$ at the step of q' and s'.

Now we define $C^2_\theta(p)$. Let u and u' be the steps of st_θ incident to p on the left and right, respectively. Then we set $r^2_\theta(p)$ to be the point incident to the step pointed by p^1_u on the upper side, If p^1_u points to null, we set $r^2_\theta(u)$ to be the point of P at $\mathsf{st}_{\theta+\pi/2} \cap \mathsf{st}_{\theta+\pi}$. We set $q^2_\theta(p)$ and $s^2_\theta(p)$ to be the points incident to the step pointed by $\mathsf{p}^3_{u'}$ on the upper and lower side, respectively. If $\mathsf{p}^3_{u'}$ points to null, $C^2_\theta(u)$ is not defined. Otherwise, $C^2_\theta(p) := C_\theta(p, q^2_\theta(p), r^2_\theta(p), s^2_\theta(p))$.

Definition of $C^3_\theta(p)$. Assume that there is a cross $C = C_\theta(p', q', r', s')$ which is a minimum-width cross of orientation θ and type 3. Then the step of s' and p' is present at st_θ, and the step of r' and q' is present at $\mathsf{st}_{\theta+\pi}$. Let v be the vertex of the step of s' and p', and v' be the vertex of the step of r' and q'. Then v is also the vertex of $Q_0(C)$ and v' is also the vertex of $Q_2(C)$. Since C consists of strips of the same width, the line going through v and v' is of orientation $\theta + \pi/4$.

Now, we define $C^3_\theta(p)$. If p is the leftmost point of st_θ, $C^3_\theta(p)$ is not defined. Otherwise, let $s^3_\theta(p)$ be the point of P on st_θ making the step together with p on the left side of p. Let u be the leaf node of $\mathcal{T}_0$ such that $\widehat{u}$ is the step. Then we set $q^3_\theta(p)$ and $r^3_\theta(p)$ to be the points of P incident to the step pointed by p^2_u on the right and left, respectively. If p^2_u points to null, $C^3_\theta(p)$ is not defined. Otherwise, $C^3_\theta(p) := C_\theta(p, q^3_\theta(p), r^3_\theta(p), s^3_\theta(p))$.

Algorithm. As θ increases, we handle *cross events* to find a minimum-width cross. Cross events occur at θ when two strips of $C^i_\theta(p)$ for some p on st_θ and $i \in \{1,2,3\}$. The detailed algorithm is as follows.

When $\theta = 0$ and whenever a hull event occurs, we compute $q^i_\theta(p)$, $r^i_\theta(p)$, and $s^i_\theta(p)$ for all p on st_θ and $i \in \{1,2,3\}$ as an initialization. We define $\theta^i(p)$ to be the orientation in $[0, \pi/2)$ such that the widths of the cross $C^i_\theta(p)$ are equalized at $\theta = \theta^i(p)$ for $p \in P$ on st_θ and $i \in \{1,2,3\}$. If $C^i_\theta(p)$ is not defined, no such orientation exists, or the widths are the same for all orientation in $[0, \pi/2)$, $\theta^i(p)$ is not defined. We maintain all the $\theta^i(p)$ values that is larger than current θ into a *cross event queue* in the form of priority queue.

When θ reaches $\theta^i(p)$ for some p and i, we handle the cross event. In the cross event, we test whether $C^i_\theta(p)$ encloses P. This can be done in $O(\log n)$ time by Lemma 5. If it encloses P, we consider $C^i_\theta(p)$ as a candidate of a minimum-width cross. By reporting a candidate cross with the minimum width, the algorithm correctly computes a minimum-width cross.

For each hull event, computing $q^i_\theta(p)$, $r^i_\theta(p)$, and $s^i_\theta(p)$ for $i \in \{1,2,3\}$ takes $O(1)$ time for each p. Then we compute the cross event queue of $O(n)$ values accordingly, which takes $O(n \log n)$ time. Thus, it takes $O(n^2 \log n)$ time in total for hull events.

For each pointer event, let u be the leaf node in $\mathcal{T}_0$ such that its external pointer is updated in the pointer event, and let $p_1, p_2 \in P$ be the points making the step $\widehat{u}$. Then $q^i_\theta(p)$, $r^i_\theta(p)$, and $s^i_\theta(p)$ for $p \in \{p_1, p_2\}$ and $i \in \{1,2,3\}$ might be changed. Then we update $\theta^i(p)$ values accordingly, and update the cross event queue. Since we only update $O(1)$ values at once, it takes $O(\log n)$ time for each pointer event. Therefore, it takes $O(n^2 \log n)$ time in total for pointer events.

Since $O(n)$ new $\theta^i(p)$ values are updated for each hull event, and $O(1)$ new values are updated for each pointer event, there $O(n^2)$ cross events occur in total. Moreover, since we store $O(1)$ values for each $p \in P$ on st_θ in the cross event queue, it uses $O(n)$ space. Therefore, together with Lemma 4, we have the following.

Lemma 10 *We can compute a minimum-width cross with each of the bounding line containing exactly one point of P on the boundary of cross in $O(n^2 \log n)$ time and $O(n)$ space.*

By Lemma 7 and 10, we conclude the following.

Theorem 11 *For a set P of n points in $\mathbb{R}^2$, we can compute a minimum-width cross in $O(n^2 \log n)$ time and $O(n)$ space.*

References

[1] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. A (1+ ε)-approximation algorithm for 2-line-center. *Computational Geometry*, 26(2):119–128, 2003.

[2] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. Approximation algorithms for a k-line center. *Algorithmica*, 42:221–230, 2005.

[3] P. K. Agarwal and M. Sharir. Planar geometric location problems and maintaining the width of a planar set. In *Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1991)*, pages 449–458. SIAM, 1991.

[4] P. K. Agarwal and M. Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete Comput Geom*, 16:317–337, 1996.

[5] T. Ahn and S. W. Bae. Constrained two-line center problems. In *International Symposium on Algorithms and Computation*, pages 5:1–5:17, 2024.

[6] T. Ahn, C. Chung, H.-K. Ahn, S. W. Bae, O. Cheong, and S. D. Yoon. Minimum-width double-slabs and widest empty slabs in high dimensions. *Computational Geometry*, 129:102173, 2025.

[7] C. Alegría-Galicia, D. Orden, C. Seara, and J. Urrutia. On the $\mathcal{O}_\beta$-hull of a planar point set. *Computational Geometry*, 68:277–291, 2018. Special Issue in Memory of Ferran Hurtado.

[8] S. W. Bae. Minimum-width double-strip and parallelogram annulus. *Theoretical Computer Science*, 833:133–146, 2020.

[9] S. W. Bae, C. Lee, H.-K. Ahn, S. Choi, and K.-Y. Chwa. Computing minimum-area rectilinear convex hull and L-shape. *Computational Geometry*, 42(9):903–912, 2009.

[10] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *International Journal of Computational Geometry and Applications*, 12(1-2):67–85, 2002.

[11] C. Chung, T. Ahn, S. W. Bae, and H.-K. Ahn. Parallel line centers with guaranteed separation. *Computational Geometry*, 129:102185, 2025.

[12] A. K. Das, S. Das, and J. Mukherjee. Approximation algorithms for orthogonal line centers. *Discrete Applied Mathematics*, 338:69–76, 2023.

[13] J. M. Díaz-Báñez, M. A. López, M. Mora, C. Seara, and I. Ventura. Fitting a two-joint orthogonal chain to a point set. *Computational Geometry*, 44(3):135–147, 2011.

[14] A. Glozman, K. Kedem, and G. Shpitalnik. On some geometric selection and optimization problems via sorted matrices. *Computational Geometry: Theory and Applications*, 11(1):17 – 28, 1998.

[15] J. W. Jaromczyk and M. Kowaluk. The two-line center problem from a polar view: A new algorithm and data structure. In *Workshop on Algorithms and Data Structures*, pages 13–25. Springer, 1995.

[16] M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997.

[17] N. Megiddo and A. Tamir. On the complexity of locating linear facilities in the plane. *Operations research letters*, 1(5):194–197, 1982.

[18] G. T. Toussaint. Solving geometric problems with the rotating calipers. In *Proceedings of the 2nd Mediterranean Electrotechnical Conference (IEEE Melecon 1983)*, volume 83, page A10, 1983.

Covert Computation in the Prebuilt aTAM[*]

Timothy Gomez[†] Robert Schweller[‡] Tim Wylie[‡]

Abstract

We prove several results related to the concept of hidden computation in the most well-known model of self-assembly, the Abstract Tile-Assembly Model (aTAM). Previous work showed that the aTAM, with negative glues and no detachment, or in 3D, is capable of covert computation. Without negative glues, the aTAM is still capable of covert computation, but it only seems possible with exponential-sized output assemblies. Here, we show that with a constant number of constant-sized prebuilt assemblies, covert computation is possible. Further, we use this to show that the fundamental self-assembly problem of Unique Assembly Verification (UAV) in the prebuilt aTAM is coNP-complete, which is in contrast with UAV in the standard aTAM, which is polynomial. Finally, we look at the polyTAM and prove that Producibility is in P and UAV is FPT with respect to the size of the polytiles in the tile set.

1 Introduction

With the ability to manufacture nanoscale structures and to use DNA as building blocks for structures [24] or for data storage [11], there has been a great increase in the need to process and compute information at the same level. Thus, the study of self-assembling computation has been an important and active area of research over the last two decades.

Designing self-assembling systems that compute functions is an active and well-studied area of computational geometry and biology [5, 19]. The ability to craft monomers capable of placing themselves, especially when doing precision construction and computation at scales where conventional tools are incapable of operating, (e.g., the nanoscale) has tremendous power. One of the few downsides to self-assembly computation is that the entire history of the computation is visible. In certain cases, this may be undesirable for privacy or security reasons, which we motivate below. Thus, we build on recent work [3, 7, 8, 10] to explore *covert computation*, where we build Tile Assembly Computers (TACs) designed to obtain the output of computation while obscuring the inputs and computational history. We do this by proving that covert computation is possible in one of the simplest standard models of self-assembly (the Abstract Tile-Assembly Model [26]) if only a constant number of prebuilt assemblies are allowed in the tile set.

1.1 Previous Work

The Abstract Tile-Assembly Model (aTAM) was first introduced in [26] and inherited the ability to perform Turing computation from Wang tiles [25]. Since then, investigation into the model has led in many directions, such as Intrinsic Universality [17, 20], efficient assembly of shapes [22], and parallel computation [6, 21]. Many generalizations have also appeared, such as allowing for RNA tiles that can be deleted [1, 14], multiple stages of growth [7, 13, 15], and even negative glues [10, 16]. The aTAM is powerful because not only can the tile set store information, but work has also gone into using the seed [4], or even the temperature [12, 23], for making systems more complex.

Tile Assembly Computers were defined in [6, 21], and Covert Computation, as defined in the field of self-assembly, was first introduced in 2019 [10] for negative growth-only aTAM. In *negative* variations of tile self-assembly models, tiles are capable of not only attachment to, but also detachment from, an assembly if the assembly has a cut through the bond graph less than the temperature, which might have been introduced by a negative glue. In negative *growth-only* aTAM, the system must be designed so that no tile detachment can occur, even when there may be glues providing a repellent force. The covert computation framework was created to analyze the Unique Assembly Verification (UAV) problem within that model, and showing it to be coNP-complete. Covert computation has been explored in two other models of self-assembly as well: Staged Self-Assembly [7] and Tile Automata [8].

Unique Assembly Verification and Producibility are fundamental problems in the field of self-assembly. In the standard aTAM (no negative glues), the Producibility and UAV problems are solvable in polynomial time [2]. These algorithms generalize to the 3D aTAM as well. However, covert computation *is* possible in the 3D aTAM [3] even though UAV is polynomial. With prebuilt assemblies in the aTAM, Producibility is NP-complete, and UAV is known to be coNP-complete with

[*]This research was supported in part by National Science Foundation Grant CCF-2329918.

[†]Massachusetts Institute of Technology, tagomez7@mit.edu

[‡]University of Texas Rio Grande Valley, {robert.schweller,timothy.wylie}@utrgv.edu

Model	Max Size	Amt. >1	τ	Result	Ref.
aTAM	1	0	τ	P	[2]
Neg$_{GO}$ aTAM*	1	0	2	coNP-c	[10]
3D aTAM	1	0	2	P	[2]
aTAM	38	$O(\|A\|)^\dagger$	2	coNP-c	[9]
aTAM	12	8	2	coNP-c	Thm. 1
polyTAM	$O(1)$	any	τ	FPT	Thm. 3

Table 1: Complexity results of Unique Assembly Verification in the aTAM with our result added. All other UAV results have focused on the tile set being singletons. UAV is undecidable in the negative aTAM, but coNP-complete with negative glues if the system never allows detachment (*growth only). τ is the temperature of the system. Max Size refers to the size of the attachable tiles in the tile set, and the Amt. > 1 column is the number of different polyominos greater than a singleton in the system. † This refers to the size of the assembly A that is input to UAV.

a linear number of constant-sized (38) assemblies [9]. A summary of some previous UAV results in the aTAM is shown in Table 1.

1.2 Our Contributions

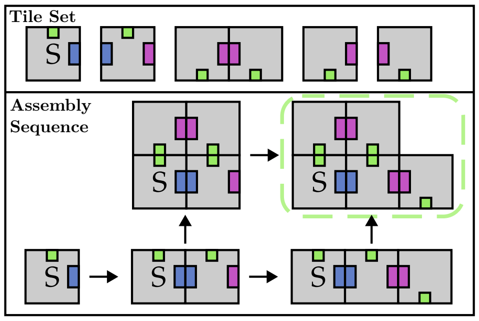
Here, we drastically improve the previous prebuilt aTAM result by proving that, not only is it capable of covert computation, but that UAV, even with only eight prebuilt assemblies of constant size (12), is still coNP-complete. This is different from the polyTAM model [18] since each prebuilt assembly must be built from individual tiles in the system. The covert construction and UAV result are covered in Section 3. Following in Section 4, we show that Producibility in the polyTAM (to show a separation from the prebuilt model) and Unique Assembly Verification are Fixed Parameter Tractable with respect to the size k of the largest polytile.

2 Definitions

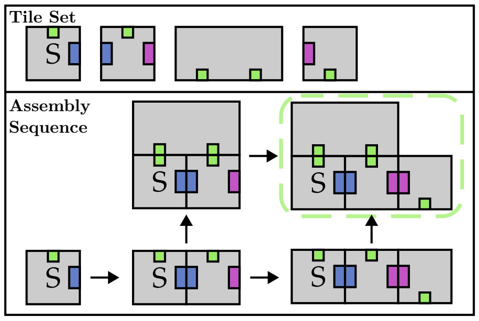
We begin with an overview of the Prebuilt Abstract Tile-Assembly Model, the Polyomino Tile Assembly Model, and then follow with a definition of Tile Assembly Computers and covert computation.

2.1 Abstract Tile Assembly Model

We provide a brief, high-level definition of the Abstract Tile-Assembly Model (aTAM) and refer the reader to [22] for a formal definition. At a high level, the Abstract Tile-Assembly Model (aTAM) uses a set of *tiles* capable of sticking together to construct shapes. These



(a) Prebuilt aTAM Example



(b) polyTAM Example

Figure 1: Example temperature-2 systems with a seed S and a 2×1 tile. The terminal assembly is outlined, matching colors are matching glue labels, and the size of the glue is the strength (1 or 2). (a) Prebuilt aTAM system. (b) polyTAM system.

tiles are typically squares (2D) or cubes (3D) with *glues* on each side where they may attach to one another. A glue is labeled to indicate its type, governing what other tiles it may bond with and the *strength* of the bond. A tile with all of its labels is a *tile type*. A *tile set* contains all the tile types of the system. A single tile may attach at a location if the combined strength of the matching glues is greater than or equal to the *temperature* τ. An *assembly* is a shape made up of one or more combined tiles. Construction is started around a designated *seed* assembly S. Any assembly capable of being made from the seed is called a *producible* assembly. An assembly is *terminal* if no more tiles can attach. A terminal assembly is said to be *uniquely produced* if it is the only terminal assembly that can be made by a tile system. A tile system is formally represented as an ordered triplet $\Gamma = (T, s, \tau)$ of the tile set, seed assembly, and temperature parameter, respectively.

2.2 Prebuilt aTAM and polyTAM

Formal definitions for the prebuilt aTAM and the polyTAM can be found in [9] and [18], respectively. Figure 1 gives a small example aTAM and polyTAM system to illustrate the main concepts and the difference in larger attaching assemblies. Briefly, the prebuilt aTAM generalizes the aTAM by allowing the tile-set to contain pre-assembled, τ-stable macro assemblies that attach to a growing seed if such attachment yields at least τ strength. Similarly, the polyTAM allows polyomino-shaped tiles in the tile set, but does not require that these polyominos be made up of individual tiles, which fundamentally affects the problems of assembly production and unique assembly production.

2.3 Covert Computation

We provide a brief, informal definition of covert computation, and refer the reader to [10, 21] for a formal definition. Briefly, a tile system computes a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^k$ if there exists a uniform method for encoding an n-bit string x into a seed such that the unique terminal assembly of the system with such seed maps to the k-bit string $f(x)$. A system that computes f is further said to *covertly* compute f if $f(x_1) = f(x_2)$ implies that the final terminal assemblies produced from inputs x_1 and x_2 are guaranteed to be identical. In other words, the assembly produced from the computation of $f(x)$ cannot be used to distinguish which input was used to create it. This includes obfuscating not only the input bit string encoded in the initial seed, but also the entire computational history of the computation. In contrast, standard aTAM Turing machine simulations explicitly encode the initial bit string and each computational step of the computation's history.

3 Covert Computation in the Prebuilt aTAM

Here, we show that covert computation is possible in the 2D aTAM with polynomial assembly size if we allow prebuilt assemblies. We then show that this implies Production is NP-complete and Unique Assembly Verification (UAV) is coNP-complete in the prebuilt aTAM via a reduction from Circuit SAT. Although both of these problem complexities were known for the general problem [9], we greatly lower the size of the prebuilt assemblies from 36 to 12, and we only need 8 different assemblies rather than a number polynomial in the input assembly size. Thus, we have shown the problems can not be parameterized on the prebuilt assembly size (they are paraNP-complete/para-coNP-complete).

The hardness result follows the same basic construction design as used in the construction from [10]. We then follow with the results for a constant number of prebuilt assemblies.

Definition 1 (Planar Circuit SAT) *Instance: A planar directed acyclic graph (DAG) $G = (V, E)$ with n boolean inputs, one output, and all gates are NAND gates (or NOR gates). Every $v \in V$ is either a NAND gate ($deg^-(v) = 2$, $deg^+(v) = 1$) or a fanout ($deg^-(v) = 1$, $deg^+(v) = 2$). The source vertices, $v_i \in V$ s.t. $deg^-(v_i) = 0$ and $1 \le i \le n$, are the variables. The sink vertex, $s \in V$ s.t. $deg^+(v_i) = 0$ is the "output" of the boolean circuit.*
Question: Does there exist a setting of the inputs such that the output to the circuit is 1?

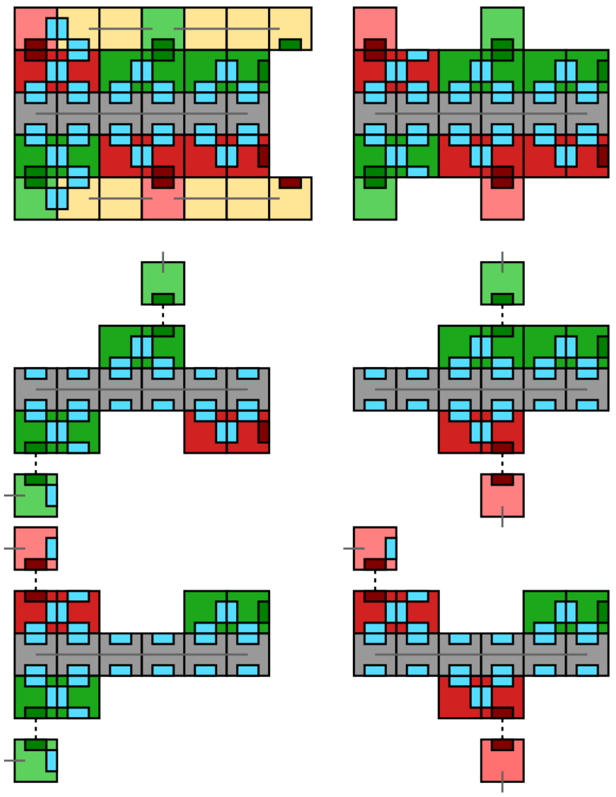
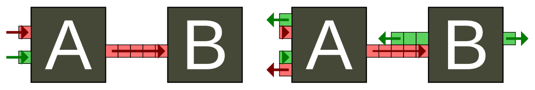


Figure 2: The NAND gate construction. The green, red, and blue glues are all strength one (system strength is 2), and the lines represent unique full-strength glues. The fully constructed gate at the end is shown on the left. The input to any of the gates will be the upper green (True) or lower red (False) tiles on the left side, which each have a strength-1 green/red glue. Depending on which two are present, there are two prebuilt assemblies that can attach: the one with two green tiles on the right, and the one with the reds. These are equivalent to outputting True or False, respectively. In order, there is the full gadget, the 2 True inputs with False output, True/False with True output, False/True with True output, and 2 False inputs with a True output.

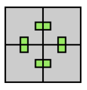
3.1 Overview

We use a dual-rail logic implementation of variables and wires to build NAND, FANOUT, and BUFFER gates. As each gadget attaches, it backfills the previously unused wires and gadgets so that the input to a gate is obscured (see Figure 3a).

The main insight for the construction is that a 4×4 block can be used in a prebuilt gadget where the assembly is stable with strength-1 glues, but it is not constructable from a seed without other tiles (see Figure 3b). Thus, only when backfilling are helping tiles there to allow other parts of the gadget to assemble. These



(a) Backfilling



(b)

Figure 3: (a) Backfilling with a NAND (gadget A)and NOT (gadget B) connected. Only when the inputs are received will the gadget backfill the other possible input value. (b) The minimum prebuilt assembly that can not be assembled from a seed in a strength-2 system.

4×4 blocks are visible in all gadgets as the cycles using strength-1 blue glues.

Variables and Wires. A single row of tiles represents a true or false signal in the dual-rail logic. These are seen as the input to the gates shown in Figures 2 and 5 where the green tiles are true and the red false.

NAND Gate. The NAND gate is shown in Figure 2. The first image shows the filled in gadget, and the next four show the different prebuilt assemblies that function as the NAND gadget. The red and green tiles on the left side are the wires attaching to the white tiles of the gadget. The two wires are both needed for the gadget to cooperatively attach, but only the prebuilt gadget with the correct output can attach. The lines attaching tiles all have strength-2 glues, and all other blue, green, and red glues have strength-1.

Note that the prebuilt assemblies can not be built with single tile attachments due to the output tiles that are cooperatively attached, but could not have been built in the aTAM. This is also why the other truth value line does not populate when the gadget attaches. The two missing tiles attached with blue glues can only be filled once the other line has backfilled to the gadget.

Finally, we note that each NAND needed in a circuit is a distinct prebuilt assembly (with the four variants) due to needing distinct glues for the inputs. If these were not unique, the circuit would not be built properly.

FANOUT Gate. Figure 5 shows the complete gadget and the two prebuilt assemblies that implement it. The output works similar to the NAND gadget where the other output tiles can only be placed by backfilling.

BUFFER Gate. For the fanout, since we need one rail to backfill before the other rail can go, we create a backfill gadget (Figure 6), that simply forces the backfill to happen behind itself. We can then place these on the output of each fanout rail. These would be needed if you had both outputs of a fanout going into the same NAND as in Figure 7.

3.2 Verification with Prebuilt Assemblies

Theorem 1 *Production and Unique Assembly Verification in the prebuilt aTAM are NP-complete and coNP-complete respectively with only 8 prebuilt assemblies of size-12 or fewer tiles.*

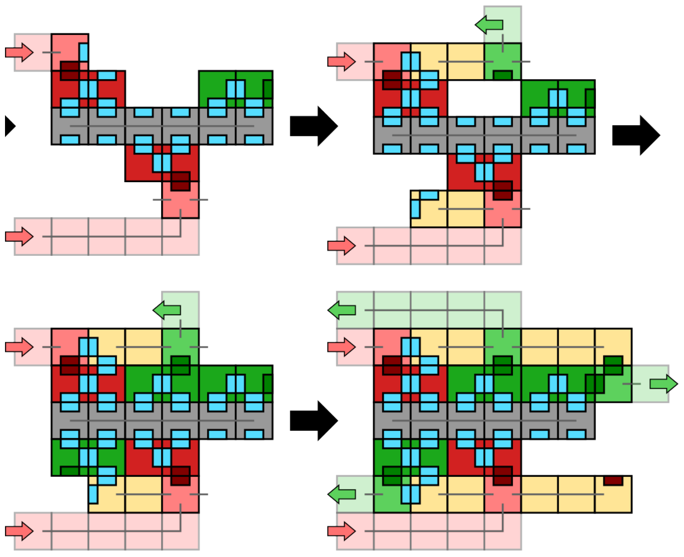


Figure 4: The NAND gadget with two false inputs. The true rails backfill, then the side tiles go around the gadget, but only the prebuilt tiles can continue the signal. The rest of the gadget completes when the false rail is backfilled from the next gadget.

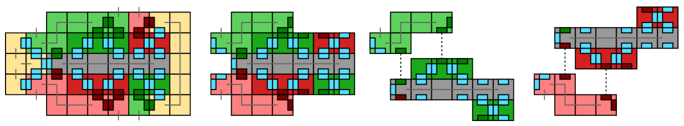


Figure 5: The FANOUT construction. Assumes that the end of the wire has a unique tile indicating a fanout. The fully filled gadget is shown on the left. The center figures shows the prebuilt True FANOUT attaching to a True line, and the end figure shows a prebuilt False FANOUT attaching to a False line.

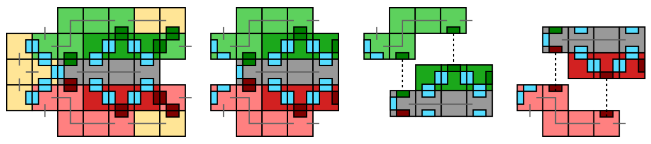


Figure 6: The backstop/buffer gadget only requires one input. This is equivalent to a buffer gate.

Proof. Given any planar Circuit SAT with a DAG made of wires, NANDs, and FANOUTs, we build a prebuilt aTAM system by converting all gates to functional NAND and FANOUT prebuilt assemblies (with unique glues) in the system. The wires are made of unique tiles that can only connect the prebuilt assemblies.

We create the seed assembly as shown in Figure 9 where either true or false prebuilt gadgets can attach to each variable. Each variable has its own version of the true and false assembly that differ by the connecting glue. Thus, all possible seeding truth values can occur.

Once each gadget places, it backfills the previous

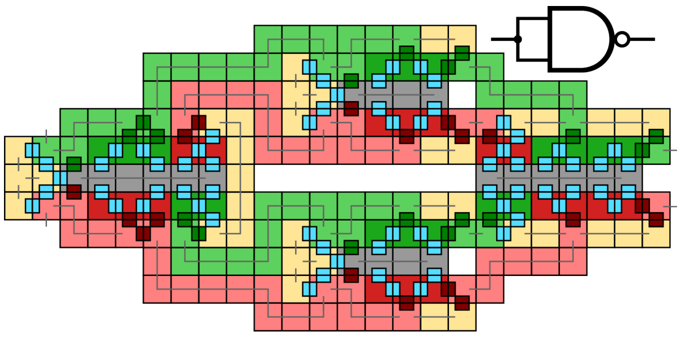


Figure 7: An example usecase where the backstop gadget is needed. Here, a fanout has both outputs going into the two inputs of a NAND. If a false signal was the input to the fanout, without the backstop to force the backfilling to assemble the other output of the fanout, the NAND could never cooperatively attach.

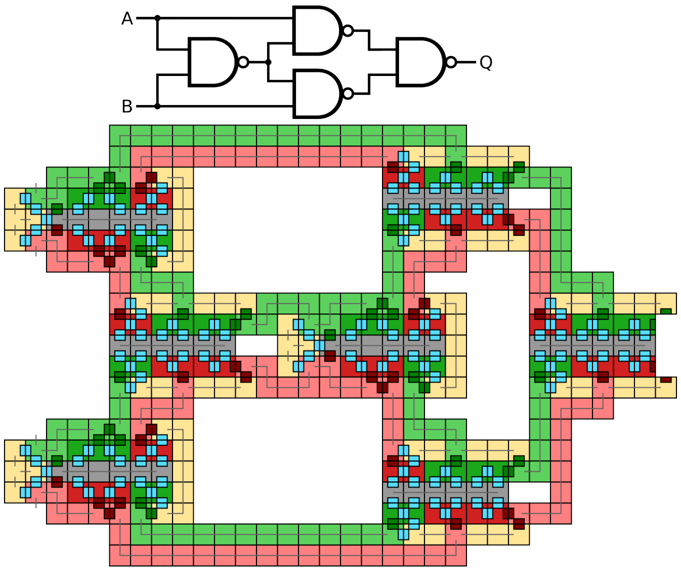


Figure 8: The XOR gate implemented with NANDs and FANOUTs and then shown as a terminal assembly built from the prebuilt assemblies and tiles. Note that in an actual build, only the true or false would be output in the final NAND gadget.

wires and gadgets. Thus, at the end, all terminal assemblies look the same unless some assignment satisfies the circuit sat, which would then have a 'True' output on the final gadget as opposed to a 'False'. □

4 PolyTAM UAV

In this section, we consider the complexity of producibility and UAV in the polyTAM. Unlike the prebuilt aTAM, the polyominos of the polyTAM are not made up of individual tiles. This changes the complexity of producibility and UAV since, in the prebuilt model, multiple distinct layouts of prebuilt assemblies could come together to form a unique underlying pattern of tiles. However, in the polyTAM, this does not happen, making the problems of producibility and UAV provably easier.

Unique Assembly Verification has not been studied within the literature for the polyTAM, however, a slightly more involved variation of the aTAM UAV algorithm (based on the size of the polyominos in the tile set) can solve the problem [2].

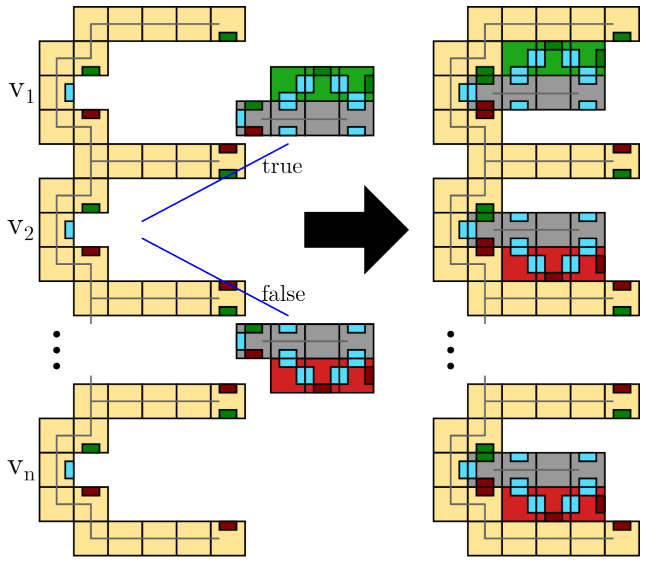


Figure 9: The input seed is a simple linear assembly with variable buffer assemblies attached. The buffer gadgets can be attached nondeterministically with the cooperative input glues. For the Circuit SAT reduction, we must allow all possible assignments. The linear assembly seed with no variables attached can be added, and all possible assignments are feasible.

Theorem 2 *Producibility in the PolyTAM is in P.*

Proof. We run the same greedy grow algorithm from [2]. For system $\Gamma = (T, S, \tau)$ and target assembly A, start with the seed assembly $A' = S$. While there exists a poly tile $t \in A$ that may attach to A', attach it. If $A' == A$, then accept, or if we reach a terminal assembly such that $A' \neq A$, then reject.

At a high level, the algorithm correctness still holds because placing a tile never prevents reaching the target and thus, order of placement does not matter. Formally, if $A' \to B$ and $A' \to C$, then $C \to B \bigcup C$. If $A' \to B$ via attaching tile p, then p may still attach to C. All the glues used by p to attach still exists in C because $A' \sqsubseteq C$. Further, the tile q used by $A' \to C$ does not overlap with p since they both exist in A. Thus, we can attach p to C and reach $A' \bigcup \{p\} \bigcup \{q\} = B \bigcup C$. □

The algorithm also generalizes the UAV algorithm for the aTAM from [2]. However, instead of only excluding a single tile in each loop, we exclude all the neighborhoods of that tile. At a high level, we must do this to make sure we avoid any possible *geometric blocking*.

Theorem 3 *Unique Assembly Verification in the PolyTAM is solvable in time $2^{O(k)}poly(|T|,|A|)$ where k is the size of the largest polytile. Thus, UAV is FPT with respect to polytile size.*

Proof. For polytile $p \in A$, let $N_A(p,i)$ be the set of polytiles q such that q is within i unit squares of p in A. We refer to a rogue assembly as any assembly R such that either (1) $R \not\sqsubseteq A$ or (2) R is a strict subassembly of A and is terminal. Thus, the existence of a producible rogue assembly is sufficient and necessary for a UAV instance to be false. While in the prebuilt aTAM we had to handle the second case in the polyTAM, we know that if the assembly is producible then no subassembly is terminal via the same argument as in producibility. Placing a tile from A never prevents reaching the target.

The algorithm is as follows: for each polytile $p \in A$ and subsets $Q \subseteq N_A(p,4k)$, consider the assembly $B = A \setminus p \bigcup Q$. Starting from the seed S, run the producibility algorithm from Theorem 2 and take the maximally produced subassembly $C \sqsubseteq B$. Check if any tile $r \neq p$ can attach to C. If any such tile can attach, then return $R = C \bigcup r$.

It remains to be proven that the returned R is a rogue assembly. If the UAV instance is false, then there exists some producible rogue assembly R'. Since it is producible there exists some $R' \rightarrow R$, we can repeat such a process until we reach some non-rogue assembly $B' \rightarrow R$. Thus, we can always assume there exists a rogue that is within 1 attachment from some subassembly $B' \sqsubseteq A$. Consider the smallest such B'. Let r be the polytile such that $R = B' \bigcup \{q\}$, which exists since $B' \rightarrow R$. Let $\beta_A(r)$ be the subset of polytiles of A that overlap with r, i.e., the tiles that block r. Let $\alpha_{B'}(r)$ be a minimum set of polytiles in B' that r uses to attach to B', i.e., the glue tiles. Now consider some other producible assembly C', as long as $C' \bigcup \alpha_{B'}(r) = \alpha_{B'}(r)$ and $C' \bigcap \beta_A(r) = \emptyset$, i.e., C' allows r to attach. Now we will find some C' in our algorithm and that attaching r to C' forms a rogue assembly. Consider some tiles $e \in A$ such that e and r overlap. Consider the neighborhood $N(e,4k)$. Note everything in $\alpha_{B'}(r)$ and $\beta_A(r)$ are included in $N(e,4k)$ as the bounding box containing both r and e is max $2k$ in both dimensions. We need to double the size of the bounding box again since $\beta_A(r)$ and $\beta_A(r)$ are within k distance of r. Thus, since we check all subsets of $N(e,4k)$, we will find a subset with everything in $\alpha_{B'}(r)$ and nothing in $\beta_A(r)$. Thus, we build some assembly C' that allows r to attach building a rogue assembly. □

5 Conclusion and Future Work

In this paper we showed the prebuilt aTAM is capable of covert computation and used that to improve the hardness result of verification problems to show they only require a constant number of tiles. Thus, in parameterized complexity terminology these problems are paraNP-complete (or para-coNP-complete) with respect to the prebuilt assembly size, which means the problem is NP-hard (coNP-hard) for some exact constant, which is 8 for these problems. This in contrast to the polyTAM where we have an FPT algorithm with respect to the polytile size. These results lead to some interesting directions for future work.

- One open problem is to improve the FPT algorithm. Can we achieve a polynomial time algorithm or is it NP-hard for super-constant sized polytiles?
- In the prebuilt aTAM we have pushed the prebuilt assembly size from 38 to 12. With a modification of the original aTAM UAV algorithm, similar to the FPT algorithm for polyTAM, it seems feasible to show that UAV with prebuilt assemblies of size ≤ 3 is polynomial since size-4 assemblies are the smallest stable assemblies that could exist that is not buildable from singletons (a cycle of 4 tiles with strength-1 glues between them in a $\tau = 2$ system). See Figure 3b. This leaves a complexity gap for prebuilt assemblies of sizes $4 - 11$.
- We only used 8 different prebuilt assemblies. What if only a single prebuilt assembly is allowed?
- Covert computation is already possible in the 3D aTAM, however, we believe that in the 3D prebuilt aTAM, if we modify the covert circuit (from [9]) with a fewer number of smaller sized prebuilt assemblies, covert computation is possible and UAV would be coNP-complete, which is in P without prebuilt assemblies.
- Although not investigated, we may also get some straightforward corollaries from the polyTAM, such as the prebuilt aTAM being capable of universal computation at temperature-1 whenever we have prebuilt assemblies of at least size 3 [18].

References

[1] Z. Abel, N. Benbernou, M. Damian, E. D. Demaine, M. L. Demaine, R. Flatland, S. D. Kominers, and R. Schweller. Shape replication through self-assembly and rnase enzymes. In *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'10, pages 1045–1064, 2010.

[2] L. M. Adleman, Q. Cheng, A. Goel, M.-D. A. Huang, D. Kempe, P. M. de Espanés, and P. W. K. Rothemund.

Combinatorial optimization problems in self-assembly. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 23–32, 2002.

[3] R. M. Alaniz, D. Caballero, T. Gomez, E. Grizzell, A. Rodriguez, R. Schweller, and T. Wylie. Covert Computation in the Abstract Tile-Assembly Model. In *Symposium on Algorithmic Foundations of Dynamic Networks*, volume 257 of *SAND'23*, pages 12:1–12:17, 2023.

[4] A. Alseth and M. J. Patitz. The need for seed (in the abstract tile assembly model). In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, SODA'23, pages 4540–4589, 2023.

[5] S. Berman, S. P. Fekete, M. J. Patitz, and C. Scheideler. Algorithmic foundations of programmable matter (dagstuhl seminar 18331). In *Dagstuhl Reports*, volume 8. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[6] Y. Brun. Arithmetic computation in the tile assembly model: Addition and multiplication. *Theoretical Comp. Sci.*, 378:17–31, 2007.

[7] D. Caballero, T. Gomez, R. Schweller, and T. Wylie. Covert computation in staged self-assembly: Verification is pspace-complete. In *Proceedings of the 29th European Symposium on Algorithms*, ESA'21, 2021.

[8] D. Caballero, T. Gomez, R. Schweller, and T. Wylie. Verification and computation in restricted tile automata. *Natural Computing*, 2021.

[9] D. Caballero, T. Gomez, R. Schweller, and T. Wylie. Complexity of verification in self-assembly with prebuilt assemblies. *Journal of Computer and System Sciences*, 136:1–16, 2023.

[10] A. A. Cantu, A. Luchsinger, R. T. Schweller, and T. Wylie. Covert computation in self-assembled circuits. *Algorithmica*, 83:531 – 552, 2019.

[11] L. Ceze, J. Nivala, and K. Strauss. Molecular digital data storage using dna. *Nature Reviews Genetics*, 20(8):456–466, 2019.

[12] C. Chalk, A. Luchsinger, R. Schweller, and T. Wylie. Self-assembly of any shape with constant tile types using high temperature. In *Proc. of the 26th Annual European Symposium on Algorithms*, ESA'18, 2018.

[13] C. T. Chalk, E. Martinez, R. T. Schweller, L. Vega, A. Winslow, and T. Wylie. Optimal staged self-assembly of general shapes. *Algorithmica*, 80(4):1383–1409, 2018.

[14] E. Demaine, M. Patitz, R. Schweller, and S. Summers. Self-assembly of arbitrary shapes using rnase enzymes: Meeting the kolmogorov bound with small scale factor. *Symposium on Theoretical Aspects of Computer Science (STACS2011)*, 9, 01 2010.

[15] E. D. Demaine, S. P. Fekete, C. Scheffer, and A. Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4 – 18, 2017. Computational Self-Assembly.

[16] D. Doty, L. Kari, and B. Masson. Negative interactions in irreversible self-assembly. *Algorithmica*, 66(1):153–172, 2013.

[17] D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, and D. Woods. The tile assembly model is intrinsically universal. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 302–310. IEEE, 2012.

[18] S. P. Fekete, J. Hendricks, M. J. Patitz, T. A. Rogers, and R. T. Schweller. Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly. In *Proceedings of the 2015 Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'15, pages 148–167, 2015.

[19] P. W. Frederix, I. Patmanidis, and S. J. Marrink. Molecular simulations of self-assembling bio-inspired supramolecular systems and their connection to experiments. *Chemical Society Reviews*, 47(10):3470–3489, 2018.

[20] D. Hader, A. Koch, M. J. Patitz, and M. Sharp. The impacts of dimensionality, diffusion, and directedness on intrinsic universality in the abstract tile assembly model. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2607–2624. SIAM, 2020.

[21] A. Keenan, R. Schweller, M. Sherman, and X. Zhong. Fast arithmetic in algorithmic self-assembly. *Natural Computing*, 15(1):115–128, Mar 2016.

[22] P. W. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468, 2000.

[23] R. Schweller, A. Winslow, and T. Wylie. Complexities for high-temperature two-handed tile self-assembly. In R. Brijder and L. Qian, editors, *DNA Computing and Molecular Programming*, pages 98–109, Cham, 2017. Springer International Publishing.

[24] K. F. Wagenbauer, C. Sigl, and H. Dietz. Gigadalton-scale shape-programmable dna assemblies. *Nature*, 552(7683):78–83, 2017.

[25] H. Wang. Proving theorems by pattern recognition — ii. *The Bell System Technical Journal*, 40(1):1–41, 1961.

[26] E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

Partitioning Colored Points into Monochromatic Islands is NP-Complete

Steven van den Broek* Marc van Kreveld† Soeren Terziadis*‡

Abstract

We are given a set S of colored points and a positive integer κ. A subset of S is monochromatic if it contains points of only one color. We prove that it is NP-complete to decide whether S can be partitioned into at most κ monochromatic subsets whose convex hulls are pairwise-disjoint.

1 Introduction

Research on partitioning and separating colored points is abundant in both discrete and computational geometry. For an overview of discrete-geometry results, we refer to a recent survey [19]. The shapes used to partition or separate the points may have constant-complexity such as lines [8, 10, 15, 16, 18, 21, 22], triangles [2, 7, 23], disks [9, 11, 24], rectangles [1, 5, 29], wedges [15, 17], strips [15, 17], or L-shapes [25]; or they may have linear complexity such as convex polygons [3, 4, 12, 13].

Here we study the problem of using convex polygons to partition colored points into monochromatic islands. A set of points is *monochromatic* if it contains points of only one color. For a colored point set P, an *island* I is defined by Bautista-Santiago et al. [4] to be a subset of P such that $\mathcal{CH}(I) \cap P = I$, where $\mathcal{CH}(I)$ denotes the convex hull of I (including the interior).

We prove that the following problem is NP-complete (see Figure 1 for an illustration of the problem):

2C-IP Given a bichromatic set of points and a positive integer κ, does a partition of the points into at most κ monochromatic islands exist?

We identify an island (a subset of P) with its convex hull (a subset of $\mathbb{R}^2$). In particular, a partition into islands has the requirement that the convex hulls of the islands are pairwise-disjoint. Furthermore, we will refer to the two colors in the 2C-IP problem as red and blue.

*Department of Mathematics and Computer Science, TU Eindhoven, {s.w.v.d.broek, s.d.terziadis}@tue.nl

†Department of Information and Computing Science, Utrecht University, m.j.vankreveld@uu.nl

‡S.T. has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101034253.

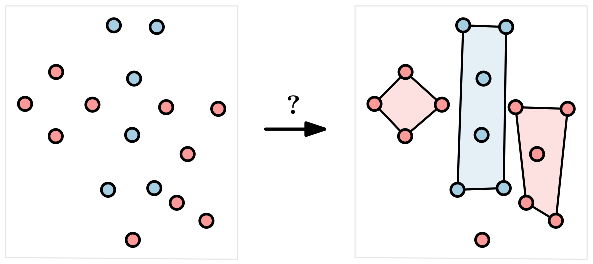


Figure 1: The 2C-IP problem. A set of red and blue points and a partition into four monochromatic islands.

Related work. Monochromatic island partitions have been studied before [12, 27]. There is also work on finding an optimal monochromatic island in a colored point set [4] (with a flexible definition of 'optimal'), and island partitions have also been used to define a notion of *coarseness* that captures how blended a set of bicolored points are [6]. However, to our knowledge, the computational complexity of the 2C-IP problem has not been studied. The problem of deciding whether a bichromatic set of points can be *covered* by κ monochromatic islands is known to be NP-complete [4] as it follows from a reduction by Agarwal and Suri [2]. Agarwal and Suri reduce from the planar 3-SAT problem by using one color of points to constrain islands of the other color. However, this reduction fails for the 2C-IP problem: islands formed by blocker points cannot overlap islands of the other color. This fact makes an NP-hardness reduction more challenging. We circumvent this issue in our reduction by creating a structured problem instance where the optimal partitions of the blue points and the red points have little interaction. Our reduction is inspired by the reduction by Van Kreveld, Speckmann, and Urhausen [28] for a similar problem.

2 Overview

We show hardness via a reduction from an independent-set problem on line segments (which we will simply refer to as segments from now on). The problem asks: given a set of n segments and a positive integer k, does a subset of k segments exist that are pairwise disjoint? Kratochvil and Nešetřil [20] showed this to be NP-complete, even if the segments are aligned with exactly three directions and no segment endpoint lies on another segment. We adapt their reduction to show that this problem is still NP-complete even if there exists a regular triangular grid T defined by $O(n^2)$ grid lines such that:

- the segments and their endpoints lie on grid lines and grid vertices of T respectively;
- no pair of segments lie on the same grid line of T.

Following Kratochvil and Nešetřil's notation we refer to this problem as pure (segments cannot contain other endpoints) equiangular 3-directional segment maximum independent set (PEA3D) and show it is NP-complete in Section 3. While it is not surprising that the problem stays NP-complete under the given restrictions and the proof requires only minor changes to the original reduction [20], this variant might be of independent interest as a basis for reductions to problems on triangular grids.

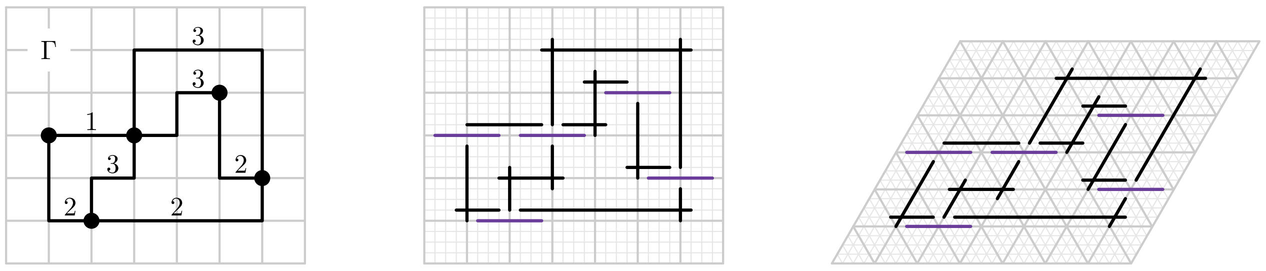
In Section 4 we reduce PEA3D to 2C-IP. The triangular grid T of the PEA3D instance is used as a basis for the 2C-IP instance. We shrink each face of T slightly, which gives rise to narrow corridors in which we place blue points to represent the segments of PEA3D. We also place blue points in the interior of the triangles and red points on the boundary of triangles. Our 2C-IP instance separates red and blue island candidates and limits their interaction in the following sense. We show that there exists an optimal partition P_r of the red points into monochromatic islands (avoiding the blue points but not blue islands) and an optimal partition P_b of the blue points into monochromatic islands (avoiding the red points), such that the islands in P_r and P_b are pairwise disjoint. Our instance provides this separation by allowing us to place blue points in the interior of triangles to block red islands, without needing additional islands to cover these 'blockers'. Similarly, red blockers can be freely placed on the boundary of triangles.

3 NP-completeness of PEA3D

Kratochvil and Nešetřil [20] reduce the independent-set problem in a planar graph $G = (V, E)$ of maximum degree 4 (which remains NP-hard [14]) to finding an independent set of segments with exactly three distinct slopes. To show hardness of PEA3D we retrace the steps of this reduction and make small changes in several steps to guarantee that the constructed set of segments complies with our additional restrictions.

Their reduction starts by creating a planar orthogonal drawing Γ of G (in polynomial time and on an $O(|V|) \times O(|V|)$ grid [26]; an example is shown in Figure 2a). Every edge $e \in E$ is represented by a polyline with k_e linear pieces (the numbers in Figure 2a). The idea of the reduction is to represent Γ by a set S of segments, such that the maximum independent-set size of S is proportional to the maximum independent-set size of G. The set S of segments Kratochvil and Nešetřil create form an intersection representation of an auxiliary graph G'; that is, there is a segment in S for every vertex of G' and two segments intersect if and only if their vertices are connected by an edge. Graph G' is constructed by subdividing edges of G. To recreate drawing Γ with segments, every edge of G needs to be subdivided by at least k_e vertices, and to ensure the relation between independent-set sizes every edge of G needs to be subdivided by an even number of vertices. Hence, Kratochvil and Nešetřil define G' by subdividing each edge by adding an even number $2\lfloor(k_e+1)/2\rfloor+2$[1] of vertices. This implies the size $\alpha(G')$ of a maximum independent set in G' is equal to $\alpha(G) + \sum_{e\in E}(\lfloor(k_e+1)/2\rfloor + 1)$. They then create S by replacing every vertex of Γ with a horizontal segment and replacing every linear piece of an edge in Γ with a corresponding horizontal or vertical segment (Figure 2b). These edge segments are slightly elongated to create proper intersections at the points where edges in Γ have bends except for the segments that would intersect the vertex segments. These are instead slightly shortened on that end and if they are horizontal segments they are additionally slightly vertically offset to ensure that the vertex segments are not inter-

[1]The original publication typesets this as $2\,[(k_e+1)/2]+2$.



(a) Drawing Γ; numbers show the complexity k_e of an edge e (b) Segment representations of vertices (purple) and linear pieces of edges (black) (c) Segments after applying the scaling and skewing transformations

Figure 2: The first steps of the hardness reduction for PEA3D. We assume for illustration purposes that vertices in Γ do not lie directly next to each other, so that the segments representing vertices can be wide. This can always be ensured by doubling the resolution of the grid.

sected. Note that these operations can be performed such that the segments still lie on a regular square grid by doubling the resolution of the grid a constant number of times. So far, k_e segments have been placed per edge e. Before the next step of the reduction we apply our first adaption, which is applying the following linear transformation to S:

$$\begin{pmatrix} 1 & \tan(\pi/6) \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{3}/2 \end{pmatrix}.$$

This keeps horizontal segments horizontal, and first shrinks and then skews the vertical segments to ones of the same length but now with a slope of $\pi/3$ (this is shown in Figure 2c). To complete the segment intersection representation of G' (Figure 3a), Kratochvil and Nešetřil add an additional 2 or 3 segments if an edge e in Γ consists of an even or odd number of linear segments respectively. If e has an even number of segments, we place on both ends of e a small segment with slope $-\pi/3$ such that it intersects e and the vertex segment (case I in Figure 3b). If e has an odd number of segments, we similarly add on one end a single segment with slope $-\pi/3$; however, on the other end we instead add two segments with slopes $-\pi/3$ and $\pi/3$ (case II in Figure 3b). Note that these connection segments can always be placed in this way because the vertices are represented by segments with slope 0, and linear pieces of edges by segments with slope either 0 or $\pi/3$. Further note that it suffices to double the regular triangular grid a constant number of times such that the connection segments can be placed on the grid.

This ends the reduction by Kratochvil and Nešetřil; however, we now apply our final adaption. Let n denote the number of segments in S. There might still be up to $O(n)$ segments that lie on the same grid line. To avoid this, we can simply double the resolution of the grid until every grid line is replaced by sufficiently many (at most $O(n)$) grid lines, i.e., as many grid lines as there are line segments lying on that specific grid line (Figure 3c). It is easy to see that (i) any two segments that lay on the same grid line can be shifted to be on different grid lines in such a way that any two segments that intersected before still intersect after being shifted, and (ii) this procedure adds at most $O(n)$ grid new lines per original grid line and therefore the entire set of segments fits into a regular triangular grid with $O(n^2)$ grid lines.

By construction, the final set of segments forms an intersection representation of G', which has an independent set of size at least $k + \sum_{e\in E}(\lfloor(k_e+1)/2\rfloor+1)$ if and only if G has an independent set of size at least k. Therefore, the PEA3D problem is NP-hard.

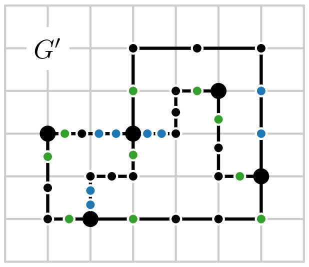
Lastly, NP-containment is straightforward since we can easily represent an independent set of segments embedded in an integer grid in polynomial space, and verify in polynomial time that (i) they are pairwise non-intersecting and (ii) that the set has a large enough cardinality. This completes our reduction and results in the following theorem.

Theorem 1 PEA3D *is* NP*-complete.*

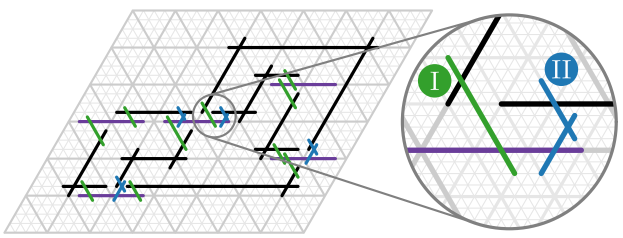
4 NP-completeness of 2C-IP

We are now set up to present our reduction from PEA3D to 2C-IP. Let S be a set of segments and let k be a positive integer such that together they form an instance of PEA3D. We will construct a set P of red and blue points and a value κ such that there exists an independent set of S of size k if and only if there exists a partition of P into κ monochromatic islands.

Auxiliary structure. Our reduction uses as auxiliary structure a bounded triangular unit-grid on which the segments S lie. We choose this grid such that it has a convex outer boundary and such that no interior point of a segment lies on this boundary. See Figure 4 for an illustration of this grid and upcoming definitions. Let T denote the set of points in $\mathbb{R}^2$ that lie on the edges and vertices of the grid that are not incident to the outer face. We shrink each (triangular) face of the grid by taking the Minkowski difference with a disk of diameter $\varepsilon = 0.1$ to create a set Δ of triangles. The



(a) Graph G'



(b) Intersection representation of G'

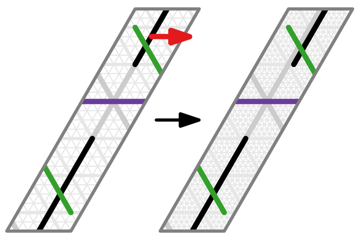
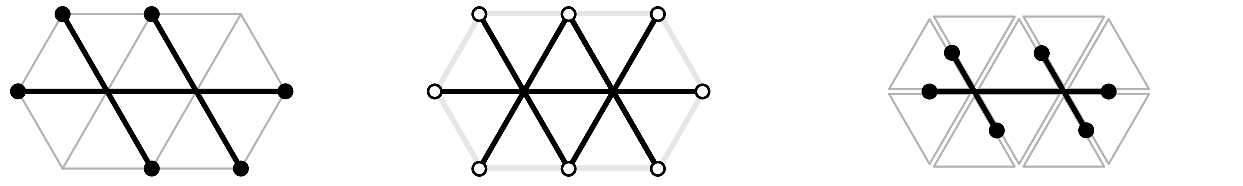

(c) Moving segment on new grid line

Figure 3: The last steps of the reduction for PEA3D. The segments in (b) lie on a regular triangular grid consisting of $O(n)$ grid lines. Connections to vertex segments are made using either one (I) or two (II) segments, which correspond to green and blue subdivision vertices of (b), respectively. After step (c), there are $O(n^2)$ grid lines.



(a) Segments S in triangular grid (b) Set T (in black, open at markers) (c) Triangles Δ and shortened segments

Figure 4: Structure of the 2C-IP instance.

free space between the triangles of Δ form narrow *corridors* of width ε. Note that each corridor corresponds to a grid line—which we will refer to as *rows* from now on—in T. We call the areas where three corridors meet *junctions*. Each junction corresponds to an *interior grid point* in T: a point where grid rows intersect. The segments of corridor in between two junctions form *alleys*, which correspond to edges of the grid T. For example, grid T in Figure 4 has 5 rows and 2 interior grid points, and the corresponding set of triangles Δ has 5 corridors, 2 junctions, and 11 alleys. Lastly, we shorten each segment in S on both sides by half a unit such that they end exactly in the middle of a grid edge; note that this preserves the intersection pattern since the endpoints of a segment in S do not lie on any other segment in S.

2C-IP instance. We create points of P in two phases. The first set of points represent triangles in Δ or segments in S; the second set of points limits the type of monochromatic islands that can connect points of the first set. We begin by placing a blue point at the centroid of every triangle (*triangle points*) and a blue point at the end of every segment (*segment points*). In every alley we place a red point at the center of each of the two triangle edges it is bounded by; we call these *bracket points* and the pair a *bracket*. See Figure 5 for an illustration. If the alley also contains a segment point, we say the bracket is *filled*, otherwise the bracket is *empty*. Each bracket point lies in exactly one corridor and one alley. Note that for each alley, the two bracket points in that alley as well as a potential segment point are all collinear with the triangle points of the triangles in Δ that bound the alley. This completes the first phase of the point placement. Let B and R be the set of all blue and red points placed so far, respectively.

In the second phase we place a set of red and a set of blue *blocker* points. For every pair p, q of distinct points in B such that p and q are not segment points of the same segment, the segment $\overline{pq}$ intersects the boundary of a triangle in Δ. This intersection exists since two segment points lie in the same corridor only if they originate from the same segment; indeed, no two segments of S lie in the same grid row, which implies that that no two endpoints of distinct shortened segments lie in the same grid row. We place a red blocker point at an arbitrary such intersection (Figure 6). Any island that contains both p and q also contains the red blocker point. Hence, after placement of these blockers, we have the following property.

Observation 1 *Two distinct blue points $p, q \in B$ can be part of the same monochromatic island only if p and q are the segment points of the same segment.*

Similarly for every pair (r, s) of red points in R lying in different corridors, the segment $\overline{rs}$ intersects a triangle. Consider the three straight-line segments emanating from the centroid of the triangle and ending at one of the corners of the triangle respectively. The segment $\overline{rs}$ intersects at least one of these straight-line segments. We place a blue blocker at an such arbitrary intersection (Figure 6). This yields a similar property as before, but now for the red points.

Observation 2 *Two points $r, s \in R$ can be part of the same monochromatic island only if they lie in the same corridor and do not form a bracket that is filled.*

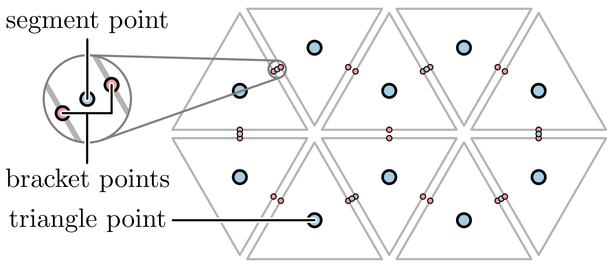


Figure 5: Placement of the first set of points.

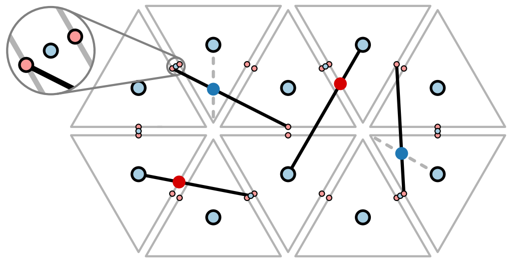


Figure 6: Examples for placement of blockers (solid red and solid blue points).

Note that all red blocker points lie on the boundary of triangles in Δ and all blue blocker points lie in the interior of triangles in Δ; hence, the sets of red and blue blocker points are disjoint. Let Z be the set of all placed blocker points. The final set of constructed points is $P = B \cup R \cup Z$. Note that Observations 1 and 2 apply only to pairs of points in B and R, not between any two blue points or any two red points in P. Therefore, an island may, for example, cover blue points in different triangles given at most one of them is a triangle point.

With this the construction of our instance is almost complete. It remains to state the value κ. Let ρ and θ be the number of rows and interior grid point in grid T respectively. We set $\kappa = |\Delta| + \rho + 4|S| + 2\theta - 2k$. The reason behind this particular choice for the value of κ will become apparent in the following sections.

Triangular grid. Before arguing how the constructed instance can be used to solve the independent-set problem, we state two observations on triangular grids.

Observation 3 *There is a bijection between minimum-cardinality segment partitions of T and maps that choose a grid axis at every interior grid point of T.*

Note that the segments in the observation above are not related to the set S of segments of the independent-set problem. Intuitively, a map such as described in the lemma above chooses at every interior grid point the segment of which row ‘continues’, with the segments from the other rows stopping at the grid point. Such a choice has to be made as the grid point should be covered by exactly one segment; conversely, each choice uniquely defines a minimum-cardinality segment partition (Figure 7). It is both sufficient and necessary to use one segment for each grid row and two additional segments for each interior grid point. Therefore, we state the following observation.

Observation 4 *A minimum-cardinality partition of T into segments has size $\rho + 2\theta$.*

NP-completeness. We now prove that a minimum-cardinality partition of P into monochromatic islands has cardinality κ if and only if k is the size of the maximum independent set of S. We start by proving the upper bound.

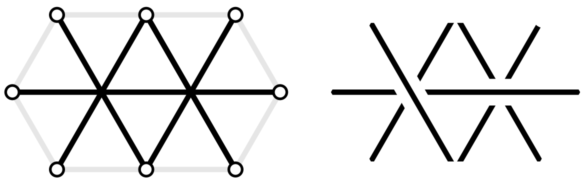


Figure 7: Left: grid T (markers and gray boundary are not part of T). Right: segment partition of T of size 9.

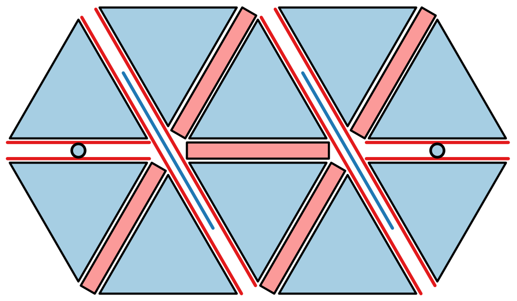


Figure 8: The structure of an optimal island partition. For illustration purposes the corridors have been made wider and the red islands have been moved into the corridors instead of lying on the boundary of triangles.

Lemma 2 *If S has an independent set of size k then P can be partitioned into κ monochromatic islands.*

Proof. Let N be the independent set of S of size k. A partition of P into κ islands can be constructed as follows; see Figure 8 for an example. Cover the interior of each triangle in Δ with a blue island. For each segment in N, cover the corresponding pair of segment points with a blue island. For each segment not in N, cover each of the corresponding segment points with a singleton island. To construct the islands covering red points, at every interior grid point of T we choose an axis as follows. If the grid point is covered by a segment in N then we choose the axis with which the segment is aligned; otherwise we choose an arbitrary axis. By Observation 3, this mapping of grid point to axis corresponds to a partition of T into segments. Call this set of segments L. For each segment $\ell \in L$ that covers a segment point in P, create two corresponding segment-shaped islands parallel to ℓ to cover red points in the corresponding alleys. For each segment in L that does not cover a segment point in P, create one quadrilateral island to cover the red points. A red point of P may lie exactly on the vertex of a triangle. Such a point would be covered by two islands in our construction; we can fix this simply by deleting such a point from one of the two islands.

All blue points are either segment points or lie in the interior of a triangle of Δ, both of which are covered by islands. All red points lie on the edges of corridors, which are also all covered by islands. Furthermore, all islands are pairwise-disjoint and monochromatic by construction. The problem instance uses $|\Delta|$ islands to cover the interiors of triangles and $2|S| - k$ islands to cover segment points. By Observation 4, set L has size $\rho + 2\theta$. Each of the k segments in L that covers two segment points is split into two red islands. Each of the $2(|S| - k)$ segments in L that covers one segment point is split into two red islands. The remaining segments correspond to exactly one red island. Hence, the total number of islands is:

$$\begin{aligned} &|\Delta| + 2|S| - k + \rho + 2\theta + k + 2(|S| - k) \\ = &|\Delta| + 4|S| - 2k + \rho + 2\theta = \kappa \end{aligned}$$

□

Next, we match the upper bound with a corresponding lower bound.

Lemma 3 *If S has a maximum independent set of size k then P cannot be partitioned into fewer than κ monochromatic islands.*

Proof. We argue that any monochromatic island partition of P uses at least κ islands. Our argument consists of two parts, one for the set of blue points and one for the set of red points.

The set B has by construction cardinality $|\Delta| + 2|S|$. By Observation 1, a monochromatic island that covers points of B covers either (i) a single point of B or (ii) a pair of segment points belonging to the same segment. Note that each island of type (ii) covers a distinct segment in S in the sense that the points in the convex hull of the island are a superset of the points on the segment. A set of islands of type (ii) then cover a set of pairwise non-intersecting segments of S. As such an independent set of S has size at most k, there can be at most k islands of type (ii) in an island partition. Hence, an island partition of P requires at least $|B| - k = |\Delta| + 2|S| - k$ blue islands.

By Observation 2, red points in R can be part of the same monochromatic island only if they are in the same corridor; furthermore, red islands from only one row can cross a junction. Therefore, by Observation 3, an island partition can be turned into a segment partition of T: at every grid point of T, if an island using at least two points in R crosses the corresponding junction, then choose the grid axis that corresponds to the corridor in which the two red points lie; otherwise, choose an arbitrary axis. Call this segment partition L. We map a point $r \in R$ to the segment of L that crosses the alley that contains r. By Observation 4, this yields $\rho + 2\theta$ sets of red points, each corresponding to a segment in L. The convex hull of such a set of points can contain either (i) no segment point, (ii) one segment point, or (iii) two segment points. Note that, by construction, an island of the partition covers points from at most one set. Therefore, we can bound the number of islands by bounding the number of sets. Sets of type (i) require at least one island (as they are non-empty). Sets of type (ii) and (iii) require at least two islands. Sets of type (iii) arise from segments of L that cover a set of pairwise non-intersecting segments of S. Hence, as an independent set of S has size at most k, there are at most k sets of type (iii). Thus, there are at least $2|S| - 2k$ sets of type (ii) or (iii), and an island partition of P requires at least $\rho + 2\theta + k + (2|S| - 2k) = \rho + 2\theta + 2|S| - k$ red islands.

By summing the two lower bounds, we find that a monochromatic island partition of P uses at least $|\Delta| + 4|S| - 2k + \rho + 2\theta = \kappa$ islands. □

The two lemmas combine to form our main result.

Theorem 4 2C-IP *is* NP*-complete.*

Proof. Let (S, k) be a instance of PEA3D and (P, κ) the corresponding instance of 2C-IP. By Lemma 2 and 3, set S has an independent set of size k if and only if P can be partitioned into κ monochromatic islands. Hence, because (P, κ) can be constructed in polynomial time from (S, k) and has polynomial size, NP-hardness follows from Theorem 1. To show containment in NP, note that we can represent a solution to 2C-IP in polynomial space as each island is a subset of P and we can check in polynomial time whether a set of subsets of P is indeed a monochromatic island partition of P. □

5 Conclusion

We have shown NP-completeness of 2C-IP. The 2C-IP problem is related to the similar problem 2C-TP where triangles are used instead of convex polygons to partition the points; variants of 2C-TP have been studied by Agarwal and Suri [2] and Bergold et al. [7]. Our reduction does not apply immediately to the 2C-TP problem, so the complexity of 2C-TP is still open. It does, however, hold when covering with quadrilaterals. Our reduction makes heavy use of collinear points. By replacing every point in R and B by an appropriate pair of points, we suspect the reduction can be adapted to hold true for points in general position.

References

[1] A. Acharyya, M. De, S. C. Nandy, and S. Pandit. Variations of largest rectangle recognition amidst a bichromatic point set. *Discrete Applied Mathematics*, 286:35–50, 2020.

[2] P. K. Agarwal and S. Suri. Surface approximation and geometric partitions. *SIAM Journal on Computing*, 27(4):1016–1035, 1998.

[3] C. Alegría, D. Orden, C. Seara, and J. Urrutia. Separating bichromatic point sets in the plane by restricted orientation convex hulls. *Journal of Global Optimization*, 85(4):1003–1036, 2023.

[4] C. Bautista-Santiago, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urrutia, and I. Ventura. Computing optimal islands. *Operations Research Letters*, 39(4):246–251, 2011.

[5] S. Bereg, S. Cabello, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, and I. Ventura. The class cover problem with boxes. *Computational Geometry*, 45(7):294–304, 2012.

[6] S. Bereg, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, C. Seara, and J. Urrutia. On the coarseness of bicolored point sets. *Computational Geometry*, 46(1):65–77, 2013.

[7] H. Bergold, A. K. Das, R. Lauff, M. Scheucher, F. Schröder, and M. D. Sieper. On triangular separation of bichromatic point sets. In *Booklet of abstracts of European Workshop on Computational Geometry (EuroCG)*, 2025.

[8] E. Bonnet, P. Giannopoulos, and M. Lampis. On the parameterized complexity of red-blue points separation. *Journal of Computational Geometry*, 10(1):181–206, 2019.

[9] S. Cabello, J. M. Díaz-Báñez, and P. Pérez-Lantero. Covering a bichromatic point set with two disjoint monochromatic disks. *Computational Geometry*, 46(3):203–212, 2013.

[10] G. Călinescu, A. Dumitrescu, H. J. Karloff, and P. Wan. Separating points by axis-parallel lines. *International Journal of Computational Geometry & Applications*, 15(6):575–590, 2005.

[11] A. Cannon and L. Cowen. Approximation algorithms for the class cover problem. *Annals of Mathematics and Artificial Intelligence*, 40(3-4):215–224, 2004.

[12] A. Dumitrescu and J. Pach. Partitioning colored point sets into monochromatic parts. *International Journal of Computational Geometry & Applications*, 12(5):401–412, 2002.

[13] H. Edelsbrunner and F. P. Preparata. Minimum polygonal separation. *Information and Computation*, 77(3):218–232, 1988.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, New York, 1979.

[15] E. Glazenburg, T. van der Horst, T. Peters, B. Speckmann, and F. Staals. Robust bichromatic classification using two lines. In *Proc. 35th International Symposium on Algorithms and Computation (ISAAC)*, pages 33–1, 2024.

[16] F. Hurtado, M. Mora, P. A. Ramos, and C. Seara. Separability by two lines and by nearly straight polygonal chains. *Discrete Applied Mathematics*, 144(1-2):110–122, 2004.

[17] F. Hurtado, M. Noy, P. A. Ramos, and C. Seara. Separating objects in the plane by wedges and strips. *Discrete Applied Mathematics*, 109(1-2):109–138, 2001.

[18] H. Jowhari and M. Rezapour. Monochromatic partitioning of colored points by lines. *Information Processing Letters*, 182:106402, 2023.

[19] M. Kano and J. Urrutia. Discrete geometry on colored point sets in the plane - A survey. *Graphs and Combinatorics*, 37(1):1–53, 2021.

[20] J. Kratochvíl and J. Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 031(1):85–93, 1990.

[21] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

[22] N. Megiddo. On the complexity of polyhedral separability. *Discrete & Computational Geometry*, 3:325–337, 1988.

[23] Z. Moslehi and A. Bagheri. Separating bichromatic point sets by minimal triangles with a fixed angle. *International Journal of Foundations of Computer Science*, 28(04):309–320, 2017.

[24] J. O'rourke, S. Rao Kosaraju, and N. Megiddo. Computing circular separability. *Discrete & Computational Geometry*, 1:105–113, 1986.

[25] F. Sheikhi, A. Mohades, M. de Berg, and M. Davoodi. Separating bichromatic point sets by L-shapes. *Computational Geometry*, 48(9):673–687, 2015.

[26] L. G. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, 30(2):135–140, 1981.

[27] S. van den Broek, W. Meulemans, and B. Speckmann. Greedy monochromatic island partitions. In *Booklet of abstracts of European Workshop on Computational Geometry (EuroCG)*, 2024.

[28] M. van Kreveld, B. Speckmann, and J. Urhausen. Diverse partitions of colored points. In *Proc. 17th Algorithms and Data Structures Symposium (WADS)*, pages 641–654, 2021.

[29] T. van Lankveld, M. van Kreveld, and R. Veltkamp. Identifying rectangles in laser range data for urban scene reconstruction. *Computers & Graphics*, 35(3):719–725, 2011.

Puzzles are hard enough just by rotations

Takeshi Yamada* Tom van der Zanden† Ryuhei Uehara*

Abstract

In this paper, we investigate the computational complexity of a series of puzzles. We are given a set of n centers of circles and n unit disks. Each disk is based on a unit circular shape, but a part of it can be bitten by other disks, which is called a lune. We investigate three variants of this puzzle. First, we investigate the classic packing puzzle of lunes. We note that the centers of disks are given as a part of input, and some disks can be lunes. Therefore, essentially, we only can rotate the disks and lunes to pack them. Even under this strong constraint, the packing problem is still NP-complete. Next we turn to the combinatorial reconfiguration variant of this puzzle. That is, we are given two nonoverlapping arrangements of disks and lunes on a given set of centers on a board. Each disk is pinned at the center, and thus we can just rotate it. The problem asks if we can transform one to the other by just rotations of disks without overlapping. We show that this puzzle is PSPACE-complete. Lastly, we focus on the cases in which the puzzle can be solved in polynomial time. The first tractable case is one-dimensional packing puzzle. The second one is the screw-type variant of this puzzle. In this variant, each disk or lune is realized by a thick screw. We are given a nonoverlapping arrangement of them. The operation we can do is that we can screw a disk if its orbit is not blocked by any other screw. We prove that this variant can be solved in polynomial time.

1 Introduction

In the context of research on computational complexity and algorithms, the study of puzzles has played an important role. One of the reasons is that an algorithm is a methodology for how to combine basic operations, while a puzzle is an abstract model of how to combine the basic pieces under some constraints of pieces. For example, in [1], Asano et al. investigate the computational complexity of a puzzle that asks for finding proper order of a deck of cards with/without rotation and/or flipping. On the other hand, in [4], Hearn and Demaine propose a general framework that can capture the computational complexities of many puzzles, including sliding block puzzles. This framework is useful to show the hardness of many problems in combinatorial reconfiguration, especially, token sliding problem on a graph. Recently, in [3], Kanzaki et al. investigate the jumping block puzzles, which is a puzzle counterpart of token jumping problem on a graph investigated in the context of the combinatorial reconfiguration.

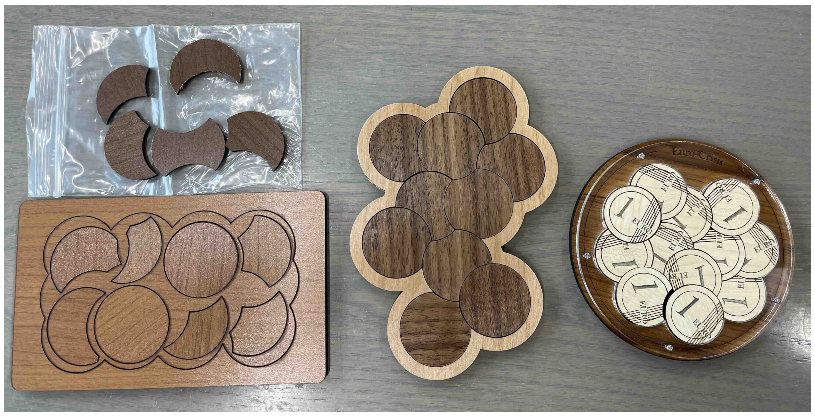
In this paper, we investigate representative puzzles that impose strong constraints on both of movements and pieces. This is a framework that has long been popular in the puzzle society, and many commercially available products exist (Figure 1). These puzzles have the following common property: (1) each piece is based on a unit disk of radius 1, (2) some pieces can be missing parts, which is called *lunes*, (3) the centers of disks are essentially fixed (or easy to specify). Let's call these puzzles *rotation puzzles*. That is, the input of the puzzle is a set of n unit disks or lunes and a set of n centers. We call each piece *lune* since they can be missing parts from a unit disk. We assume that the center can be easily determined from the lune. Precisely, when a lune is inscribed in a unit circle, the center point of the lune should be uniquely determined.

We investigate three variants of the rotation puzzles. The first one is the *packing puzzle* that asks us to determine (or to find) if all the lunes can be put on a board so that the centers of the disks are matching to the given centers, and no pair of lunes overlaps (Figure 1(a)). We call such a solution *feasible solution*. That is, the packing puzzle asks us to determine if a given set of n centers and n lunes have a feasible solution. We show that this problem is NP-complete (in Section 2). The next puzzle is the *reconfiguration puzzle* that asks us, for two feasible solutions S and T of the lunes with fixed centers, to determine if S can be transformed to T by a sequence of rotations of lunes. That is, we cannot pick up any piece, and the only operation we can do is rotate a lune at once. We prove that this quite restricted puzzle is PSPACE-complete in general (in Section 3).

We turn to some tractable cases in Section 4. We first show that one-dimensional packing puzzle is linear-time solvable. That is, when the centers are on a line, it can be solved efficiently. This case is inspired by one of classic puzzles shown in Figure 1(c), which requires exponential operations to take out. Another tractable case is the *screw puzzle* (Figure 1(d)). This puzzle is relatively new one comparing to the other classic puz-

*School of Information Science, Japan Advanced Institute of Science and Technology, {tyama,uehara}@jaist.ac.jp

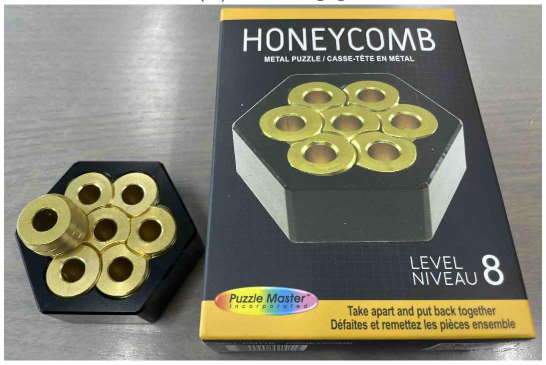
†Department of Data Analytics and Digitalization, Maastricht University, t.vanderzanden@maastrichtuniversity.nl



(a) Circle Crazy, Balloon Boogler[1], Euro-Crisis[2]

(b) Whirligig Plate

(c) Spin Out


HONEYCOMB
METAL PUZZLE / CASSE-TÊTE EN MÉTAL
LEVEL NIVEAU 8
Puzzle Master
Take apart and put back together
Défaites et remettez les pièces ensemble

(d) Honeycomb[3]

Figure 1: Lune puzzles based on (a) packing and (b) rotation. (c) A classic puzzle "Spin Out" and (d) a lune puzzle based on screwing.

zles. Each piece forms a screw, and we can take it if we can rotate it. In the real puzzle shown in Figure 1(d), it is not difficult to find the first two pieces screwed out, however, the third one to be taken out is not so easy. (We will discuss this point in Section 4.) We will show that this puzzle can be solved in polynomial time.

In this paper, all lunes $\ell_1, \dots, \ell_n$ are of radius 1. That is, every lune is inscribed in a unit circle, and it cannot be inscribed in a circle of radius $r < 1$. Each of n centers c_i is given by its coordinates (x_i, y_i). (We assume a RAM model as a computational model and each number takes a real value.) A lune is obtained from a unit disk by bitten out some constant number of overlaps with other disks. Therefore, a lune ℓ_i can be represented by the set of unit disks that bite it off.

[1]https://puzzlemist.com/product/balloon-boggler-puzzle/

[2]https://www.puzzlemaster.ca/browse/inventors/jurgenreiche/16681-euro-crisis-deluxe

[3]https://www.puzzlemaster.ca/browse/metalpuzzles/metalpuzzlemaste/18111-honeycomb-metal-puzzle

2 NP-completeness of packing puzzle

In Figure 1(a), we show some commercial products. We can generalized this puzzle as follows:

Packing puzzle of bitten disks

Input: Two sets of n centers c_i and n lunes ℓ_i for $i = 1, \dots, n$.

Output: Decide if we can arrange n lunes onto n centers without overlapping of lunes.

The main theorem in this section is below:

Theorem 1 *The packing puzzle of lunes is NP-complete in general.*

Proof. (Outline.) It is easy to see that this problem is in NP since we can check if a given arrangement is feasible in polynomial time. Thus we show NP-hardness by reducing the well-known planar 1-in-3 SAT to the packing puzzle. (For NP-hardness of the planar 1-in-3 SAT problem, see, e.g., [5].) Let $F = C_1 \wedge \cdots \wedge C_m$ be an instance of Planar 1-in-3 SAT, where C_i is a clause of three literals in $X = \{x_1, x_2, \dots, x_{n'}\}$ and $\bar{X} = \{\bar{x_1}, \bar{x_2}, \dots, \bar{x_{n'}}\}$. We assume that the corresponding graph $G = (V, E)$ for F defined by (1) $V = X \cup \bar{X}$ and (2) $E = E_1 \cup E_2$ where $E_1 = \{\{x_i, \bar{x_i}\}\}$ for each i and $E_2 = \{\{l_i, C_j\}\}$ for each literal l_i in C_j is planar. We first prepare $n' + 1$ different real numbers $\sqrt{2} < r_1 < \cdots < r_{n'} < r_{n'+1} < 2$. We construct *clause gadget*, *wire gadget*, and *variable gadget* as follows.

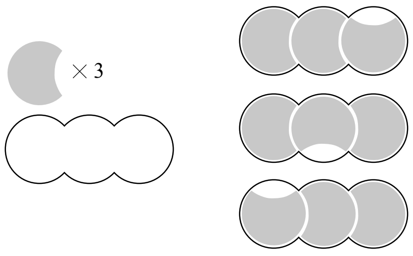


Figure 2: A clause gadget.

A clause gadget consists of three congruent lunes and three centers on a line as shown in the left of Figure reffig:clause. In the figure, we describe boundaries of gadget instead of centers for ease to see. The distance of each pair of consecutive two centers is $r_{n'+1}$, and each lune lacks one lens-shape corresponding to $r_{n'+1}$. (We say that this lens-shape has a *thickness* of $r_{n'+1}$.) When we pack three lunes onto the three centers, as shown in the right of Figure reffig:clause, exactly one of three lunes can have one lens-shape gap. We will use this gap to represent one true literal. We prepare m copies of clause gadgets with reasonable margins.

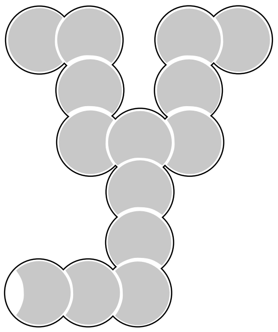


Figure 3: Wire gadgets with turns.

A wire gadget is the same as the clause gadget. It consists of sequence of centers of uniform distance $r_{n'+1}$ from one circle in a clause gadget. We prepare the same lunes that lack one lens-shape of thickness $r_{n'+1}$. The sequence of the wire gadgets extends the clause gadget in three directions in a natural way as shown in Figure 3. It is easy to make a turn.

The key gadget is the variable gadget for each variable x_i. A variable gadget for the variable x_i consists of $2a_i + 2$ centers spaced equally apart on a line with a distance r_i between them, where a_i is the number of appearance of x_i in F. It contains $a_i + 1$ unit disks, one

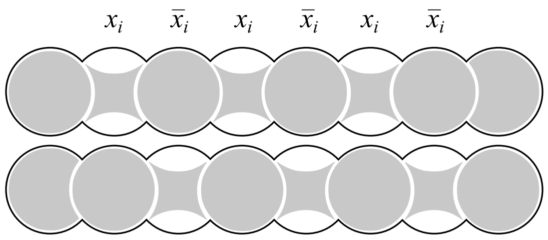


Figure 4: A variable gadget for $a_i = 3$ with two ways of packing.

lune lacking one lens-shape of thickness r_i, and a_i lunes that lack 4 lens-shapes of thickness r_i. It is not difficult to see that we only have four possible ways to pack those lunes and disks onto the centers and we only have two possible boundaries of them as shown in Figure 4. In Figure 4, bottom one describes x_i is true, and the upper one describes x_i is false. Each wire gadget should be connected to the circle of the variable gadget labeled by the corresponding literal x_i or $\bar{x_i}$.

Using the same trick between a clause gadget and a wire gadget, we can connect a variable gadget to a corresponding clause gadget in a natural way. The reduction can be done in a polynomial time. Thus we show that the packing puzzle has a solution if and only if F has an assignment that satisfy 1-in-3 SAT condition.

When F has a feasible assignment, each clause have one literal that is true. In the clause gadget, we make a gap at the corresponding lune. It is transferred by the wire gadget, and the corresponding position of the variable gadget can be convex. On the other hand, if we have a convex shape at the clause gadget since the corresponding literal is false, it is transferred by the wire, and the corresponding position of the vertex gadget should be concave. Therefore, when F has a feasible assignment, we can assign concave lune at each position corresponding to false literal and convex disk at each position corresponding to true literal in the variable gadget. Thus we can pack all the lunes.

Next we assume that the constructed packing puzzle has a solution. We first focus on the variable gadget for x_1. The gadget allow to pack lunes of thickness r_1 since $r_1 < r_i$ for all $i > 1$. Thus we need all lunes of thickness r_1 to pack it. But it has a gab between lunes. If we use some lunes of thickness r_i with $i > 1$ to fill the gap in the gadget for x_1, it is easy observe that we cannot pack the gadget for x_i. Using this argument repeatingly, we can observe that we have to use all lunes of thickness r_i to fill the gadget for the variable x_i. Then, we have to use all lunes of thickness $r_{n'+1}$ to fill the wire gadgets and clause gadgets because we have no gap in these gadgets. Thus, eventually, we use all unit disks to pack the gadgets for x_i. Therefore, when the packing puzzle has a solution, the way of packing gives us the feasible assignment of F, which completes the proof. □

3 PSPACE-completeness of reconfiguration puzzle

In the puzzle shown in Figure 1(b), the lunes are all the same shape and they are pinned at the uniform square grid. This puzzle and its generalization has been investigated in [6] as the *cyclic shift puzzle*. In this paper, we introduce a natural variant of our rotation puzzles in the context of the combinatorial reconfiguration (see, e.g., [2]):

Reconfiguration puzzle of lunes

Input: Two sets of n centers c_i and n lunes ℓ_i for $i = 1, \dots, n$, and two feasible arrangements S and T of the lunes on the centers.

Output: Decide if we can reconfigure from S and T by a sequence of rotations of lunes.

We note that all lunes are pinned at their centers, and we can just rotate them. We also assume that we rotate one lune at a time. This contrasts to the other reconfiguration problems that allow to shift (or slide), or even jump as discussed in [3].

The main theorem in this section is below:

Theorem 2 *The reconfiguration puzzle of lunes is PSPACE-complete in general.*

Proof. (Outline.) We reduce the nondeterministic constraint logic (NCL) proposed and discussed in [4]. As an NCL instance, we are given a directed graph $G_0 = (V_0, E_0)$ with an edge e in E_0. In each step, we can obtain E_{i+1} from E_i by flipping an edge. The NCL asks if a specific edge e can be flipped or not, and it is PSPACE-complete in general. Further more, in [4], it is still PSPACE-complete even if (1) G_0 is planar, (2) V_0 contains two types of nodes called "AND" and "OR" in [4, Theorem 5.12], and (3) each "OR" has only one "true" assignment [4, Theorem 5.13].

Therefore, essentially, it is sufficient to show that the "AND" node and the "OR" node in the NCL can be realized by our reconfiguration puzzle of lunes like the other puzzles discussed in [4, Chapter 9]. The wire gadget is already described in Figure 3. In the wire gadget, the wire send "a lens-shape gap" as a signal. Therefore, the AND gadget can be realized easily as shown in Figure 5. When it catches two lens-shape gaps at both sides of the central disk, the central lune can be rotated in 180°, and it can transmit the gap to down. We need some device for realizing the OR gadget as shown in Figure 6. In Figure 6, we cannot rotate the central lune since it is locked by two top neighbors. However, once one of the two neighbors is free by receiving a lens-shape gap as shown in Figure 7, it can be rotated until it strikes the other neighbor (in the figure, the central lune receives the lens-shape gap from left, and it rotated in counterclockwise, and left small gap comes to down). Then the down neighbor can be rotated freely, and it can provide a big lens-shape gap at left side to down.

Then, one of two small gaps at the central lune can be down, and its down neighbor can be rotated. □

4 Polynomial-time solvable puzzles

4.1 One-dimensional packing puzzle

The "Spin Out" shown in Figure 1(c) is a classic puzzle similar to the other famous classic puzzle "Chinese Ring". They are uniformly constructed and both require exponential number of operations like the Hanoi Tower. Inspired by the Spin Out, we can consider the packing puzzle of lunes discussed in Section 2 in one dimension. That is, we consider the following packing puzzle in 1D:

Packing puzzle of lunes in 1D

Input: Two sets of n centers c_i and n lunes ℓ_i for $i = 1, \dots, n$ such that (1) the centers are assumed to be equally spaced on a line with a distance $r < 1$ between them, and (2) each lune is (a) a unit disk, (b) it lacks one lens-shape of thickness r, or (c) it lacks two lens-shape of thickness r on opposite sides.

Output: Decide if we can arrange n lunes onto n centers without overlapping of lunes.

This case can be solved in linear time:

Theorem 3 *The packing puzzle of lunes in 1D is linear time solvable.*

Proof. (Outline.) Let L_a, L_b, L_c be the sets of lunes in the groups (a), (b), and (c), respectively. When $L_a \neq \emptyset$, we can assume that one element should be put on c_1 without loss of generality. Any pair of lunes in L_a cannot be adjacent. Therefore, the elements of L_a must be placed at least two apart. Between them, we have to use at least one element in L_c. We also observe that we can put each element in L_b in an arbitrary way between the elements in L_a and L_c. Thus we can pack all elements in $L_a \cup L_b \cup L_c$ if and only if $(|L_a| - 1) < |L_c|$, which can be checked in linear time. □

4.2 Screw puzzle

The Honeycomb Metal Puzzle is a new product released in 2023 (Figure 1(d)). We can consider this puzzle is a variant of a lune puzzle based on screwing, which we name as the *disassemble puzzle*. For a given arrangement of lunes, we can solve the disassemble puzzle in polynomial time when we restrict ourselves to just the rotation operation:

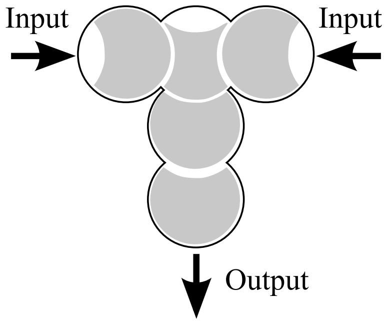


Figure 5: An AND gadget.

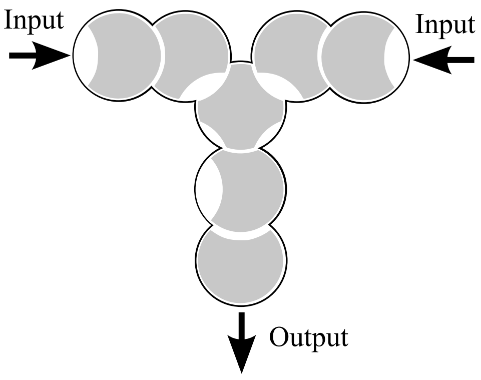


Figure 6: An OR gadget.

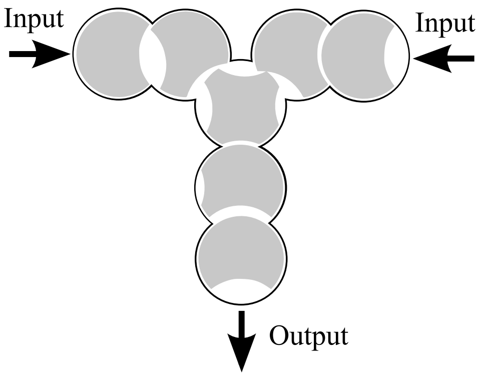


Figure 7: How an OR gadget transmits a signal from left to down.

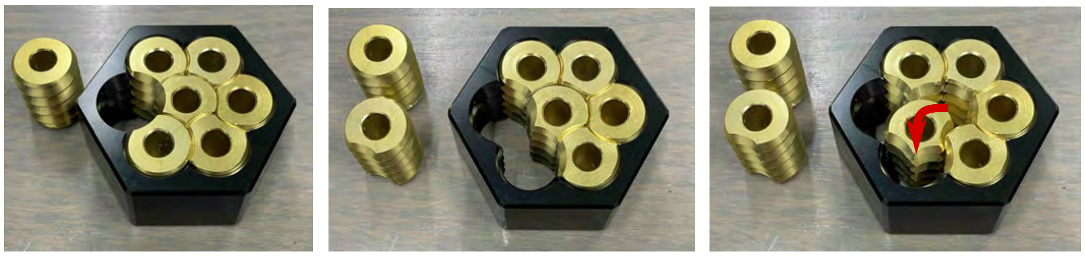


Figure 8: How to disassemble the Honeycomb Metal Puzzle.

Lemma 4 *The disassemble puzzle of lunes can be solved in polynomial time when the possible operations are restricted to rotations at the center of each lune only.*

Proof. For a given arrangement, we can construct a directed graph $G = (V, A)$ as follows. First, we let V be the set of lunes ℓ_i with $i = 1, 2, \ldots, n$. Then the set A contains a directed edge (ℓ_i, ℓ_j) if and only if ℓ_i is an obstacle that prevents ℓ_j from rotating. It is easy to observe that we can screw out ℓ_i from this arrangement when ℓ_i has indegree 0 on G. If we have such a lune, we screw it out and remove it from V. Repeating this process, we can disassemble, and we cannot otherwise. □

As observed in the introduction, in the arrangement in Figure 1(d), we can screw out the first two lunes. However, we cannot screw out the third one in place. That is, this disassemble puzzle has no solution if the possible operations are restricted to rotations at the centers of lunes only.

In fact, we have to rotate the third lune along a *different* center to take out it (Figure 8). It is not difficult to characterize when a lune can be moved locally in this way, as it depends on the conditions of its neighbors. (A similar analysis can be found in the similar puzzle based on hexagonal grid; see, e.g., [7].) For this commercial product, it is not difficult to find the way of disassembling. The more general case remains open.

References

[1] Tetsuo Asano, Erik D. Demaine, Martin L. Demaine, and Ryuhei Uehara. NP-completeness of generalized Kaboozle. *Journal of Information Processing*, Vol. 20, No. 3, pp. 713-718, July, 2012.

[2] Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the Complexity of Reconfiguration Problems. *Theoretical Computer Science*, Vol. 412, pp. 1054-1065, 2010.

[3] Masaaki Kanzaki, Yota Otachi, Giovanni Viglietta, and Ryuhei Uehara. Computational Complexity of Jumping Block Puzzles, *Theoretical Computer Science*, Vol. 983, 114292(19 pages), February 2024.

[4] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*, A K Peters, July 2009.

[5] Alexander Pilz. Planar 3-SAT with a Clause/Variable Cycle, *Discret. Math. Theor. Comput. Sci.*, Vol. 21, No. 3, 20 pages, 2019.

[6] Kwon Kham Sai, Ryuhei Uehara, and Giovanni Viglietta. Cyclic Shift Problems on Graphs. *IEICE Trans.*, Vol. E105-D, No. 3, pp. 532-540, March 2022.

[7] Yanqiu Tan. Research on the Computational Complexity of the Crush Ice Game. *Master thesis*, Japan Advanced Institute of Science and Technology, March 2025.

Input-Sensitive Reconfiguration of Sliding Cubes

Hugo A. Akitaya* Matias Korman† Frederick Stock*

Abstract

A configuration of n unit-cube-shaped *modules* (or *robots*) is a lattice-aligned placement of the n modules so that their union is face-connected. The reconfiguration problem aims at finding a sequence of moves that reconfigures the modules from one given configuration to another. The sliding cube model (in which modules are allowed to slide over the face or edge of neighboring modules) is one of the most studied theoretical models for modular robots.

In the sliding cubes model we can reconfigure between any two shapes in $O(n^2)$ moves ([Abel *et al.* SoCG 2024]). If we are interested in a reconfiguration algorithm into a *compact configuration*, the number of moves can be reduced to the sum of coordinates of the input configuration (a number that ranges from $\Omega(n^{4/3})$ to $O(n^2)$, [Kostitsyna *et al.* SWAT 2024]). We introduce a new algorithm that combines both universal reconfiguration and an input-sensitive bound on the sum of coordinates of both configurations, with additional advantages, such as $O(1)$ amortized computation per move.

1 Introduction

Modular self-reconfigurable robotic systems offer several advantages over typical robotic systems. Such systems are composed of a set of robotic units (called **modules**) that can communicate, attach, detach, and move relative to each other. This means modules can move to change the overall system's shape, providing versatility to perform unforeseen tasks. While advantageous, the increased degree of freedom also incurs an algorithmic challenge, as it is not easy to determine how to use the module's operations so that the union of all modules creates a specific shape (this is known as the **reconfiguration** problem). The algorithmic community has investigated the reconfiguration problem for many variations of modular robots, be it the type of moves allowed (sliding [5,9,10], pivoting [3,13], and so-called square "atoms" [7,8]) or shape (such as hexagons [4]).

We investigate one of the most established models, which is called the **sliding (hyper-)cube model**: modules are lattice-aligned unit (hyper-)cubes where two modules are **adjacent** if they share a face (i.e., a $(d-1)$-dimensional facet). In this model, two types of moves are allowed: a module can slide along the co-(hyper-)planar faces of two other adjacent modules, or it can slide along two orthogonal faces of another module (see Fig. 1).

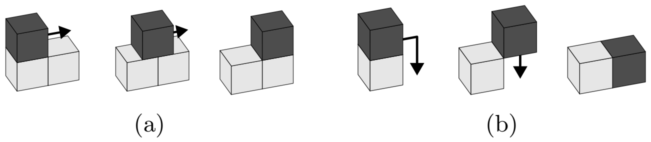


Figure 1: (a) Slide and (b) Convex transition.

The **reachability** problem asks whether, given two configurations with n modules, there exists a sequence of moves that can transform one into the other. If the answer is always 'yes' we say that the model admits **universal** reconfiguration.

The first universal result for the two dimensional sliding model was obtained by Dumitrescu and Pach [9] with a reconfiguration algorithm that required $O(n^2)$ moves. This result could be seen as optimal since some instances require $\Omega(n^2)$ moves. The 2-dimensional result was afterwards extended to three and higher dimensions in an informal publication of Abel and Kominers [2]. Their paper claimed universal reconfiguration and that $O(n^3)$ many moves were always sufficient.

Other research [3] focused on variations of the model, in both the shape of the modules and the type of movements that were allowed. They obtained characterization of which variations of the model have universal reconfiguration. For the versions in which no universal reconfiguration algorithm exists, they showed that determining if a particular instance can be reconfigured into a target configuration is PSPACE-complete.

Recently there have been two simultaneous, independent results in three dimensions. Abel *et al.* [1] formally published their preliminary result, reducing the number of moves to $O(n^2)$. In the same paper, the authors explain how the algorithm can be made in-place and input-sensitive (the exact bound depends on the dimension of the bounding box of both configurations). **In-place** means throughout the reconfiguration there are never more than a constant number of modules outside the bounding boxes of the start and end configuration. Further, these modules are never more than

*University of Massachusetts Lowell, Lowell, USA, {hugo_akitaya@, frederick_stock@student.}uml.edu.
Supported by NSF grant CCF-2348067.
†Siemens Electronic Design Automation, Wilsonville, USA, matias.korman@siemens.com

distance 1 from the bounding box [1]. Parallel to [1], Kostitsyna *et al.* [12] independently produced a different input-sensitive reconfiguration algorithm: a configuration C can be reconfigured into a **compact configuration**, intuitively, a shape where all modules are clumped together without holes (formal definition given in Section 2). Their algorithm takes $O(\sum_{\mathbf{m}\in C_1} \|\mathbf{m}\|_1)$ sliding moves where $\|\mathbf{m}\|_1$ denotes the L_1 norm of the position of a module $\mathbf{m}$. This result has a better input-sensitive bound, but is slightly limited: they only show how to reconfigure into a compact configuration. Since there is more than one compact configuration with the same number of modules, this does not directly lead to a universal reconfiguration strategy. Reconfiguration between compact configurations is significantly simpler than compactification, but does not immediately follow.

This paper we improve on both [1] and [12]: we give a universal reconfiguration whose number total number of moves depends on the sum of input and target coordinates ($O(\sum_{\mathbf{m}\in C_1} \|\mathbf{m}\|_1 + \sum_{\mathbf{m}\in C_2} \|\mathbf{m}\|_1)$. Since configurations are connected, it is easy to see that this bound is in $\Omega(n^{4/3}) \cap O(n^2)$ (when all modules form a solid cube or when they form the 1-skeleton of a larger cube, respectively). Our main result is formally stated as follows.

Theorem 1 *Given two configurations C_1 and C_2 with n cube modules in 3 dimensions where all modules lie in the positive xyz orthant, there is an in-place reconfiguration between them that uses at most $18(\sum_{\mathbf{m}\in C_1} \|\mathbf{m}\|_1 + \sum_{\mathbf{m}\in C_2} \|\mathbf{m}\|_1) + 120n + O(1)$ sliding moves where $\|\mathbf{m}\|_1$ denotes the L_1 norm of the position of a module $\mathbf{m}$. Such a reconfiguration can be computed in $O(1)$ amortized time per sliding move.*

In our algorithm we use a constant number of auxiliary "helper" modules that will allow nearby modules to move. We follow existing literature [3] and refer to the auxiliary modules as **musketeers**. The multiplicative factor 120 may seem large, we note that our algorithm uses six musketeer modules, so this constant is better characterized by $20n$ moves per musketeer module. Further, since we reconfigure between C_1 and C_2 by reaching an intermediate configuration, this means that we actually need $10n$ moves per musketeer to transform any configuration of n modules into *compact* form.

Although the main emphasis is in the total number of moves required, it is also interesting to bound the time needed to compute the sequence of moves needed to reconfigure between two configurations. Our algorithm can compute each move in $O(1)$ amortized time. Although the computation time is not directly addressed in the previous papers, it is not hard to see that the algorithm of [1] also achieves $O(1)$ amortized (after a BFS traversal on both configurations, each step is easily computed in O(1) time). On the other hand, a naïve implementation of [12] would require $\Theta(n)$ time per move as a global search must be executed each time. Due to space constraints some proofs are omitted. Details can be found in the extended version of this paper [6]

[1]Traditionally, in-place algorithms are a bit more restrictive, only one module is allowed outside the bounding box, making both constants exactly 1. In this paper we relax the condition for ease of description.

2 Definitions and Preliminaries

A **configuration** C of n unit cube modules is a set of n cells in the cube lattice. A cell is **occupied** if it is in C or **empty** otherwise. We abuse notation by sometimes referring to occupied cells as modules. Two cells are **adjacent** if they share a face. Let the **adjacency graph** G_C be the graph whose vertices are the cells in C and edges are defined by pairs of adjacent occupied cells. We call C **connected** if G_C is connected. A module is **articulate** if it is a cut vertex in G_C, and **nonarticulate** otherwise. For an occupied cell m, the notation $\mathbf{m}$ refers to the module at cell m. Note that a connected configuration C defines a polycube. We denote by ∂C the boundary of C. The **outer boundary** of C is the boundary of the unbounded component of the complement $\overline{C}$ of C. We say a module is "in the outer boundary" if at least one of its faces lies in the outer boundary of C.

A **move** is an operation that transforms an n-module configuration C into another C' so that $C \cap C'$ is a configuration of $n-1$ modules (and thus $C \setminus C'$ is a set with one module). We require the **single backbone condition**: a move between connected configurations C and C' is only allowed if $C \cap C'$ is also connected. We say that the module in position $C \setminus C'$ **moved** to position $C' \setminus C$. In other words, the single-backbone condition requires each moving module to be nonarticulate.

The **sliding model** allows two types of moves (refer to Fig. 1):

- A **slide** moves a module $\mathbf{a}$ from a to an adjacent empty cell b, and requires that there are adjacent occupied cells a' and b' such that a is adjacent to a' and b is adjacent to b'.
- A **convex transition** moves a module $\mathbf{a}$ from a to an empty cell b where a and b share a common edge e, and are both adjacent to an occupied cell c, and requires that the cell $d \notin \{a, b, c\}$ that contains e is empty. Note that every edge e is incident to exactly 4 cells.

Note a slide requires the final cell b to be empty, and a convex transition requires the target b as well as an intermediate cell to be empty. These are called the **free-space requirements** of the moves. If these requirements are not met, performing either of these moves would cause two modules to collide. Note that a module might be nonarticulate but is not allowed to move if the free-space requirements mentioned above are not satisfied. We call a module **movable** if it is nonarticulate and satisfies the free-space requirements of either a slide or a convex transition.

We may refer to a cell using the coordinates of its closest corner to the origin. Hence cell $(0,0,0)$ corresponds to the unit cube with vertices $(0,0,0)$, $(0,0,1)$, $(0,1,0)$, $(1,0,0)$, $(0,1,1)$, $(1,1,0)$ and $(1,1,1)$. We denote by $\|\mathbf{m}\|_1$ the L_1 norm of the cell occupied by a module $\mathbf{m}$. For example, if $\mathbf{m}$ occupies $(x_\mathbf{m}, y_\mathbf{m}, z_\mathbf{m})$, then $\|\mathbf{m}\|_1 = |x_\mathbf{m}| + |y_\mathbf{m}| + |z_\mathbf{m}|$.

We assume that both configurations are contained in the positive orthant and that a corner of the bounding box of either configuration is the origin.

Our algorithm uses the **slice graph** from Fitch and Rus [11]. For a connected configuration C of n modules and $z_0 \in \mathbb{Z}$, the **slice** at height z is $C \cap \{z = z_0\}$ (the modules whose z-coordinate is equal to z_0). Each maximally connected component in a slice is called a **cluster**. Each cluster defines a vertex of the slice graph. Two vertices are connected if at least one module in each cluster share a face (see Fig. 2).

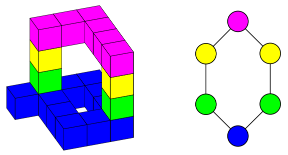


Figure 2: A configuration and its slice graph.

A cell c **dominates** another cell c' if c' is contained in the minimum axis aligned bounding box containing c and the origin. We also say that $c = (x, y, z)$ is **below** $c' = (x', y', z')$ if $x = x'$, $y = y'$ and $z < z'$ (similarly, we define **above** in a similar way. A module is called **compact** if every cell dominated by it is occupied. A module is called **quasi-compact** if every cell that is dominated by it and is not below it is occupied. A configuration or cluster is **compact** (resp., **quasi-compact**) if every module in it is compact (resp., quasi-compact). A cluster is **extremal** if not all of its modules are quasi-compact, and all adjacent modules above it (if any) are quasi-compact. (Note that a cluster above a non-quasi-compact cluster may be quasi-compact.)

2.1 General strategy.

A common technique in reconfiguration algorithms is to define a **canonical** configuration C^* and describe a sequence of moves that transform a given configuration C into C^*. Because the moves are reversible, this implies a solution for the reachability problem: Given two configurations C and C', one can obtain reconfiguration sequences from both to C^*, and then compose the reconfiguration from C to C^* with the reverse of the reconfiguration from C' to C^*. Instead of a single configuration, we use a class of canonical configurations: all compact configurations of n modules. The strategy is to compute a sequence of moves to transform C and C' to two configurations D and D' of this class. We then transform C into D, then D into D', and finally reconfigure D' to C'. Thus, our algorithm can be divided into two phases: reconfiguration between a given configuration and a compact one (Section 3); and reconfiguration between two compact configurations (Section 5).

3 Compacting configurations

In this section, we describe an algorithm we call COMPACTIFY, which reconfigures a given configuration into a compact one. As a preprocessing step, use LOCATEANDFREE (from [1]) to obtain six movable modules on the outer boundary. These modules will be referred to as **musketeers**.

Once done, the algorithm repeatedly picks an extremal cluster S and uses the musketeers to move every module in S without a lower neighbor to a position with a smaller z-coordinate. By doing so, S will merge with at least one cluster in the slice below. Initially, a module is only moved if the position below it is empty. Otherwise, it is left where it was. We call these leftover modules **stragglers**. We use another procedure FIX, to find a suitable position for the stragglers. The result is that every module previously in S either has a lower z-coordinate or becomes quasi-compact. By iterating over all clusters we will finish with a compact configuration or all modules in the $z = 0$ plane. Pseudocode of COMPACTIFY is presented in Alg. 1 (in Section A).

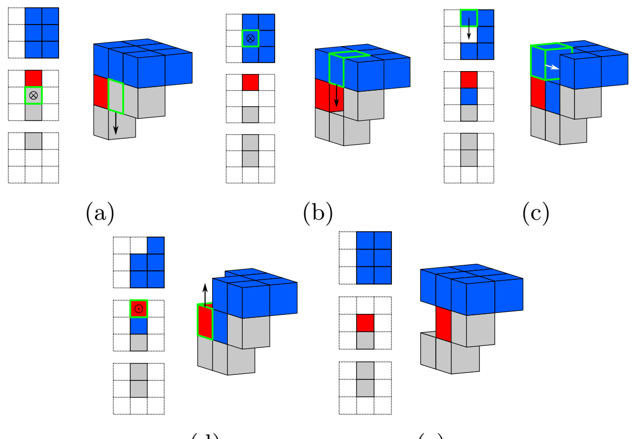
3.1 Obtaining Musketeers

The main role of the musketeers will be to provide temporary connectivity to nearby modules. We use a subroutine LOCATEANDFREE from [1] to gather them above the highest cluster.

Lemma 2 *A constant number k of musketeer modules can be positioned on the outer boundary of any configuration in at most $4kn + O(1)$ moves.*

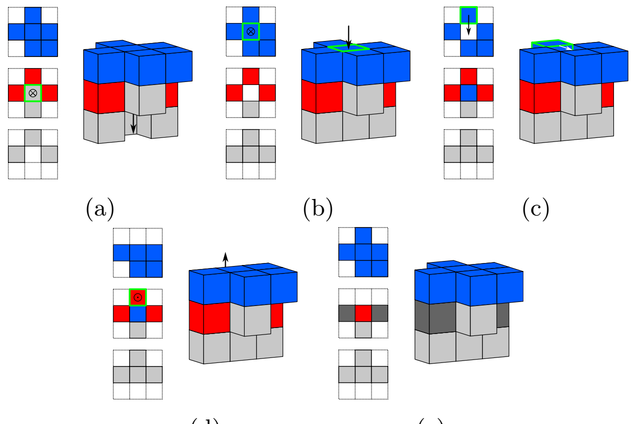
3.2 Lowering Modules

Given an extremal cluster S let S^* be the set of non-quasi-compact modules in S, and let T be a minimum Steiner tree of S^* using the quasi-compact modules in S (if any) as Steiner vertices. We process T in post-order, rooted arbitrarily, moving it to the z-layer below if it has no lower neighbor. Lowering is accomplished via the musketeer modules (shown in blue in Figs. 3 and 4).



(a) (b) (c) (d) (e)

Figure 3: The four moves used to lower a straight degree-2 module in an extremal cluster using the musketeers for connectivity. The left of each figure shows the top view of the three relevant z-layers. ⊙ and ⊗ represent arrows going into and out of the plane, respectively. The moving module is highlighted in green.



(a) (b) (c) (d) (e)

Figure 4: Four moves used to lower a degree-4 module.

In red we show auxiliary modules that stablish the connectivity between the processed and unprocessed parts of T. If $\mathbf{p}$ has a lower neighbor, it cannot move directly down, thus it becomes a straggler. (After completing our traversal of T we return and "fix" them.)

Since we visit modules in post order, we can guarantee the children of the module currently being processed (if any) have been already processed. This implies that, when processing a module, the neighbors in the same z-layer are either red or is the parent of the module in T. The exact strategy depends on the degree of $\mathbf{p}$ in T. The strategies for a straight degree-2 (child and parent share either x or y coordinate) and degree-4 is shown in Figs. 3 and 4, respectively. Degree-3 and bent degree-2 modules are handled similarly as degree-4. Details for other cases are in the full version [6].

Lemma 3 *Given a Steiner tree T of the non-quasi-compact modules of an extremal cluster S and 6 musketeer modules positioned above T as in Fig. 3a or Fig. 4a, let ℓ be the number of modules that were lowered and s be the number of remaining stragglers ($|T| = \ell + s$). Then, $17\ell + 13s$ moves suffice to move down by one unit every module in T that has no lower neighbor while maintaining connectivity.*

Proof sketch. Each module is visited on average 2 times by the musketeers: for every high-degree module in T (degree 3 or 4) there is a leaf that does not need to be visited. Each visit costs 6 moves totaling 12 moves per module. COMPACTIFY moves stragglers toward the leaves of T and each module is the base of a straggler at most once (13 moves per module). The lowered modules cost 4 extra moves depicted in Fig. 3 or Fig. 4. □

3.3 Handling straggler modules

After completing our post-order traversal, there is at least one straggler $\mathbf{m}$. Since $\mathbf{m}$ is not quasi-compact, there is at least one empty position in the cuboid region dominated by $\mathbf{m}$ and not directly below it. In FIX($\mathbf{m}$) we search for such an empty position p and effectively move $\mathbf{m}$ to p, either directly or through a chain of moves that fill p and leave m empty. The pseudo-code for FIX is presented in Alg 2 (in the Appendix) .

For a given cell $p = (x_p, y_p, z_p)$, we denote by $\square(p)$ the set of cells $(x_p - i, y_p - j, z_p)$ for all $0 \le i \le x_p$ and $0 \le j \le y_p$, i.e., the cells intersecting the minimum axis aligned horizontal square containing $(0, 0, z_p)$ and p. We define that a straggler is **fixable** if every *module* in $\square(\mathbf{m}) \setminus \{\mathbf{m}\}$ in the same cluster of $\mathbf{m}$ is quasi-compact. We first try moving $\mathbf{m}$ closer to the origin in $\square(\mathbf{m})$ (Fig. 5a). If we can't, either (i) $\mathbf{m}$ moves to a z-layer below it or becomes quasi-compact (and we are done); (ii) $\mathbf{m}$ is on the edge of the bounding box (and we can move through the plane $x = -1$ or $y = -1$ to an empty position); or (iii) $\mathbf{m}$ is blocked by other modules in the same layer that are quasi-compact (as in Fig. 5). Since $\mathbf{m}$ is fixable and non-quasi-compact, there is an empty position p dominated by $\mathbf{m}$ in column $(x_{\mathbf{m}} - 1, y_{\mathbf{m}}, .)$ or $(x_{\mathbf{m}}, y_{\mathbf{m}} - 1, .)$, adjacent to a module $\mathbf{q}$ in column $(x_{\mathbf{m}} - 1, y_{\mathbf{m}} - 1, .)$ (Fig. 5a). A sequence of moves fills p and frees $(x_{\mathbf{m}} - 1, y_{\mathbf{m}} - 1, z_{\mathbf{m}})$ (Fig. 5b). We can then fill that empty position while freeing $\mathbf{m}$'s cell (Fig. 5c).

Lemma 4 *If $\mathbf{m}$ is fixable, FIX($\mathbf{m}$) does not disconnect the configuration C, and performs at most $\|\mathbf{m} - p\|_1 + 2$ moves. After FIX($\mathbf{m}$), if $\mathbf{m}$ remained in the same z-layer, $\mathbf{m}$ is now quasi-compact.*

Proof. Since $\mathbf{m}$ is a straggler, if it is the only module moved by FIX($\mathbf{m}$), it is clear that connectivity is preserved. Lines 1–6 move $\mathbf{m}$ monotonically towards the origin, except potentially for the move that brings $\mathbf{m}$

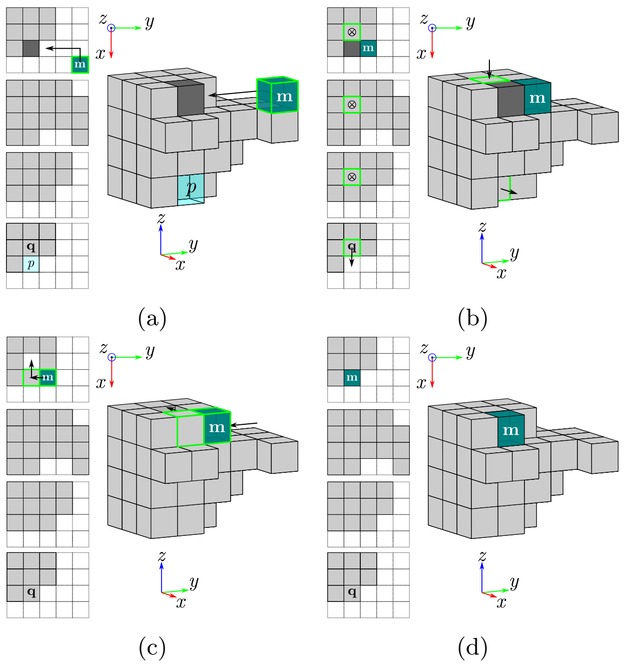


Figure 5: Illustration of the moves in Fix (a) line 2, (b) lines 9–11, and (c) line 12. (d) After Fix, $\mathbf{m}$ becomes quasi-compact.

outside of the bounding box (line 6). The bound on the moves holds in this case, and if $\mathbf{m}$ moved down (in z) the claim holds.

It remains to prove the claim for case (iii), i.e., lines 8–12. By the fact that $\mathbf{m}$ is fixable, when we reach line 8, cells $\mathbf{m}+(-1,0,0)$, $\mathbf{m}+(0,-1,0)$ and $\mathbf{m}+(-1,-1,0)$ must be filled by quasi-compact modules. This implies that all cells with nonnegative coordinates strictly smaller than $\mathbf{m}$'s are full. Since $\mathbf{m}$ is not yet quasi-compact, there must exist at least one empty position dominated by $\mathbf{m}$ and not directly below it. Indeed, all such empty cells must be in columns $(x_\mathbf{m}-1, y_\mathbf{m}, .)$ or $(x_\mathbf{m}, y_\mathbf{m}-1, .)$, otherwise $\mathbf{m}+(-1,0,0)$ or $\mathbf{m}+(0,-1,0)$ would not be quasi-compact. Then, p exists and $\mathbf{q}$ is a compact nonarticulate module. Thus, line 9 maintains connectivity. Note that by the choice of p, every cell edge-adjacent in the same z-layer to a module moved by line 11 is full except for the cell in the column $x_\mathbf{m}$. Thus, the neighborhood of the moving module remains connected and line 11 maintains connectivity. When we reach line 12, the cell at $(x_p, y_p, z_\mathbf{m}-1)$ is full (by the choice of p). The fact that $\mathbf{m}$ is a straggler implies that every edge-adjacent cell above $\mathbf{m}$ have been already processed by Fix, so they are quasi-compact and thus do not require $\mathbf{m}$ or $(x_p, y_p, z_\mathbf{m})$ for connectivity. Then, both moves in line 12 maintain connectivity. Except for line 9 and the move from $(x_p, y_p, z_\mathbf{m})$ to $\mathbf{m}+(-1,-1,0)$ (line 12), all the moves are monotonic in the direction $p-\mathbf{m}$. Thus the total number of moves is as claimed. □

3.4 Compactify for planar configurations

With a few small adjustments, our algorithm can be applied to planar configurations (when all modules lie in the $z=0$ plane). Note that we allow using planes $z=1$ and $z=-1$ to move musketeers and stragglers. The clusters are now defined by the y-slice graph (maximal components induced by a fixed y coordinate). Lowering modules follows the same principle, except that we lower the y-coordinates of the modules (cases are simpler, as modules have degree 2 or less within the cluster). Fix also becomes substantially simpler as stragglers can simply move to the closest dominated empty cell through the $z=-1$ plane.

4 Analysis of Compactify

Lemma 5 Compactify(C) *transforms a configuration C into a compact configuration maintaining connectivity.*

Proof sketch. The claim is mostly established by Theorem 3 and lemma 4. We process the stragglers in increasing order of L_1 norm, guaranteeing that the first straggler is fixable. By Lemma 4, the straggler either moves down or becomes quasi-compact, making the subsequent processed straggler fixable. □

Lemma 6 *A sequence of* $17\sum_{\mathbf{m}\in C}\|\mathbf{m}\|_1 + 60n + O(1)$ *moves suffices to transform a configuration C with n modules into a compact one.*

Proof sketch. By Lemma 3, we can charge 17 moves to the decrease of the z-coordinate of the lowered modules. By Lemma 2 2 we can obtain six musketeers in $24n+O(1)$ moves. While traversing C, every musketeer walks on the bottom face of each module of a cluster at most once, and walks on the side face of one module exactly once (see Fig. 6 for example). This amounts $6n+O(1)$ musketeer movements (each cluster is only traversed in this way once). In the planar version of Compactify the musketeers can move in the positive y-direction using the bottom faces, which adds $6n$ moves. The total amounts $12n+O(1)$ musketeer movements. The musketeers may walk on the top face of a module at most two times, $24n$ moves. Overall, the musketeers preform $24n+12n+O(1)=36n+O(1)$ moves. Combined with the bound from Lemma 2, we get a total $24n+36n+O(1)=60n+O(1)$ moves. □

Lemma 7 Compactify *has runtime that is within an amortized $O(1)$ factor of the moves performed.*

Proof. Whenever a musketeer moves, it is either following a constant size schedule of moves such as Fig. 3, or is traveling along a shortest path from one cluster to another. The former case clearly only requires $O(1)$

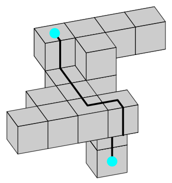


Figure 6: The path of an upwards traversal of musketeers from one cluster to another. Their start and end positions marked by blue circles.

computation, and in the latter, computing the path requires time proportional to the length of the path, hence amortized $O(1)$ computation per movement. The only section of this algorithm where an $O(1)$ computation cost is not simple is FIX, when we must locate an appropriate empty position p. We show if each module stores a constant amount of information, these positions can be located efficiently. Assume that initially each module $\mathbf{m}$ has knowledge of the following three properties: 1) Is $\mathbf{m}$ compact? 2) Is $\mathbf{m}$ quasi-compact? 3) Is every position of $\square(\mathbf{m})$ filled?

Note, each of these are transitive, and, to answer these questions, $\mathbf{m}$ need only query its neighbors with smaller sum of coordinates. Consequently, if $\mathbf{m}$ moves, recomputing if $\square(\mathbf{m})$ is full takes only $O(1)$ time. If $\mathbf{m}$ becomes (quasi-)compact it may be the case that many modules also become (quasi-)compact. While this may affect many more modules than the number of moves performed, a module can become (quasi-)compact only once. Therefore, in aggregate these updates will only take $O(n)$ time, which can then be amortized over the number of moves to $O(1)$ time.

If $x_{\mathbf{m}} = 0$ or $y_{\mathbf{m}} = 0$ (case(ii)), FIX attempts to move it to an empty dominated position p. If $\mathbf{m}$ is not compact, then p exists. Therefore we can determine if p exists in $O(1)$ time. If $\mathbf{m}$ is not compact, we can locate p with $||\mathbf{m} - p||_1$ queries: Checking positions directly below $\mathbf{m}$ (from $\mathbf{m}$) we will either find an empty position (and we are done) or a compact module by the fact that $\mathbf{m}$ is fixable. We have found the z coordinate of p and we can find it by looking in the $-x$ or $-y$ direction (note that $\mathbf{m}$ and p lie in the same plane $x_{\mathbf{m}} = 0$ or $y_{\mathbf{m}} = 0$).

In case (iii), since we know p is either in column $(x_{\mathbf{m}} - 1, y_{\mathbf{m}}, .)$ or $(x_{\mathbf{m}}, y_{\mathbf{m}} - 1, .)$, we can find it with $||\mathbf{m} - p||_1$ queries with a linear search alternating between the two columns. □

5 Reconfiguring Between Compact Configurations

Lemma 8 *Given two compact configurations C_1 and C_2 with n modules where all modules lie in the positive xyz orthant, we can reconfigure one into the other by using at most $\sum_{\mathbf{m} \in C_1} \|\mathbf{m}\|_1 + \sum_{\mathbf{m} \in C_2} \|\mathbf{m}\|_1$ sliding moves.*

Proof. We use induction on $|C_1 \setminus C_2|$. The base case is when $|C_1 \setminus C_2| = 0$ and thus $C_1 = C_2$ and we are done. Otherwise, we claim that there is a module $\mathbf{m}_1 \in C_1 \setminus C_2$ whose cell is not dominated by any other module in $C_1 \cup C_2$. We prove this by contradiction: assume that every module if $C_1 \setminus C_2$ is dominated by some module in $C_1 \cup C_2$. Because domination establishes a partial order, there must exist a module $\mathbf{m} \in C_1 \setminus C_2$ that us dominated by a module in $\mathbf{m}' \in C_1 \cap C_2$. That is, we have found a module $\mathbf{m}$ is only present in C_1 and is dominated by $\mathbf{m}' \in C_1 \cap C_2$ that is present in both C_1 and C_2. However, this is a contradiction to the fact that C_2 is compact. The interesting property of $\mathbf{m}_1$ is that $C_1 \setminus \{\mathbf{m}_1\}$ is compact.

Using a symmetric argument, we show that there exists a cell $\mathbf{m}_2 \in C_2 \setminus C_1$ so that every dominated cell is occupied in $C_1 \cap C_2$. For contradiction, assume no such module exists. Let $\mathbf{m}$ be the module in $C_2 \setminus C_1$ with smallest $\|\mathbf{m}\|_1$. Since C_2 is compact, all cells that $\mathbf{m}$ dominates are occupied in C_2. Let $m' \in C_2 \setminus C_1$ be a module dominated by $\mathbf{m}$, which must exist by assumption. Then $\|\mathbf{m}'\|_1 < \|\mathbf{m}\|_1$, contradicting the choice of $\mathbf{m}$. We conclude that $C_1 \cup \{\mathbf{m}_2\}$ is compact. Furthermore, the two claims combined imply that $C_1' = C_1 \cup \{\mathbf{m}_2\} \setminus \{\mathbf{m}_1\}$ is compact. Then $|C_1 \setminus C_2| = |C_1' \setminus C_2| + 2$ as desired.

To complete the proof we must show that the reconfiguration can be done in the claimed number of moves. Note that $\mathbf{m}_1$ and $\mathbf{m}_2$ must both be in the outer boundary of C_1 and C_1' respectively. By definition of compact, the boundary of $C_1 \setminus \{\mathbf{m}_1\}$ that is not on the $x = 0, y = 0$ or $z = 0$ planes is an x-, y-, and z-monotone surface. Thus, the shortest path from $\mathbf{m}_1$ to $\mathbf{m}_2$ on this surface is also monotone and, hence its length is upper bounded by $\|\mathbf{m}_1\|_1 + \|\mathbf{m}_2\|_1$. □

6 Conclusion

Algorithmic bounds have been derived from the number of modules [2,9], to dimensions of bounding boxes [1,5] and eventually the sum of coordinates (this paper and [12]). We note that, none are optimal if the starting and target configurations are far from compact but very similar. Is there a better parameter to bound the length of reconfiguration between configurations? Such a bound is also important for approximation algorithms, of which there are none. Note that in general, shortest reconfiguration is known to be NP-complete [5].

Another avenue of research would be parallelization where we look into the **makespan** (i.e., time required to execute all moves when we allow parallel moves).

References

[1] Z. Abel, H. A. Akitaya, S. D. Kominers, M. Korman, and F. Stock. A universal in-place reconfiguration algorithm for sliding cube-shaped robots in a quadratic number of moves. In *SoCG*, pages 1:1–1:14, 2024.

[2] Z. Abel and S. D. Kominers. Universal reconfiguration of (hyper-)cubic robots. *arXiv preprint*, 2008.

[3] H. A. Akitaya, E. M. Arkin, M. Damian, E. D. Demaine, V. Dujmović, R. Flatland, M. Korman, B. Palop, I. Parada, A. v. R. Renssen, and V. Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $O(1)$ Musketeers. *Algorithmica*, 83:1316–1351, 2021.

[4] H. A. Akitaya, E. D. Demaine, A. Gonczi, D. H. Hendrickson, A. Hesterberg, M. Korman, O. Korten, J. Lynch, I. Parada, and V. Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In *SoCG*, volume 189, pages 10:1–10:20, 2021.

[5] H. A. Akitaya, E. D. Demaine, M. Korman, I. Kostitsyna, I. Parada, W. Sonke, B. Speckmann, R. Uehara, and J. Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. In *SWAT*, pages 4:1–4:19, 2022.

[6] H. A. Akitaya, M. Korman, and F. Stock. Input-sensitive reconfiguration of sliding cubes. *arXiv preprint*, 2025.

[7] G. Aloupis, S. Collette, M. Damian, E. D. Demaine, R. Flatland, S. Langerman, J. O'Rourke, S. Ramaswami, V. Sacristán, and S. Wuhrer. Linear reconfiguration of cube-style modular robots. *Computational Geometry*, 42(6-7):652–663, 2009.

[8] G. Aloupis, S. Collette, E. D. Demaine, S. Langerman, V. Sacristán, and S. Wuhrer. Reconfiguration of cube-style modular robots using $o(\log n)$ parallel moves. In *ISAAC*, pages 342–353. Springer, 2008.

[9] A. Dumitrescu and J. Pach. Pushing squares around. *Graphs and Combinatorics*, 22(1):37–50, 2006.

[10] A. Dumitrescu, I. Suzuki, and M. Yamashita. Motion planning for metamorphic systems: Feasibility, decidability, and distributed reconfiguration. *IEEE Transactions on Robotics and Automation*, 20(3):409–418, 2004.

[11] R. Fitch, Z. Butler, and D. Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *IROS*, volume 3, pages 2460–2467. IEEE, 2003.

[12] I. Kostitsyna, T. Ophelders, I. Parada, T. Peters, W. Sonke, and B. Speckmann. Optimal in-place compaction of sliding cubes. In *SWAT*, pages 31:1–31:14, 2024.

[13] C. Sung, J. Bern, J. Romanishin, and D. Rus. Reconfiguration planning for pivoting cube modular robots. In *ICRA*, pages 1933–1940. IEEE, 2015.

A Pseudocode of Compactify and Fix

Algorithm 1 Compactify(C)

```
1: Compute the slice graph of $C$.
2: while $C$ is not quasi-compact (ignoring slice $z = 0$) do
3:     Let $R$ be the cluster where the musketeers currently sit. DFS on the slice graph from $R$, preferring upward edges (in the $z$ direction). Let $S$ be the first extremal cluster (with possibly $S = R$).
4:     Move the musketeers to $S$.
5:     Lower non-quasi-compact modules in $S$ as in Section 3.2.
6:     Let $M$ be the set of stragglers $\mathbf{m}$ sorted by $\|\mathbf{m}\|_1$.

7:     for $\mathbf{m} \in M$ do
8:         Fix($\mathbf{m}$)
9: while $\exists\mathbf{m}$ where $\mathbf{m}+(0,0,-1)$ is empty and $z_{\mathbf{m}} > 0$ do
10:    Slide $\mathbf{m}$ down
11: if $C$ is not compact then
12:    Apply the planar version of Compactify on slice $z = 0$
```

Algorithm 2 Fix($\mathbf{m}$)

```
1: while $\mathbf{m}$ can move monotonically in the direction $-\mathbf{m}$ do
2:     Move $\mathbf{m}$ closer to the origin
3:     if $\mathbf{m}$ moved to a lower $z$-coordinate or became quasi-compact then
4:         return
5: if $x_{\mathbf{m}} = 0$ or $y_{\mathbf{m}} = 0$ and $\mathbf{m}$ is not compact then
6:     Move $\mathbf{m}$ (through the plane $x = -1$ or $y = -1$) to a dominated empty cell $p$
7: else
8:     Let $p$ be the closest dominated empty position to $\mathbf{m}$ where $x_p = x_{\mathbf{m}} - 1$ (resp., $y_p = y_{\mathbf{m}} - 1$), and $\mathbf{q}$ be the module at $p + (0,-1,0)$ (resp., $p + (-1,0,0)$)
9:     Move $\mathbf{q}$ to $p$ and let $q$ be the cell left empty originally occupied by $\mathbf{q}$
10:    for $i \in \{1, \ldots, (z_{\mathbf{m}} - z_q)\}$ do
11:        Move the module at $q+(0,0,i)$ to $q+(0,0,i-1)$ via slide
12:    Slide the module at $(x_p, y_p, z_{\mathbf{m}})$ to $\mathbf{m}+(-1,-1,0)$ and slide $\mathbf{m}$ to $(x_p, y_p, z_{\mathbf{m}})$
13: return
```

Reverse Rush Hour is NP-Complete

Jonathan Gabor[*] Aaron Williams[†]

Abstract

Rush Hour is a well-known grid-based sliding block puzzle by ThinkFun. In this classic puzzle, a number of vehicles are placed within a rectangular parking lot. Each vehicle is horizontal (1-by-k) and can slide horizontally, or vertical (k-by-1) and can slide vertically, and there is one special red vehicle. The goal is to slide the vehicles so that the red vehicle can drive out of the parking lot.

We introduce a 'reverse' version of *Rush Hour*. The vehicles start in traffic lanes outside of the parking lot, and the goal is to slide them into the parking lot. There is no limit on the number of times each vehicle can move in the parking lot and/or lanes. Despite this potential source of unboundedness, we prove that the problem is in NP. In fact, the certificates are simple enough to be used in newspapers like Don Rubin's original *Lots of Luck* puzzles. We also prove that the decision problem is NP-hard via a reduction from 3-dimensional matching. The reduction is parsimonious in terms of the final packing of vehicles, implying that the associated ASP (Another Solution Problem) is also NP-complete.

1 Introduction

Rush Hour is a sliding block puzzle that was invented by Nob Yoshigahara [17] in the late 1970s. The puzzle was initially sold in Japan under the name *Tokyo Parking*. It was branded as *Rush Hour* by Binary Arts which later changed its name to ThinkFun [58, 59].

Challenges are played on a 6-by-6 *parking lot* with an 'exit' on the right side of the third row. Each *vehicle* is placed vertically or horizontally to occupy consecutive cells without overlap. More specifically, the vehicles are *cars* of length 2 and *trucks* of length 3. Each move slides a vertical vehicle up or down, or a horizontal vehicle left or right. The goal is to slide a red target car to the exit.

Rush Hour is notoriously difficult. Solutions often require vehicles to move back-and-forth (see Figure 1) and the original 6-by-6 grid supports challenges needing up to 93 moves [18]. In fact, it has *exponentially long* challenges (i.e., requiring exponentially many moves to solve) and it was one of the first puzzles to be shown PSPACE-complete [34]. The instructions caution that

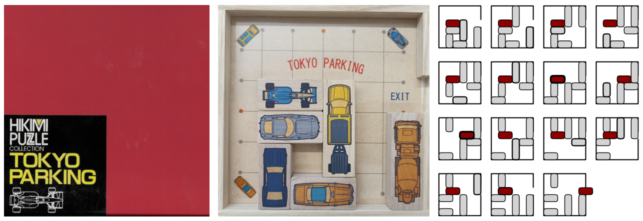


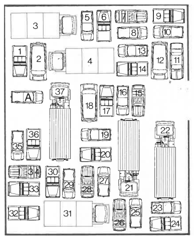
Figure 1: Puzzle No. 1 in *Tokyo Parking* by Nob Yoshigahara and the shortest solution in 14 moves.

"I cannot be held responsible if this causes you to miss out on promotion, fail a class, or your girlfriend runs away from you" (そのせいで出世が遅れたり、単位を落したり、恋人に逃げられたとしても、僕は責任は負えない。).

The puzzle has sold millions of physical copies along with additional decks challenge cards and sequels like *Railroad Rush Hour*. The sliding mechanism is well-suited to mobile touchscreens, which led to a spate of copycats including *ParkingLot* by Quetouch and *Unblock Me* by Kiragames; the former predated Apple's *App Store* as a jailbroken app, and the latter has been downloaded over 100 million times on iOS / Android.

1.1 The Real Origin

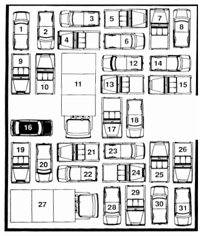
Apple and Quetouch were sued in 2009, not by ThinkFun, but by prolific puzzle designer Don Rubin. As stated in *Apple v. Rubin* [51], Rubin's *The Real Puzzle* article on October 1, 1977 in Boston's weekly newspaper *The Real Paper* featured the puzzle *Lots of Luck* that was identical in idea and expression to *ParkingLot*. A reprinting of Rubin's puzzle appears in Figure 2a.



(a) Puzzle E16 [45].

11	12	13	14	15	16
21	25	26	31	29	30
37	A	1	2	4	17
18	A	37	29	30	31
25	26	21	A	17	18
5	6	7	8	15	16
13	14	11	12	A	

(b) Solution [45].



(c) Version 2 [53].

Figure 2: *Lots of Luck* puzzles by Don Rubin.

Yoshigahara probably wasn't an avid reader of *The Real Paper*, but he may have seen *Lots of Luck* reprinted in Rubin's *What's the Big Idea?* in 1979 [52]. The real

[*]Department of Computer Science, Cornell, jpg264@cornell.edu
[†]Department of Computer Science, Williams College, aaron.williams@williams.edu

origin of the puzzle is unclear due to a lack of precise information on *Tokyo Parking*'s release. But independent creation seems quite likely as the puzzle is a natural variation of *Klotski* and its various commercializations (e.g., *Dad's Puzzler* [45]) during the 20th century.

Rubin created further challenges under various names (e.g., *Thanks – A Lot* (1980), *Jeepers* (1984) [45]), but it never became a recurring staple like *Crossword* or *Magic Squares*. This is likely due to its unwieldy solutions: Figure 1's solution style is too lengthy for syndication, while Figure 2b's is unsightly and difficult to follow. Furthermore, solutions may require trial and error (or "lots of luck!") and are not unique. Thus, step-by-step deduction of **the** solution isn't always possible, which is a tenet of many well-designed logic puzzles [10, 11].

1.2 Reverse Rush Hour

We introduce a puzzle that can be described as *Reverse Rush Hour*. It uses the same pieces and physics as *Rush Hour*, however, the playfield has *traffic lanes* that extend indefinitely outside of the parking lot's rows and columns. All of the vehicles start outside of the parking lot in one of these lanes, and the goal is to slide them into the parking lot. A vehicle can move any number of times, including dalliances back into its traffic lane or into the lane on the opposite side of the parking lot.

Unlike most sliding block puzzles [44], these puzzles do not have exponentially long challenges. In fact, each vehicle needs to move at most twice in a solution. Moreover, players and puzzle designers can simply draw the final vehicle positions as **the** solution. Nevertheless, the puzzles are still challenging: we prove that the decision problem is NP-complete. In fact, a related problem is ASP-complete, meaning that it is difficult to design challenges with a unique solution. This matches classic pencil and paper puzzles like Nikoli's *Sudoku* [60].

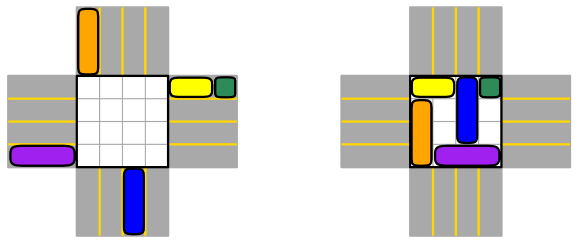

(a) Challenge S. (b) A solved state.

Figure 3: A solvable instance of *Reverse Rush Hour*. (Hint: Start by sliding the blue truck up 7 cells and fully into its lane above, then park all of the vehicles inside the parking lot as in (b) with blue last.) This challenge cannot be solved by sliding each vehicle only once (i.e., RevRush1(S) is no) as some vehicle will be blocked. Note that S has multiple shortest move sequences, but (b) is its unique final state or parking packing.

1.3 Outline

We define *Reverse Rush Hour* in Section 3, and prove it is in NP in Section 4. Sections 5–6 prove it is NP-hard (and hence NP-complete) by a reduction from 3-dimensional matching. Section 7 has final remarks, including an observation that our reduction is parsimonious (up to the final packing) so it is NP-complete to determine if a challenge has a unique solution [60]. We begin with the complexity of 1-player puzzles.

2 Computational Complexity of 1-Player Puzzles

To analyze the computational complexity [54] of a puzzle, we must first generalize it to allow arbitrarily large instances. The details of the generalization will affect its complexity. For example, it is natural to consider *Rush Hour* either on arbitrary square or rectangular grids. Likewise, a generalization could use the same vehicles (i.e., 2-by-1 cars and 3-by-1 trucks) or arbitrarily long (i.e., k-by-1) or even thick vehicles as in Figure 2.

Generalizations of well-known 1-player puzzles are often NP-complete or PSPACE-complete[1]. The first question is if the generalization is *bounded* [44]. In other words, can every yes instance be solved in a polynomially many 'moves' relative to the instance's size? If so, a suitable move sequence provides a certificate for membership in NP as it can be verified in polynomial-time.

Next we survey bounded and unbounded puzzles, including previous results from this conference series and new frameworks that have broad application.

2.1 Bounded Puzzles

When a puzzle is bounded, there is often a limited resource that is diminished by each move. For example, a grid-based puzzle may require filling in empty cells. The best source of these puzzles is the Japanese company Nikoli. Researchers are quick to analyze new Nikoli puzzles [3, 50] to establish a short *Nikoli gap* [2]. Modifying the rules to more precisely determine the source of complexity is also popular [1, 30]. A recent step forward on the ASP-completeness of grid-based Hamiltonicity problems led to 38 Nikoli puzzles being proven to be NP-complete [14].

Determining the complexity of a bounded puzzle can unearth new results on classic problems. For example, *Buttons & Scissors* is a bounded puzzle in which items are removed from a grid [41]. The study of this puzzle [15, 16] helped sharpen polynomial-time algorithms for perfect edge and triangle matchings in a graph [19].

[1]Some puzzles are naturally easier, while others are more difficult. For example, *Lights Out* is in P [44], while *Baba Is You* is undecidable [37]. But the generalization details matter (e.g., see various results on *Super Mario Bros.* [4, 5, 31, 49]).

2.2 Unbounded Puzzles with or without an Agent

The most natural notion of a 'move' may lead to unboundedness. For example, some classic video games support levels requiring an exponential number of controller presses (✥) to complete. These levels may force the player to count through all n-bit binary strings (in Gray code order) [36, 39, 40] and they only invalidate move sequences as suitable certificates for NP membership. Iterating through 2^n different possibilities is often embedded into reductions from the TQBF problem [54].

PSPACE-completeness may be proven in two stages, with an initial investigation proving NP-hardness (e.g, *Super Mario Bros.* [4, 31] and *Block Dude* [7, 9]). Less popular puzzles may stay NP-hard (e.g., *Megalit* [38]).

The history of PSPACE-complete 1-player puzzles has been dominated by box pushing and sliding blocks. Box pushing games have an *agent* (i.e., the player controls an avatar) while sliding block puzzles do not.

The massively popular box pushing game *Sokoban* was first proven NP-hard [33, 35, 57] then PSPACE-complete [20] even without walls [43]. Variations often under the `Push` moniker have been studied extensively [23, 26, 28, 29, 32]. The recent motion planning with gadgets framework [25, 27, 48] has led to PSPACE-hardness for various simple game mechanics [6, 8, 24].

While ThinkFun sometimes mentions a protagonist named "(Rush Hour) Joe" the puzzle does not actually have an agent. The generalization first proven PSPACE-complete by Flake and Baum [34] involves cars and trucks on an n-by-n grid in which the exit can be on any row. Their proof inspired the *nondeterministic constraint logic* (NCL) framework [42] that has been applied to dozens of puzzles [44]. An open problem in [55] was answered in [13]: *Rush Hour* is PSPACE-complete when restricted to fixed blocks and 1-by-1 vehicles. This problem inspired the iPhone game *Subway Shuffle* which is PSPACE-complete [21] (also see [12]) and is commercially available as *Athena* by *Popular Playthings*.

3 Reverse Rush Hour

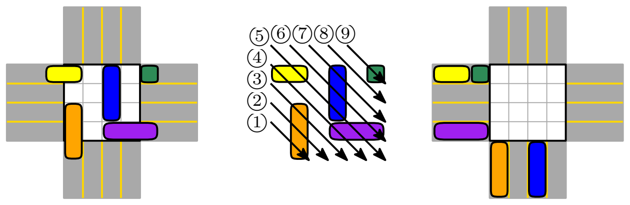
Now we introduce terminology for discussing *Reverse Rush Hour* puzzles and provide some basic results. Then we specify three decision problems.

Each *vehicle* is one cell wide. The playfield is a *parking lot* with r rows and c columns with *lanes* extending from each. A *state* S has m vehicles oriented horizontally and/or vertically in the playfield without overlap, and it is *normalized* if there are no gaps between the vehicles in the lanes (i.e., they are as close to the parking lot as possible). An *initial state* is normalized with all vehicles completely in lanes, and it is *k-sided* if each vehicle is in one of k sides. In particular, a *bottom-left state* is 2-sided with vehicles in the bottom and left lanes. An initial state is *full* if the area of the vehicles is $r \cdot c$.

Two states S and S' are *consistent* if they have the same relative order of vehicles on each row and column. By imagining the underlying state graph, we say that S and S' are *connected* if there is a sequence of vehicle slides that transforms S into S'. We now point out that it is always possible to slide the vehicles into the (unique) bottom-left state. Moreover, this can be done by moving each vehicle at most once (see Figure 4).

Lemma 1 *Every state S is connected to a unique bottom-left state S'. Moreover, S can be transformed into S' by moving each vehicle at most once.*

Proof. We proceed by a simple shelling argument. Each vehicle v is associated with the coordinate (r_v, c_v) of the bottom-left cell that it occupies. We create a total order on the vehicles by the value $r_v - c_v$ and then r_v. In other words, we order the vehicles by lowest down-right pointing diagonals (see Figure 4b). Moving each vehicle from its position in S to its position in S' requires only one slide, so long as we follow this total order. □



(a) State. (b) Order. (c) Bottom-left.

Figure 4: Transforming state (a) into the unique consistent 2-sided state in which the vehicles are in the bottom and left lanes (c). The vehicles slide to their final position based on their bottom-left cell as per Lemma 1: orange, yellow, purple, blue, green.

A state is a *solution* (or *parking packing*) if all of the vehicles are fully inside of the parking lot. Given an initial state S, the decision problem RevRush(S) asks if a sequence of moves connects S to a solution state. The decision problem RevRush1(S) is similar, but each vehicle is allowed to move at most once. Likewise, RevRush2(S) allows two moves per vehicle.

Corollary 2 *If S is a bottom-left state, then* RevRush(S) = RevRush1(S). *In other words, moving vehicles once is sufficient for solving challenges where every vehicle is in a bottom or left lane.*

Reverse Rush Hour was invented by the second author in early 2018 while working at Bard College at Simon's Rock. That summer the idea was pitched to ThinkFun who showed some initial interest but did not follow up on it. Central to the pitch was a touchscreen app written by Simon's Rock student Zoe Offermann. For this reason, "(Rush Hour) Zoe" is its protagonist.

4 Membership in NP

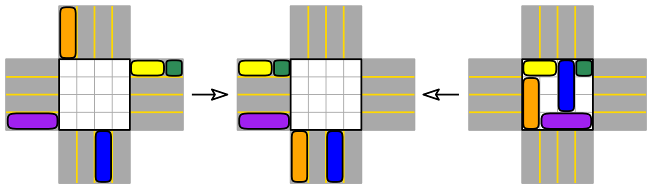
Next we show that the decision problems are in NP. This is immediate for RevRush1 and RevRush2 as move sequences are suitably short certificates. Now we will prove that this is also true for RevRush.

Recall consistent and connected pairs from Section 3. We prove that these concepts are the same by applying Lemma 1 twice, with the second application reversed. The approach is illustrated in Figure 5. Then we obtain Corollary 4 from Theorem 3 by letting S' be any solution state (i.e., a consistent parking packing).

Theorem 3 *Two states S and S' are connected, if and only if, they are consistent. Moreover, S can be translated into S' by moving each vehicle at most twice.*

Proof. By Lemma 1 we can transform S into its bottom-left state by moving each vehicle at most once. Likewise, S' can be transformed into the same bottom-left state by moving each vehicle once. By reversing the second move sequence, we can transform S into S'. □

Corollary 4 RevRush(S) *is yes, if and only if, S has a consistent parking packing S'. Moreover, moving vehicles twice is sufficient:* RevRush(S) = RevRush2(S).



(a) State S. (b) Bottom-left. (c) State S'.

Figure 5: Transforming S to S' by moving vehicles at most twice by Theorem 3. Since S' is a solution state, this gives an alternate solution to Figure 3's challenge.

While sliding physical (or virtual) vehicles may be helpful and enjoyable, Corollary 4 shows that these movements are not necessary to certify a solution. More specifically, the existence of a consistent parking packing is sufficient. This is because Theorem 3 gives an efficient linear-time algorithm to convert a solution state into a move sequence that creates it: start by sliding all of the vehicles into the bottom and left lanes, then proceed by the shelling process given by Lemma 1 in reverse.

This observation has several practical implications for *Reverse Rush Hour* puzzles. For example, the solution provided to each challenge can be a parking plan and not a move sequence. In other words, we can avoid solutions like Figure 2b. In addition, we can generate random puzzles by creating any parking plan, then by moving the vehicles outward to the lanes. Of course, players will not enjoy these puzzles as much as well-designed puzzles that can be solved step-by-step by logical deduction [10].

4.1 Parking or Packing?

Another way to interpret Corollary 4 is that *Reverse Rush Hour* is essentially a geometric packing problem on a grid [22,47]. Given vertical lengths for each column V and horizontal lengths for each row H, the *parking packing* decision problem ParkPack(V, H) asks if there is a suitable packing. For example, the challenge S in Figure 3 translates to the yes instance ParkPack(V, H) for vertical lengths $V = (2; ; 3;)$ and horizontal lengths $H = (2,1; ; ;)$. The ParkPack and RevRush decision problems differ when considering unique solutions.

5 3-Dimensional Matching

Let $X_m = \{x_1, \ldots, x_m\}$ and similarly define Y_m and Z_m. The *3-dimensional matching* decision problem 3DM takes as input a set of n triples $T \subseteq X_m \times Y_m \times Z_m$ and asks if it has a *perfect matching* $M \subseteq T$. In other words, can M be chosen so that each member of the *ground set* (i.e., $X_m \cup Y_m \cup Z_m$) appears in exactly one triple in M? Note that $|M| = m$ is a necessary but not sufficient condition to being a perfect matching. Also note that the size of the ground set is $3m$.

The 3DM problem was one of the first problems to be proven to be NP-complete [46]. It has since been used to establish the NP-hardness of many problems including Nonogram [56]. A sample instance of 3DM with $n = 6$ triples and its solution from [56] appears in (1). In particular, the columns of M illustrate that it is a perfect matching of $X_m \times Y_m \times Z_m$ for $m = 3$.

$$\begin{array}{llll}
T = \{\langle x_1, y_1, z_1\rangle, & M = \{ & \text{(i)} & (1)\\
\quad\langle x_1, y_2, z_3\rangle, & \quad\langle x_1, y_2, z_3\rangle, & \text{(ii)} & \\
\quad\langle x_2, y_3, z_1\rangle, & \quad\langle x_2, y_3, z_1\rangle, & \text{(iii)} & \\
\quad\langle x_3, y_1, z_2\rangle, & \quad\langle x_3, y_1, z_2\rangle & \text{(iv)} & \\
\quad\langle x_3, y_2, z_1\rangle, & & \text{(v)} & \\
\quad\langle x_3, y_3, z_3\rangle\} & \quad\} & \text{(vi)} &
\end{array}$$

We let $\#(g, T)$ be the number of triples with ground set element g in T. For example, $\#(y_2, T) = 3$ in (1).

6 Reducing 3DM to RevRush1

In this section we provide a polynomial-time reduction from 3DM to RevRush1. Furthermore, the reduction creates full bottom-left instances of *Reverse Rush Hour*. Hence, RevRush1 is NP-complete even when restricted to these instances. The reduction is defined below and is illustrated in Figure 6a.

Definition 1 *Let $T = \{t_1, t_2, \ldots, t_n\} \subseteq X_m \times Y_m \times Z_m$ be an instance of the* 3DM *problem. From T we create a full bottom-left instance of* RevRush1 *as follows.*

- *The parking lot has n rows and $6m+1$ columns. Consecutive columns pairs (excluding column 1) are associated with successive ground set elements. For example, columns 2 and 3 are associated with x_1.*
  - *Columns 2 to $2m+1$ are x-columns. In particular, the x_i columns are $2i$ and $2i+1$.*
  - *Columns $2m+2$ to $4m+1$ are y-columns. In particular, the left y_j column is $2m+2j$.*
  - *Columns $4m+2$ to $6m+1$ are z-columns. In particular, the left z_k column is $4m+2k$.*
- *Vertical vehicles of size 1 are in lanes below the following columns.*
  - *Both of the x_i columns have 1 vehicle.*
  - *The left column of y_j has $\#(y_j,T)-1$ vehicles.*
  - *The left column of z_k has $\#(z_k,T)-1$ vehicles.*
- *Horizontal vehicles are in the left lane of each row.*
  - *Row r has vehicles of the following sizes from left-to-right where $t_r = \langle x_i, y_j, z_k\rangle$.*
    - $2i-1$
    - $2(m+j-i)$
    - $2(m+k-j)-1$
    - $2(m-k)+1$.

Observation 1 *The* RevRush1 *instances are full.*

Proof. There are $n\cdot(6m+1) = 6mn+n$ cells in the parking lot. Each row has horizontal vehicles whose size adds up to $6m-1$, so the total size of the horizontal vehicles is $6mn-n$. But the x-columns collectively have $2m$ vehicles, the y-columns have $n-m$ vehicles, and the z-columns have $n-m$ vehicles, and they all have size 1, so the total size of the vertical vehicles is $2n$. □

Figure 6a explains that each row must have exactly two vertical vehicles to complete it. Lemma 5 argues that this can be done in exactly one of two ways. These two ways are referred to as *xx-satisfied* and *yz-satisfied* based on the specific vertical vehicles that are used to complete it. These two options are shown in Figure 6.

Lemma 5 *The reduction creates instances of* RevRush1 *in which each row must be xx-satisfied or yz-satisfied in any solution.*

Proof. Let row r correspond to $\langle x_i, y_j, z_k\rangle$ and consider how it can be completed. Recall that the instance is full by Observation 1. Since the parking lot has $6m+1$ columns and the horizontal vehicles in row r cover

$$2i-1+2(m+j-i)+2(m+k-j)-1+2(m-k)+1 = 6m-1$$

cells, it must be that the row is completed with two vertical vehicles. Now consider the vehicles in the completed row from left-to-right. We will see that there are only two possibilities.

First note that the red vehicle of width $2i-1$ must be the first vehicle. Since the left x_i vehicle is in column $2i$, it is possible for it to be next. Likewise, the right x_i vehicle is in column $2i+1$, so it is possible for it to be placed after it. However, it is not possible for the right x_i vehicle to be used without the left x_i vehicle as this would create a gap of one cell. Thus, the first possibility is for the row to be completed with both x_i vehicles, and otherwise neither are used.

Next note that the red and green vehicles have combined width $2i-1+2(m+j-i) = 2(m+j)-1$ cells. Since the y_j vehicles are in column $2(m+j)$ it is possible for it to be next. Likewise, the red and green and blue vehicles have combined width $2i-1+2(m+j-i)+2(m+k-j)-1 = 2(m+j+k)-2$ cells. Since the z_k vehicles are in column $2(m+j)$ it is possible for one of them to join a y_j vehicle in completing the row. This is the second and final possibility as none of the other vertical vehicles can be aligned between the horizontal vehicles. □

Now we can establish the correctness of our reduction.

Theorem 6 *The instance of* RevRush1 *has a solution, if and only if, the instance T of* 3DM *has a perfect matching.*

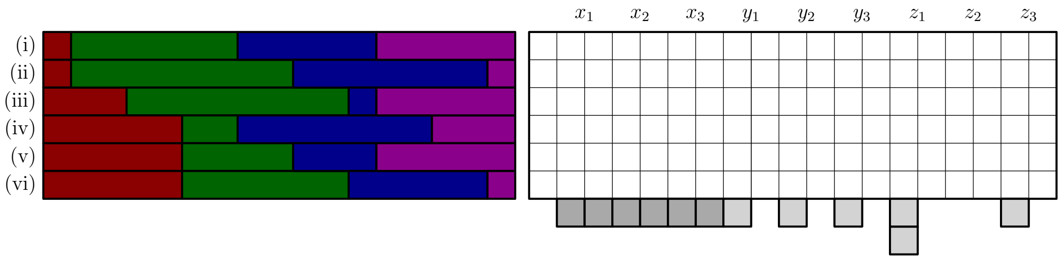
Proof. Assume that T has a perfect matching M. Now we explain how to solve the instance of RevRush1 created by our reduction.

- For each triple in M we complete the corresponding row using xx-satisfaction.
- For each triple outside of M we complete the corresponding row using yz-satisfaction.

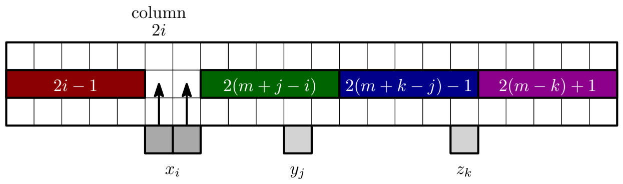
To complete this direction of the proof we must argue that the above choices do not conflict with each other. To see why this is true, suppose that $\langle x_i, y_j, z_k\rangle \in M$. Since we xx-satisfied the corresponding row, it means that the row has no gap above the left column associated with y_j nor the left column associated with z_k. Since M is a perfect matching, these are the only gaps that are not present associated with y_j and z_k. Since the left columns associated with y_j and z_k have exactly $\#(y_j,T)-1$ and $\#(z_k,T)-1$ vehicles, respectively, this means that there are exactly the correct number of gaps available for them.

Assume that RevRush1 has a solution. Now consider the instance of the 3DM problem T that it was created from. We claim that T has a perfect matching M that is constructed as follows.

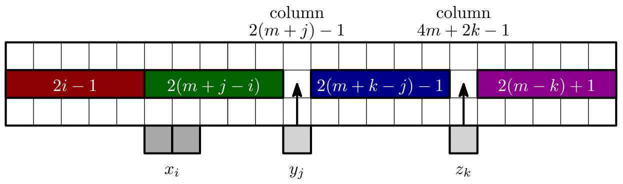
- For each xx-satisfied row, we include the corresponding triple in M.
- For each yz-satisfied row, we exclude the corresponding triple from M.



(a) Reducing the 3DM instance T (with $n = 6$ and $m = 3$) in (1) to a full instance of RevRush1. The vehicle sizes in each row follow the values given at the bottom of Definition 1. For example, row (iii) is for the triple $t_3 = \langle x_2, y_2, z_3\rangle$ so $i = 2$, $j = 3$, and $k = 1$, and this results in vehicles of sizes $2i - 1 = 3$, $2(m + j - i) = 8$, $2(m + k - j) - 1 = 1$, and $2(m - k) + 1 = 5$. Note that the sizes in each row total to $6m - 1 = 17$, which is two less than the number of columns $6m + 1 = 19$ in the parking lot. This means that each row must be extended by exactly two vertical vehicles. The number of size 1 vehicles in the left columns of each y_j and z_k are one less than the following associated frequencies in T: $\#(y_1, T) = 1$; $\#(y_2, T) = 3$; $\#(y_3, T) = 2$; $\#(z_1, T) = 1$; $\#(z_2, T) = 2$; $\#(z_3, T) = 3$. For example, $\#(y_1, T) = 1$, so there are $1 - 1 = 0$ vehicles below the left column associated with y_1.



(b) Vehicles in the x_i columns move up in xx-satisfaction.



(c) Vehicles in the left y_j and z_k columns move up in yz-satisfaction.

Figure 6: The reduction creates rows associated with $\langle x_i, y_j, z_k\rangle$ that can be satisfied in only two different ways.

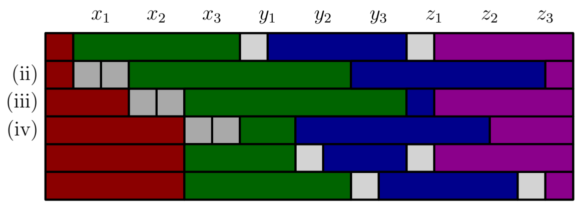


Figure 7: A solution to the instance of RevRush1 in Figure 6a. Note that the rows that are xx-satisfied are (ii), (iii), and (iv). Therefore, this solution corresponds to the 3-dimensional matching M found in (1), namely, $M = \{\langle x_1, y_2, z_3\rangle, \langle x_2, y_3, z_1\rangle, \langle x_3, y_1, z_2\rangle\}$.

Observe that exactly n of the rows must be xx-satisfied (since there are a total of $2n$ vehicles below these columns) so M contains the correct number of triples. Furthermore, it cannot be that any ground element is included more than once, since otherwise there would not be the correct number of gaps available in the RevRush1 solution. □

The reduction in Theorem 6 can be done in polynomial-time, and implies NP-hardness for RevRush and RevRush2 by Corollaries 2 and 4.

Corollary 7 *The decision problems* RevRush, RevRush1, *and* RevRush2 *are NP-complete even when restricted to full bottom-left instances.*

7 Final Remarks

We introduced *Reverse Rush Hour* problems and proved that associated decision problems are NP-complete, even when restricted to full instances where the vehicles start in the bottom and left lanes.

The reduction in Theorem 6 is *parsimonious* up to the final state or parking packing [56, 60]. In other words, the mapping does not change the number of distinct solutions in terms of ParkPack. Since 3DM is ASP-complete, this implies that ParkPack is ASP-complete. Therefore, it is difficult to determine if a *Reverse Rush Hour* puzzle has a unique solution in terms of the parking packing (and not the move sequence).

We plan to explain the above point in more detail in an extended version. We will also provide additional complexity results on *Reverse Rush Hour* puzzles.

- Puzzles with one row ($r = 1$) are polynomial-time solvable, as are those using only vehicles of size one.
- Puzzles using vehicles with at least two different sizes (e.g., cars and trucks) remain NP-complete.
- By allowing fixed blocks and/or disallowing moves into the traffic lanes, it may be possible to establish a PSPACE-complete version of the problem. In particular, it may be possible to adapt some of the constructions used in the citations from Section 2.2.

References

[1] A. Adler, M. Biro, E. D. Demaine, M. Rudoy, and C. Schmidt. Computational complexity of numberless Shakashaka. In *CCCG*, 2015.

[2] A. Adler, J. Bosboom, E. D. Demaine, M. L. Demaine, Q. C. Liu, and J. Lynch. Tatamibari is NP-complete. In *10th International Conference on Fun with Algorithms (FUN 2021)(2020)*, pages 1–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[3] A. Allen and A. Williams. Sto-Stone is NP-Complete. In *CCCG*, pages 28–34, 2018.

[4] G. Aloupis, E. D. Demaine, A. Guo, and G. Viglietta. Classic nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015.

[5] H. Ani, E. D. Demaine, H. Hall, M. Korman, et al. PSPACE-Hard 2D Super Mario games: Thirteen Doors. In *12th International Conference on Fun with Algorithms (FUN 2024)*, pages 21–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[6] J. Ani, J. Bosboom, E. D. Demaine, Y. Diomidov, D. Hendrickson, and J. Lynch. Walking through doors is hard, even without staircases: Proving pspace-hardness via planar assemblies of door gadgets. In *10th International Conference on Fun with Algorithms*, 2020.

[7] J. Ani, L. Chung, E. D. Demaine, Y. Diomidov, D. Hendrickson, and J. Lynch. Pushing Blocks via Checkable Gadgets: PSPACE-Completeness of Push-1F and Block/Box Dude. In *11th International Conference on Fun with Algorithms*, 2022.

[8] J. Ani, E. D. Demaine, D. H. Hendrickson, and J. Lynch. Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. *Theoretical Computer Science*, 969:113945, 2023.

[9] A. Barr, C. Chung, and A. Williams. Block Dude puzzles are NP-hard (and the rugs really tie the reductions together). In *CCCG*, pages 114–125, 2021.

[10] C. Browne. Deductive search for logic puzzles. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8. IEEE, 2013.

[11] C. Browne. Uniqueness in logic puzzles. *Game & Puzzle Design*, 1:35–37, 2015.

[12] J. Brunner. *Subway Shuffle, 1× 1 Rush Hour, and Cooperative Chess Puzzles: Computational Complexity of Puzzles*. PhD thesis, Massachusetts Institute of Technology, 2021.

[13] J. Brunner, L. Chung, E. D. Demaine, D. Hendrickson, A. Hesterberg, A. Suhl, and A. Zeff. 1× 1 Rush Hour with Fixed Blocks Is PSPACE-Complete. In *10th International Conference on Fun with Algorithms*, 2020.

[14] J. Brunner, L. Chung, E. D. Demaine, D. Hendrickson, A. Tockman, et al. ASP-Completeness of Hamiltonicity in Grid Graphs, with Applications to Loop Puzzles. In *12th International Conference on Fun with Algorithms (FUN 2024)*, pages 23–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[15] K. Burke, E. D. Demaine, H. Gregg, R. A. Hearn, A. Hesterberg, M. Hoffman, H. Ito, I. Kostitsyna, J. Leonard, M. Loffler, A. Santiago, C. Schmidt, R. Uehara, Y. Uno, and A. Williams. Single-player and two-player Buttons & Scissors games. *Lecture Notes in Computer Science*, 9943:60–72, 2016.

[16] K. Burke, E. D. Demaine, R. A. Hearn, A. Hesterberg, M. Hoffman, H. Ito, I. Kostitsyna, M. Loffler, C. Schmidt, R. Uehara, Y. Uno, and A. Williams. Single-player and two-player Buttons & Scissors games. In *Japanese Conference on Discrete and Computational Geometry and Graphs*, 2 pages, 2015.

[17] B. Cipra, E. D. Demaine, M. L. Demaine, and T. Rodgers. *Tribute to a Mathemagician*. CRC Press, 2004.

[18] S. Collette, J.-F. Raskin, and F. Servais. On the symbolic computation of the hardest configurations of the Rush Hour game. In *International Conference on Computers and Games*, pages 220–233. Springer, 2006.

[19] G. Cornuéjols, D. Hartvigsen, and W. Pulleyblank. Packing subgraphs in a graph. *Operations Research Letters*, 1(4):139–143, 1982.

[20] J. Culberson. Sokoban is PSPACE-complete. In *Proceedings of the 1st International Conference on Fun with Algorithm*, pages 65–76, 1998.

[21] M. De Biasi and T. Ophelders. Subway Shuffle is PSPACE-complete. *Manuscript, February*, 2015.

[22] E. D. Demaine and M. L. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23(Suppl 1):195–208, 2007.

[23] E. D. Demaine, M. L. Demaine, and J. O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *CCCG*, pages 211—219, 2000.

[24] E. D. Demaine, I. Grosof, and J. Lynch. Push-pull block puzzles are hard. In *International Conference on Algorithms and Complexity*, pages 177–195. Springer, 2017.

[25] E. D. Demaine, I. Grosof, J. Lynch, and M. Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *9th International Conference on Fun with Algorithms (FUN 2018)*, volume 100, pages 18:1–18:21. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

[26] E. D. Demaine, R. A. Hearn, and M. Hoffmann. Push-2-F is PSPACE-complete. In *CCCG*, pages 31–35, 2002.

[27] E. D. Demaine, D. H. Hendrickson, and J. Lynch. Toward a general complexity theory of motion planning: Characterizing which gadgets make games hard. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, pages 62–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[28] E. D. Demaine and M. Hoffmann. Pushing blocks is NP-complete for noncrossing solution paths. In *CCCG*, pages 65—68, 2001.

[29] E. D. Demaine, M. Hoffmann, and M. Holzer. PushPush-k is PSPACE-complete. In *Proceedings of the 3rd International Conference on FUN with Algorithms*, pages 159–170, 2004.

[30] E. D. Demaine, Y. Okamoto, R. Uehara, and Y. Uno. Computational complexity and an integer programming model of Shakashaka. In *CCCG*, 2015.

[31] E. D. Demaine, G. Viglietta, and A. Williams. Super Mario Bros. is Harder/Easier Than We Thought. In *8th International Conference on Fun with Algorithms*, page 13, 2016.

[32] A. Dhagat and J. O'Rourke. Motion planning amidst movable square blocks. In *CCCG*, pages 188–191, 1992.

[33] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215–228, 1999.

[34] G. W. Flake and E. B. Baum. Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1-2):895–911, 2002.

[35] M. Fryers and M. T. Greene. Sokoban, 1995.

[36] J. Gabor and A. Williams. Switches are PSPACE-Complete. In *CCCG*, pages 42–48, 2018.

[37] J. Geller. Baba is You is undecidable. In *2023 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2023.

[38] K. Gordon, J. Lezberg, and A. Williams. Unstacking Slabs Safely in Megalit is NP-Hard. In *CCCG*, pages 169–176, 2022.

[39] A. Greenblatt, O. Hernandez, R. A. Hearn, Y. Hou, H. Ito, M. Kang, A. Williams, and A. Winslow. Turning around and around: Motion planning through thick and thin turnstiles. In *CCCG*, pages 377–387, 2021.

[40] A. Greenblatt, J. Kopinsky, B. North, M. Tyrrell, and A. Williams. Mazezam levels with exponentially long solutions. In *20th Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2017)*, pages 109–110, 2017.

[41] H. Gregg, J. Leonard, A. Santiago, and A. Williams. Buttons & Scissors is NP-Complete. In *CCCG*, 2015.

[42] R. A. Hearn. *Games, puzzles, and computation*. PhD thesis, Massachusetts Institute of Technology, 2006.

[43] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005.

[44] R. A. Hearn and E. D. Demaine. *Games, puzzles, and computation*. CRC Press, 2009.

[45] E. Hordern. *Sliding Piece Puzzles (Recreations in Mathematics)*. Oxford University Press, 1987.

[46] R. M. Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2009.

[47] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European journal of operational research*, 141(2):241–252, 2002.

[48] J. R. Lynch. *A framework for proving the computational intractability of motion planning problems*. PhD thesis, Massachusetts Institute of Technology, 2020.

[49] MIT Hardness Group, H. Ani, E. D. Demaine, H. Hall, R. Ruiz, and N. Venkat. You can't solve these Super Mario Bros. levels: Undecidable Mario games. In *12th International Conference on Fun with Algorithms (FUN 2024)*, pages 22–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[50] D. Packer, S. White, and A. Williams. A Paper on Pencils: A Pencil and Paper Puzzle-Pencils is NP-Complete. In *CCCG*, pages 35–41, 2018.

[51] Patentarcade.com. Case update: Rubin V. Apple Inc..

[52] D. Rubin. *What's the big idea?: and thirty-five other unusual puzzles*. Lippincott, 1979.

[53] D. Rubin. *More Brainstorms: Real Puzzles for the Real Genius*. HarperCollins, 1990.

[54] M. Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1), 1996.

[55] J. Tromp and R. Cilibrasi. Limits of Rush Hour logic complexity. *arXiv preprint cs/0502068*, 2005.

[56] N. Ueda and T. Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. *preprint*, 1996.

[57] R. Uehara. 日の目を見なかった問題たち その2 [Problems that didn't see the light of day Part 2]. `https://www.jaist.ac.jp/~uehara/etc/la/99/index.html`, 1999.

[58] Wikipedia. Rush Hour (puzzle) — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Rush%20Hour%20(puzzle)&oldid=1289108687`, 2025. [Online; accessed 08-May-2025].

[59] Wikipedia. ThinkFun — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=ThinkFun&oldid=1240764920`, 2025. [Online; accessed 08-May-2025].

[60] T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003.

Approximating Metric Depth Queries

Shuhao Tan* David M. Mount*

Abstract

In statistics, the notion of depth reflects how centrally located an object is with respect to a large population of other objects. Statistical depth has numerous applications and has been widely studied in computational geometry and statistics. Some depth measures are discrete (for example, based on a count of points), and others are continuous (for example, based on the sum of distances). This paper considers a natural query problem related to a continuous depth measure. Given a set P of n points in $\mathbb{R}^d$, a *metric depth query* is given a halfspace H, and the answer to the query is the sum of distances to move all the points of $P \cap H$ outside H. This is a natural metric generalization of halfspace range queries.

We present an efficient algorithm for approximating such queries. The data structure is given two positive parameters, ε and δ. The approximation allows for two types of error: first, approximating the function value to within a factor of $1 + \varepsilon$, and second, approximating the query by moving the halfspace by a distance of $\delta \cdot \mathrm{diam}(P)$. We present a data structure for answering such queries which, assuming that d is fixed, has space polynomial in ε, δ, and $\log n$, and whose query time is polylogarithmic in ε, δ, and n. The data structure is based on the observation that the depth function is convex and then applying techniques for approximating convex bodies.

1 Introduction

Statistical depth is a widely studied topic in statistics and computational geometry. Intuitively, depth measures indicate the degree of centrality of an object within a collection of other objects. They provide a natural multidimensional generalization of familiar 1-dimensional measures like the quanitile and rank. Applications include finding the location vector and the scatter matrix for sampled data [17] and symmetry testing of an unknown distribution [14], among others.

Several different definitions of depth have been proposed and studied. One of the most well-known depth functions is the Tukey depth. Given a set of points P in real space and a point x, the *Tukey depth* is defined to be the smallest fraction (or number) of points of P in any closed halfspace that contains x [16]. Many alternative depth functions have been proposed, including the Mahalanobis depth, Zonoid depth, geometrical depth, simplicial depth, and Tverberg depth.

Depth can also be defined for geometric objects other than point sets. For example, given a point set, the *regression depth* of a hyperplane is defined to be the minimum number of points intersected by the hyperplane as it undergoes any continuous motion that takes it from its initial position to vertical [13,18]. A good source for more information is the Mosler survey on depth functions [11].

Of particular interest to us are continuous depth functions that are based on sums of distances (as opposed to discrete functions based on counts and/or quantiles). Examples include the work of Feldman, Fiat, and Sharir on coresets for approximating the sum of distance to a j-flat and applications to facility location [5], and Durocher and Mashghdoust's hyperplane distance statistic [4].

In this paper, we consider a depth-based query problem. Given a halfspace H in $\mathbb{R}^d$ and $p \in \mathbb{R}^d$, define $\mathrm{depth}(p, H)$ to be zero if $p \notin H$, and otherwise it is the perpendicular distance from p to H's defining hyperplane. Given an n-element point set P in $\mathbb{R}^d$ and halfspace H, define the *metric depth* of H with respect to P to be

$$\mathrm{depth}_{\mathrm{M}}(H, P) = \sum_{p \in P} \mathrm{depth}(H, p).$$

Intuitively, this is the total distance that the points of P would need to migrate out of H. This function is a hybrid between the halfspace-based and distance-based depth functions.

This function has a number of nice properties. It vanishes whenever $H \cap P$ is empty. It is continuous and differentiable almost everywhere. Parallel halfspaces are totally ordered by this function. If halfspaces are represented as points in the dual space (that is, as linear functionals), the function is convex.

Our Contributions. In this paper, we present a data structure that achieves sublinear space and logarithmic query time to answer metric depth queries approximately. We propose a novel error model, where we allow for user-controllable errors either in the output of or the

*Department of Computer Science, University of Maryland, College Park, USA, {shuhao,mount}@umd.edu

input to the query. Throughout, we assume that the dimension d is fixed and asymptotic forms hide constant factors that may grow exponentially with d.

Given a point set P and a halfspace H, both in $\mathbb{R}^d$, and $\varepsilon > 0$, we say that a real z is an *ε-approximation* to $\mathrm{depth}_{\mathrm{M}}(H, P)$ if

$$\frac{|z - \mathrm{depth}_{\mathrm{M}}(H,P)|}{\mathrm{depth}_{\mathrm{M}}(H,P)} \leq \varepsilon.$$

Relaxing this, we say that z is a *(ε, δ)-approximation* if z is an ε-approximation to $\mathrm{depth}_{\mathrm{M}}(H', P)$ for some halfspace H' that is parallel to H and the distance between their defining hyperplanes is at most $\delta \cdot \mathrm{diam}(P)$. Here is our main result.

Theorem 1 *Given a point set $P \subset \mathbb{R}^d$ of size n and $0 < \varepsilon, \delta < 1$, there exists a data structure of space $O\big(\varepsilon^{-(d-1)/2}\delta^{-(d+1)/2}\log^{(d+3)/2} n\big)$, such that for any halfspace $H \subset \mathbb{R}^d$, in $O(\mathrm{polylog}(\varepsilon^{-1}, \delta^{-1}, n))$ time it returns either:*

(i) an ε-approximation to $\mathrm{depth}_{\mathrm{M}}(H, P)$, or

(ii) an (ε, δ)-approximation to $\mathrm{depth}_{\mathrm{M}}(H, P)$.

Given that the (ε, δ)-approximation requirement is the weaker of the two, a data structure could satisfy the above theorem by responding only with answers of the type-(ii) variety. We will show that our data structure returns type-(i) answers for the vast majority of queries. Since we can easily return an exact result by storing the centroid and the size of the point set if the hyperplane of the query does not intersect the minimum enclosing ball, we focus on the interesting case of queries that intersect this ball. Consider the unique probability distribution on query hyperplanes that is invariant under Euclidean motions and is restricted to the interesting case of hyperplanes intersecting P's minimum enclosing ball [15]. We show that for a random query from this distribution, a type-(i) result is reported with probability $1 - O(\delta)$.

Related Work. A closely related problem is the range counting problem. Har-Peled and Sharir [6] and Li *et al.* [10] show that a random sample of size $O(\varepsilon^{-2}p^{-1}(\log p^{-1} + \log \varphi^{-1}))$ yields an approximation with probability at least $1 - \varphi$, in the sense that if a halfspace contains more than a p fraction of the total number of points, the estimated count from the sample is an ε-approximation. Observe that our depth function can be computed purely from the range count and the range sum of points of the query halfspace. For each dimension, the range sum could be seen as a range count if we properly shift the points and place multiple copies of points proportional to its coordinate value. The downside of this approach is that the sample achieves a multiplicative error only when the query contains at least a p fraction of the total weight. If a point set is highly clustered, a halfspace could contain a large area in terms of the convex hull but only a small fraction of points in terms of the weight, and this approach would not guarantee multiplicative errors for such queries.

Another closely related problem is computing the sum of distances of all the points to a query hyperplane (in contrast to our problem that considers points on only one side of the hyperplane). It is shown in [5] and [7] how to construct a coreset to answer queries of this type. More specifically, Feldman *et al.* [5] show how to construct a weighted coreset of size $\varepsilon^{-(d+1)} \log^{O(d^2)} n$ such that the weighted sum of distance to a hyperplane with respect to the coreset is an ε-approximation of the sum of distance to the hyperplane in the original point set. The key observation is that, for a given point set, there is an easily obtainable lower bound on the sum of distances. This lower bound serves to limit the resolution of the grids used in the construction of the coreset. This is not applicable in our problem formulation, since the function value can be arbitrarily small when the query is close to the boundary of P's convex hull.

Durocher and Mashghdoust [4] studied a distance-based depth function based on the sum of distances to all $\binom{n}{d}$ hyperplanes formed by the point set. The authors propose a data structure for answering queries exactly using $O\big(n^{d^2}\big)$ space and $O(\log n)$ query time. They exploit the fact that the depth function is linear within each cell of the hyperplane arrangement, and they employ a point location data structure in arrangements by Chazelle and Friedman [3] to retrieve the function. The same approach would work for our depth function as well, since the depth is indeed linear within each cell of the arrangement in the dual space. We elected to introduce approximation and higher query complexity to achieve a sublinear space complexity.

2 Preliminaries

Throughout the paper, we assume that the dimension d is a constant. Given $0 < \varepsilon, \delta < 1$, define $\varepsilon_0 = \varepsilon/\sqrt{16d}$ and $\delta_0 = \delta/\sqrt{9d}$. Given a point set P, let $\mathrm{diam}(P)$ denote its *diameter*, that is, the maximum distance between any two points of the set. Let $\mathrm{conv}(P)$ denote P's convex hull. Let $\mathrm{centroid}(P)$ denote its centroid.

For any function $f : \mathbb{R} \to \mathbb{R}$, let $f(x_0-)$ and $f(x_0+)$ denote the left and right limits of f at x_0, respectively. We make use of several standard concepts from the theory of convex functions (see, e.g., [12]). Given a convex function $f : \mathbb{R}^{d-1} \to \mathbb{R} \cup \{+\infty\}$, define its *effective domain*, $\mathrm{dom}(f)$, to be the set of points $x \in \mathbb{R}^{d-1}$ such that $f(x) \neq +\infty$. Define its *epigraph* to be $\mathrm{epi}(f) = \{(x, z) \in \mathbb{R}^{d-1} \times \mathbb{R} : z \geq f(x)\}$.

Given a set of hyperplanes L in $\mathbb{R}^d$. For any $p \in \mathbb{R}^d$, define

$$Z_{\downarrow}(p) \ = \ \{h \in L \,:\, p \text{ is on or above } h\}.$$

Define $Z_{\uparrow}(p)$ symmetrically and let $\mathrm{level}(p) = |Z_{\downarrow}(p)|$.

We will make use of a data structure presented in [1] for answering approximate membership queries for a convex body based on a hierarchy of ellipsoids. The relevant result is given below.

Lemma 2 (Approximate Convex Membership) *Given a convex body K in $\mathbb{R}^d$ and $r > 0$, there exists a data structure having space $O((\mathrm{diam}(K)/r)^{(d-1)/2})$ such that for any query point $q \in \mathbb{R}^d$, it returns a positive result if $q \in K$, a negative result if $\mathrm{dist}(q, K) > r$, and returns either result otherwise. The query time is $O(\log(\mathrm{diam}(K)/r))$.*

Abdelkader and Mount [2] adapt this data structure to show how to answer vertical ray shooting queries to the body boundary on the construction. We apply the same approach and assume the existence of the following data structure.

Lemma 3 (Approx. Convex Func. Evaluation) *Given a convex function $f : \mathbb{R}^{d-1} \to R$ whose effective domain $D = \mathrm{dom}\, f$ is convex and $r > 0$, let $K = \mathrm{epi}(f)$ be f's epigraph. There exists a data structure of size $O((\mathrm{diam}(D \times R)/r)^{(d-1)/2})$ such that for any query point $x \in D$, the data structure returns a value v such that $v \leq f(x)$ and $\mathrm{dist}((x, v), K) \leq r$. The query time is $O(\log(\mathrm{diam}(K)/r))$.*

3 The Data Structure

3.1 Construction

We will make use of a standard result that relates the diameter of a set and the radius of an enclosing ball. In particular, Jung's Theorem states that for any compact set $K \subset \mathbb{R}^d$, there exists a closed ball of radius at most $\mathrm{diam}(K) \cdot \sqrt{d/2(d+1)}$ [8]. Using this lemma, we can scale the point set P by $\sqrt{2}\,\mathrm{diam}^{-1}(P)$ and place the point set within a unit ball. This transformation uniformly scales all distances by a factor of $\sqrt{2}\,\mathrm{diam}^{-1}(P)$, which we will consider in our analysis. Henceforth, we assume that P is contained within a ball of radius 1, and $\mathrm{diam}(P) \geq \sqrt{2}$.

Next, we translate P so that the center of the enclosing ball is at the origin. As a result, the enclosing ball is contained within $[-1, 1]^d$ (see Figure 1b in Section B). The translation does not affect distances and is only performed to simplify the analysis.

We consider a cover of the $(d-1)$-sphere by hyperspherical caps of polar angle $\pi/4$. For each cap, we build an individual copy of our data structure. For any query halfspace H, we consider the direction of its normal vector and invoke the query algorithm on any such cap that contains this direction (see Figure 1c).

For each cap, we conceptually overlay the cap with P's enclosing ball. We then rotate the entire point set about the center of the ball so that the center of the cap is at $(0, 0, \ldots, 0, 1)$. From now on, we assume that P is already rotated.

We now examine the dual arrangement of P. For a point $p = (p_1, p_2, \ldots, p_d) \in P$, the *dual* of p is a hyperplane, denoted p^*,

$$p^* \ = \ \left\{(x_1, \ldots, x_d) \in \mathbb{R}^d \,:\, x_d + p_d = \sum\nolimits_{i=1}^{d-1} x_i p_i\right\}.$$

Similarly, define the dual of a hyperplane $L = \left\{x \in \mathbb{R}^d \,:\, x_d = a_d + \sum\limits_{i=1}^{d-1} a_i x_i\right\}$, as the point

$$L^* \ = \ (a_1, a_2, \ldots, -a_d).$$

Since we only need to consider queries whose directions are directed into the cap, the angle between the hyperplane bounding the halfspace and the ground plane $\{x \in \mathbb{R}^d \,:\, x_d = 0\}$ is also $\pi/4$. This means that in the dual space, the set of queries we will answer forms an infinite cylinder parallel to the dth coordinate axis and having the base $\{x \in \mathbb{R}^{d-1} \,:\, \|x\| \leq \tan \frac{\pi}{4}\}$, which is a $(d-1)$-ball of radius $\tan \pi/4$ centered at the origin. We naturally extend the base to $[-1, 1]^{d-1}$ and work with a square prism for simplicity. From now on, we use $\Lambda = [-1, 1]^{d-1} \times \mathbb{R}$ to denote this square prism.

Let $\mathrm{dist}_{\mathrm{V}}(p, L)$ denote the vertical distance between a point p and a hyperplane L, that is,

$$\mathrm{dist}_{\mathrm{V}}(p, L) \ = \ \left|\sum_{i=1}^{d-1} a_i p_i + a_d - p_d\right|$$

Since the Euclidean distance between a point and a hyperplane is the projection of the vertical distance on the normal vector of the hyperplane, we can relate the two distances by

$$\mathrm{dist}(p, L) \ = \ \frac{\mathrm{dist}_{\mathrm{V}}(p, L)}{\sqrt{1 + \sum_{i=1}^{d-1} a_i^2}}.$$

Thus, instead of answering queries with respect to Euclidean distances, we will use vertical distances.

Given a query halfspace H bounded by a hyperplane L, since L^* is a point in $\mathbb{R}^d$, the mapping from a point to the sum of vertical distances to the hyperplane naturally defines a function $f : \mathbb{R}^d \to \mathbb{R}$, where

$$f(L^*) = \sum_{p \in H \cap P} \mathrm{dist}_{\mathrm{V}}(p, H) = \sum_{p^* \in Z_{\downarrow}(L^*)} \mathrm{dist}_{\mathrm{V}}(p, L)$$

$$= \sum_{p \in P} \max\left(0, p_d - \sum_{i=1}^{d-1} a_i p_i - a_d\right).$$

We can also write an alternative expression for $f(L^*)$ as

$$\begin{aligned}
&= \sum_{p \in H \cap P} \mathrm{dist}_{\mathrm{V}}(p, H) \\
&= \sum_{p \in H \cap P} \left(p_d - \sum_{i=1}^{d-1} a_i p_i - a_d \right) \\
&= \sum_{i=1}^{d-1} \left(\sum_{p \in H \cap P} (-p_i) \right) a_i + |H \cap P|(-a_d) + \sum_{p \in H \cap P} p_d
\end{aligned} \tag{1}$$

Conceptually, each p^* induces a function that is 0 below the hyperplane and otherwise is the vertical distance to the hyperplane. The function f is simply the sum of these convex functions and is therefore also convex.

Let F_0 be the maximum possible value of f over the upper envelope of P^*. Let $F = F_0 + \delta_0 n / \log n$. We then subdivide the space between the upper and lower envelopes by function values

$$T_i \;=\; \left\{ q \in \Lambda \,:\, 2^{-(i+1)} F < f(q) \le 2^{-i} F \right\}.$$

We cannot store all possible T_i as there are infinitely many. We only process T_i where $2^{-i} F \ge \delta_0$. We apply the convex body approximation in Lemma 2 to the upper envelope of T_i to acquire an approximate membership query oracle $\widehat{T_i}$ with Hausdorff error $\varepsilon_0 \delta_0 / \log n$ (see Figure 3b).

Recall that for a point q in dual space, level(q) is the number of hyperplanes through or below q. So, the level of any point on the lower envelope is 1 and the level of any point on the upper envelope is n. Within each T_i, we further subdivide space based on the level. Let the minimum level in T_i to be l_i, we construct a partition as follows (see Figure 3c)

$$S_{i,j} \;=\; \left\{ q \in T_i \,:\, 2^j l_i \le \mathrm{level}(q) < 2^{j+1} l_i \right\}.$$

For each nonempty $S_{i,j}$, if $2^{i+j} l_i / F \ge \delta_0^{-1} \log n$, we ignore it. Otherwise, let $y_{\min} = \min_{q \in S_{i,j}} q_d$. Let

$$U_{i,j} \;=\; \mathrm{conv}(S_{i,j})$$

See Figure 3d. We then scale f restricted on $U_{i,j}$ as follows:

$$g_{i,j}(q) \;=\; \frac{2^i}{F} \cdot f\left(\frac{F}{2^{i+j+2} l_i \sqrt{d}} q \right),$$

where $g_{i,j}$ is defined from $(2^{i+j+2} l_i \sqrt{d}/F) U_{i,j}$ to $[0,1]$. The range comes from the fact that the maximum value of f when restricted to $U_{i,j}$ is $2^{-i}F$.

We apply the function-value approximation from Lemma 3 to $g_{i,j}$, obtaining an oracle $k_{i,j}$ that provides a lower approximation to the function value with Hausdorff error ε_0. We also apply the membership approximation in Lemma 2 to $U_{i,j}$ to acquire an approximate membership query oracle $U'_{i,j}$ with Hausdorff error $\varepsilon_0 \delta_0 / \log n$.

Our final data structure is the collection of $k_{i,j}$ and $U'_{i,j}$ for all i and j, across the set of covering hyperspherical caps. In addition, we store the centroid centroid(P) of the point set P.

3.2 Query

Answering a query H, involves the following steps:

1. Locate the hyperspherical cap based on the direction of H's normal vector.

2. Conceptually, rotate the point set according to the cap and compute the dual of H as H^* (see Figure 2a).

3. If H^* is below all the $U_{i,j}$, report 0 (see Figure 2c).

4. If H^* is above all the $U_{i,j}$, report

$$n \cdot \mathrm{depth}(H, \mathrm{centroid}(P))$$

(see Figure 2b).

5. Let i_0 be the largest i such that H^* is below $\widehat{T_i}$.

6. If H^* is contained in at least one of $U'_{i_0,j}$, report

$$\frac{\max_{j \,:\, H^* \in U'_{i_0,j}} \frac{F}{2^i} k_{i_0,j} \left(\frac{2^{i_0+j+2} l_i \sqrt{d}}{F} H^* \right)}{\sqrt{1 + \sum_{l=1}^{d-1} h_l^2}}$$

(see Figure 2d).

7. If there exists $U'_{i_0,j}$ above H^*, select the one with the lowest bottom, project H^* vertically up to the boundary of $U'_{i_0,j}$ as H'^*, and report the value as in Step 6.

8. Otherwise, enumerate i from i_0 and locate the first i where at least one of $S_{i,j}$ is not ignored. Project H^* first vertically up to the boundary of $\widehat{T}_{i+1}$, and then repeat Steps 6 and 7 (see Figure 2e).

4 Analysis

We first establish the correctness of our data structure, culminating in Lemma 8.

Lemma 4 *For a query H, if H^* is above all the $U'_{i,j}$, then H^* is above the upper envelope of P^*.*

Proof. Observe that we set $F = F_0 + \delta_0 n / \log n$ where F_0 is the maximum value of f on the upper envelope of the arrangement of P^* within the square prism. So $T_0 =$

$\{q \in \Lambda : (F_0 + \delta_0 n/\log n)/2 < f(q) \leq F_0 + \delta_0 n/\log n\}$. So, there exist points $q \in T_0$ above the upper envelope of P^*, such that $\mathrm{level}(q) = n$. This implies that there exists a nonempty set $S_{0,j}$ such that $2^j l_0 \leq n < 2^{j+1} l_i$. For such $S_{0,j}$, we have:

$$2^{0+j} l_i/F \leq n/F = \frac{n}{F_0 + \delta_0 n/\log n} \leq \delta_0^{-1} \log n$$

So $S_{0,j}$ is not ignored. Moreover, since the upper envelope of $U_{0,j}$ is completely above the upper envelope of P^*, if H^* is above all the $U'_{i,j}$, it must be above the upper envelope of P^*. □

We then show that for any $S_{i,j}$ we ignore during the construction, we can always find a non-ignored $S_{i',j'}$ above within a short distance.

Lemma 5 *Let $S_{i,j}$ be a partition such that $2^{i+j} l_i/F \geq \delta_0^{-1} \log n$, for any point $q \in S_{i,j}$, there exists q' vertically above q such that $q \in S_{i',j'}$ where $2^{i'+j'} l_{i'}/F < \delta_0^{-1} \log n$, and $\mathrm{dist}_V(q, q') \leq 2\delta_0$.*

The idea of the proof is to consider f restricted to the vertical line through q. We observe that the function is piecewise linear and that the ratio $\mathrm{level}(q)/f(q)$ is decreasing within each segment. We can show a worst-case configuration in terms of the distance between q and q', and prove that the worst-case distance is at most $2\delta_0$. Refer to Lemma 14 in Appendix A for a complete proof.

Lemma 6 *For a query H, if H^* is below all the $U'_{i,j}$, then either:*

- *H^* is below the lower envelope of P^*, or*
- *H^* is within a vertical distance of $3\delta_0$ to the lower envelope of P^*.*

Proof. We want to show that if H^* is below all the $U'_{i,j}$ and above the lower envelope of P^*, then H^* is within a vertical distance of $3\delta_0$ to the lower envelope of P^*.

For this to happen, we consider two cases:

1. $0 < f(H^*) \leq \delta_0/2$: The slope of f restricted on the vertical line of H^* is at least 1 between the lower envelope and H^*. So, the distance between H^* and the lower envelope is at most $\delta_0/2$.

2. $\delta_0/2 < f(H^*)$: so $H^* \in T_i$ for some i. Since $H^* \notin U'_{i,j}$ for any j, the $S_{i,j}$ that contains H^* is ignored during the construction process. Hence, we must have $\mathrm{level}(H^*)/f(H^*) \geq \delta_0^{-1} \log n$. Let q be the point vertically below H^* such that $f(q) = \delta_0/2$. q is contained in some T_i by our cutoff value of $\delta_0/2$. Since there is no $U'_{i,j}$ below H^*, we must have $\mathrm{level}(q)/f(q) \geq \delta_0^{-1} \log n$, and this holds between q and H^*. By Lemma 5, $\mathrm{dist}_V(H^*, q) \leq 2\delta_0$. Combining with the previous case, the distance between H^* and the lower envelope is at most $3\delta_0$.

□

Lemma 7 *For a query H, if $H^* \in U'_{i,j}$ for some non-ignored $U_{i,j}$ and H^* is below $\widehat{T}_i$ and above $\widehat{T}_{i+1}$, $(1 - \sqrt{d}\varepsilon_0)\frac{F}{2^i} k_{i,j}(\frac{2^{i+j+2} l_i \sqrt{d}}{F} H^*) \leq f(q) \leq (1 + 4\sqrt{d}\varepsilon_0)\frac{F}{2^i} k_{i,j}(\frac{2^{i+j+2} l_i \sqrt{d}}{F} H^*)$, where q is either H^* or a point vertically below H^* and within distance $\varepsilon_0 \delta_0 \sqrt{d}/\log n$.*

The proof idea is to consider two cases: $q \in U_{i,j}$ or $q \notin U_{i,j}$.

For the former case, we can show bounds on the diameter and the slope of the function $g_{i,j}$, so that we can apply basic geometry to relate the Hausdorff distance and the vertical distance. This allows us to apply Lemma 3 to get an appropriate error bound.

For the latter case, we again show bounds on the slope of the boundary of $U_{i,j}$ and apply Lemma 2 to acquire a bound $O(\varepsilon_0 \delta_0/\log n)$ on the vertical distance. This allows us to bound the deviation of the function value between q and the boundary of $U_{i,j}$, and to show that the deviation is within $O(\varepsilon_0)$ fraction of the local function value. We show that it is an ε-approximation after moving q vertically to the boundary of $U_{i,j}$.

Refer to Lemma 15 in Appendix A for a complete proof.

Lemma 8 *For a query H, the query algorithm described in Section 3.2 reports either:*

(i) an ε-approximation to $\mathrm{depth}_M(H, P)$, or

(ii) an (ε, δ)-approximation to $\mathrm{depth}_M(H, P)$.

Proof. We scale the original point set by $\sqrt{2}\,\mathrm{diam}^{-1}(P)$ in the beginning, making all the distance scaled by the same value. Lemma 5, Lemma 6, and Lemma 7 show that the maximum **vertical distance** in the scaled dual space a query can deviate is at most $\max(3, 2\sqrt{d})\delta_0$. So, the maximum Euclidean distance a query can deviate is no greater than that. So in the original space, the maximum Euclidean distance H can deviate is at most $\mathrm{diam}(P)\max(3, 2\sqrt{d})\delta/\sqrt{9d} \leq \delta\,\mathrm{diam}(P)$.

The scaling does not affect the multiplicative error. So, by Lemma 7, the multiplicative error of the query is at most $4\sqrt{d}\varepsilon_0 = \varepsilon$. □

Next, we analyze the space complexity of the data structure, culminating in Lemma 12.

Lemma 9 *For non-ignored $S_{i,j}$, $(2^{i+j+2}l_i\sqrt{d}/F)U_{i,j}$ is bounded in a hyperrectangle of side length $O(\delta_0^{-1}\log n)^d$.*

Proof. Since $S_{i,j}$ is not ignored, $2^{i+j}l_i/F < \delta_0^{-1}\log n$. Also because $U_{i,j} \subset \Lambda$, the first $d-1$ dimensions of $(2^{i+j+2}l_i\sqrt{d}/F)U_{i,j}$ is bounded by $[-4\sqrt{d}\delta_0^{-1}\log n, 4\sqrt{d}\delta_0^{-1}\log n]$. So, the length is $O(\delta_0^{-1}\log n)$.

Observe that since $P \subset [-1,1]^d$, the hyperplanes formed by the points have a limited range of slope and intercept. More specifically, since $-1 \le p_d \le 1$, for any $p \in P$, the dual p^* intersects the dth coordinate axis within $[-1,1]$. Moreover, $-1 \le p_i \le 1$ for any i, so the minimum possible dth dimension coordinate p^* can reach within the square prism is $-1-(d-1) = -d$, and the maximum possible coordinate is $1+(d-1) = d$. This means that the maximum possible vertical distance between the upper and lower envelope of the dual arrangement is $O(1)$ assuming a constant dimension d. So, the d-th dimensional length of $(2^{i+j+2}l_i\sqrt{d}/F)U_{i,j}$ is bounded by $O(\delta_0^{-1}\log n)$. ☐

Lemma 10 *The total number of non-ignored $S_{i,j}$ is $O((\log\delta_0^{-1}+\log n)\log n)$.*

Proof. The maximum number of T_i is $O(\log\frac{F}{\delta_0})$. In the proof of Lemma 9, we essentially show that $F_0 = O(dn) = O(n)$. So $F = O(n)$. And the number of T_i is $O(\log\delta_0^{-1}+\log n)$.

For each T_i, level(q) can go from l_i to n where $l_i > 0$. So, the maximum number of $S_{i,j}$ for each i is $O(\log n)$.

Combining these, the total number of non-ignored $S_{i,j}$ is $O((\log\delta_0^{-1}+\log n)\log n)$. ☐

Lemma 11 *There exists a set of hyperspherical caps with polar angle $\pi/4$, such that:*

- *The set covers the unit $(d-1)$-sphere, and*
- *The size of the set is $O(1)$.*

The proof idea is to consider a maximal packing of hyperspherical caps with polar angle $\pi/8$, we show that expanding each cap to polar angle $\pi/4$ creates a covering and show that the size is $O(1)$ by bounding the area of each cap. Refer to Lemma 16 in Appendix A for a complete proof.

Lemma 12 *The space complexity of the data structure described in Section 3 is $O(\varepsilon^{-(d-1)/2}\delta^{-(d+1)/2}\log^{(d+3)/2}(n))$.*

Proof. From Lemma 9, for a $S_{i,j}$ that is not ignored, $\mathrm{diam}((2^{i+j+2}l_i\sqrt{d}/F)U_{i,j}) = \sqrt{d}O(\delta_0^{-1}\log n) = O(\delta_0^{-1}\log n)$. By Lemma 3, the space complexity for each $k_{i,j}$ is $O((\varepsilon_0^{-1}\delta_0^{-1}\log n)^{(d-1)/2})$.

Both T_i and $U_{i,j}$ are contained in $[-1,1]^{d-1}\times[-d,d]$, so $\mathrm{diam}(T_i), \mathrm{diam}(U_{i,j}) \le 2\sqrt{d} = O(1)$. By Lemma 2, the space complexity for each $\widehat{T_i}$ and each $U'_{i,j}$ is $O((\varepsilon_0^{-1}\delta_0^{-1}\log n)^{(d-1)/2})$.

We know $\varepsilon_0 = O(\varepsilon)$ and $\delta_0 = O(\delta)$. By Lemma 10, there are at most $O((\log\delta_0^{-1}+\log n)\log n)$ non-ignored $S_{i,j}$, and we conclude that the space complexity for one copy of data structure with respect to a hyperspherical cap is $O(\varepsilon^{-(d-1)/2}\delta^{-(d+1)/2}\log^{(d+3)/2}(n))$.

By Lemma 11, we use only a constant number of hyperspherical caps, so the space complexity above is for the entire data structure. ☐

Finally, we analyze the query time.

Lemma 13 *The query time complexity of the data structure described in Section 3 is $O(\mathrm{polylog}(\varepsilon^{-1},\delta_0^{-1},n))$.*

Proof. Lemma 10 shows that we query oracles at most $O((\log\delta_0^{-1}+\log n)\log n)$ times. The proof of Lemma 12 shows that $\mathrm{diam}(K)/r$ in Lemma 2 and $\mathrm{diam}(D\times R)/r$ in Lemma 3 are both $O(\varepsilon^{-1}\delta^{-1}\log n)$. So, the query time for each oracle is $\log(\varepsilon^{-1}\delta^{-1}\log n)$. Enumerating spherical caps can be done in constant time. Hence, the query time overall is $O(\mathrm{polylog}(\varepsilon^{-1},\delta_0^{-1},n))$. ☐

Combining Lemmas 8, 12, and 13 establishes our main theorem, Theorem 1.

5 Discussion

Using our halfspace-based depth function, it is natural to define a depth function for a hyperplane to be the minimum depth of the two halfspaces bounded by it. Furthermore, a depth function for a point can be defined as the minimum depth among all the halfspaces passing through it. A possible topic for future research is to study these depth functions and design efficient data structures for them.

Our error model allows translating the query halfspace to avoid regions where approximation is difficult to approximate. More specifically, these are regions where a slight perturbation of the halfspace introduces very large change in the included points relative to the function value. A future direction of study is the approximability of such cases in order to either improve the error model or to derive a lower bound on any approximation data structure for this problem.

References

[1] A. Abdelkader and D. M. Mount. Economical Delone sets for approximating convex bodies. In *Proc. 16th Scand. Workshop Algorithm Theory*, pages 4:1–4:12, 2018. doi:10.4230/LIPIcs.SWAT.2018.4.

[2] A. Abdelkader and D. M. Mount. Smooth distance approximation. In *Proc. 31st Annu. European Sympos. Algorithms*, pages 5:1–5:18, 2023. doi:10.4230/LIPIcs.ESA.2023.5.

[3] B. Chazelle and J. Friedman. Point location among hyperplanes and unidirectional ray-shooting. *Comput. Geom.*, 4:53–62, 1994. doi:10.1016/0925-7721(94)90009-4.

[4] S. Durocher and A. Mashghdoust. Hyperplane distance depth. In *Proc. 36th Canad. Conf. Comput. Geom.*, pages 207–213, 2024. URL: https://cosc.brocku.ca/~rnishat/CCCG_2024_proceedings.pdf.

[5] D. Feldman, A. Fiat, and M. Sharir. Coresets for weighted facilities and their applications. In *Proc. 47th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 315–324, 2006. doi:10.1109/FOCS.2006.22.

[6] S. Har-Peled and M. Sharir. Relative (p, ε)-approximations in geometry. *Discrete Comput. Geom.*, 45:462–496, 2011. doi:10.1007/s00454-010-9248-1.

[7] Z. Huang and J. Xu. An efficient sum query algorithm for distance-based locally dominating functions. *Algorithmica*, 82:2415–2431, 2020. doi:10.1007/s00453-020-00691-w.

[8] H. Jung. Ueber die kleinste Kugel, die eine räumliche Figur einschliesst. *J. Reine Angew. Math*, 123:241–257, 1901. URL: http://eudml.org/doc/149122.

[9] S. Li. Concise formulas for the area and volume of a hyperspherical cap. *Asian J. Math. Stat.*, 4:66–70, 2011. doi:10.3923/ajms.2011.66.70.

[10] Y. Li, P. M. Long, and A. Srinivasan. Improved bounds on the sample complexity of learning. *J. Comput. Sys. Sci.*, 62:516–527, 2001. doi:10.1006/jcss.2000.1741.

[11] K. Mosler. Depth statistics. In C. Becker, R. Fried, and S. Kuhnt, editors, *Robustness and Complex Data Structures*, pages 17–34. Springer, 2013. doi:10.1007/978-3-642-35494-6_2.

[12] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, NJ, 1997. doi:10.1515/9781400873173.

[13] P. J. Rousseeuw and M. Hubert. Regression depth. *J. Amer. Statist. Assoc.*, 94:388–402, 1999. doi:10.2307/2670155.

[14] P. J. Rousseeuw and A. Struyf. A depth test for symmetry. In C. Huber-Carol, N. Balakrishnan, M. S. Nikulin, and M. Mesbah, editors, *Goodness-of-Fit Tests and Model Validity*, pages 401–412. Birkhäuser, 2002. doi:10.1007/978-1-4612-0103-8_30.

[15] L. A. Santaló. *Integral Geometry and Geometric Probability*. Cambridge Mathematical Library. Cambridge University Press, 2nd edition, 2004.

[16] J. W. Tukey. Mathematics and the picturing of data. In *Proc. Internat. Congr. Math.*, pages 523–531, 1975. URL: https://api.semanticscholar.org/CorpusID:233506359.

[17] M. Zhang, Y. Song, and W. Dai. Fast robust location and scatter estimation: A depth-based method. *Technometrics*, 66:14–27, 2024. doi:10.1080/00401706.2023.2216246.

[18] Y. Zuo. On general notions of depth for regression. *Stat. Science*, 36:142–157, 2021. doi:10.1214/20-STS767.

Appendix

A Deferred Proofs

We present complete proofs that did not fit within the page limit.

Lemma 14 (Restatement of Lemma 5) *Let $S_{i,j}$ be a partition such that $2^{i+j} l_i / F \geq \delta_0^{-1} \log n$, for any point $q \in S_{i,j}$, there exists q' vertically above q such that $q \in S_{i',j'}$, where $2^{i'+j'} l_{i'} / F < \delta_0^{-1} \log n$ and $\mathrm{dist}_{\mathrm{V}}(q, q') \leq 2\delta_0$.*

Proof. Let $q \in S_{i,j}$ where $2^{i+j} l_i / F \geq \delta_0^{-1} \log n$. Since $f(q) \leq 2^{-i} F$ and $\mathrm{level}(q) \geq 2^j l_i$, we have $\mathrm{level}(q)/f(q) \geq 2^{i+j} l_i / F \geq \delta_0^{-1} \log n$. We will show that there exists q' vertically above q, where $\mathrm{level}(q')/f(q') < \delta_0^{-1} \log n$ and $\mathrm{dist}(q, q') \leq 2\delta_0$.

Consider function $h(x) = f((q_1, q_2, \ldots, q_{d-1}, x))$, the function of f restricted on the vertical line through q. We observe the following properties of h:

- h is a piecewise linear function from $\mathbb{R}$ to $\mathbb{R}$.
- Assuming general position, there are $n+1$ segments in h, more specifically, each vertex corresponds to a $p^* \in P^*$.
- Assuming general position, for each vertex $(x, h(x))$, $h'(x+) = h'(x-) + 1$, in other words, for any two adjacent segments, the slope of the segment to the right is exactly one more tan the that of the segment to the left. This can be observed from Equation 1.

Let $x_0 = q_d$, we now show that there exists $x_0 < x' \leq x_0 + 2\delta_0$, such that $h'(x')/h(x') < \delta_0^{-1}\log n$. Observe that within a segment, $h'(x)/h(x)$ is decreasing since $h(x)$ is increasing and $h'(x)$ is unchanged. We will describe a configuration of segments that maximizes the distance between x_0 and x'.

Consider a function $m : \mathbb{R} \to \mathbb{R}$ constructed in the following way:

- $m(x) = h(x)$ for $x \leq x_0$.
- Repeatedly: following the current segment, whenever $m'(x)/m(x) = \delta_0^{-1}\log n$, add a vertex and start the next segment.

Let m_i be the x-coordinate of the ith vertex we add. And let $m_0 = x_0$. We first show that m_i are well-defined. Observe that:

$$\begin{aligned}
&m\left(m_0 + \frac{\delta_0}{\log n} - \frac{m(m_0)}{m'(m_0+)}\right)\\
&= m(m_0) + m'(m_0+)\left(\frac{\delta_0}{\log n} - \frac{m(m_0)}{m'(m_0+)}\right)\\
&= \frac{\delta_0}{\log n}m'(m_0+).
\end{aligned}$$

This shows that the construction of m would set $m_1 = m_0 + \delta_0^{-1}\log n - m(m_0)/m'(m_0)$. Now assume m_i exists for $1 \leq i < n - m'(m_0+)$. We know $m'(m_i-)/m(m_i) = \delta_0^{-1}\log n$ and $m'(m_i+) = m'(m_i-) + 1$. Observe that:

$$\begin{aligned}
&m\left(m_i + \frac{\delta_0}{m'(m_i+)\log n}\right)\\
&= m(m_i) + m'(m_i+)\left(\frac{\delta_0}{m'(m_i+)\log n}\right)\\
&= \frac{\delta_0 m'(m_i-)}{\log n} + \frac{\delta_0}{\log n} = \frac{\delta_0}{\log n}m'(m_i+).
\end{aligned}$$

This shows that the construction will pick $m_{i+1} = m_i + \frac{\delta_0}{m'(m_i+)\log n}$.

Now we show the span where $m'(x)/m(x) \geq \delta_0^{-1}\log n$ is no shorter than that of h. If $h = m$, we are done. Otherwise, consider the first difference in the arrangement of segments between m and h. Let the x-coordinate be t. Since m adds new segment at the latest possible point, if m adds a segment at t, $h'(x)/h(x) < \delta_0^{-1}\log n$ right after t, and m has a longer span. If h adds one or more segments at t, consider a modified function h_t where we delay the addition of segment until either h adds one or more segments again, or m adds a segment. Let the x-coordinate the segments moved to be t'. We observe that:

- $h_t(x) = m(x)$ for $x \leq t'$, and $h_t'(x)/h_t(x) = m'(x)/m(x) \geq \delta_0^{-1}\log n$ for $x_0 \leq x < t$.
- $h_t'(t+) < h'(t+)$ and $h_t(t) = h(t)$, so $h_t(t') < h(t')$. We also observe that $h_t'(t'+) = h'(t'+)$. So $h_t'(x)/h_t(x) > h'(x)/h(x)$ for $x > t'$.

These two properties show that the span starting at x_0 where $h'(x)/h(x) \geq \delta_0^{-1}\log n$ is a subset of that of h_t. By repeating the modification, we can change h to m without shortening the span. Hence m has the longest possible span.

Since the total number of segments is $n+1$, we can add at most $n - m'(m_0)$ segments after m_0, so we have:

$$\begin{aligned}
x' - x &\leq m_{n-m'(m_0)+1} - m_0\\
&= m_1 - m_0 + \sum_{i=1}^{n-m'(m_0)}(m_{i+1} - m_i)\\
&= \frac{\delta_0}{\log n} - \frac{m(m_0)}{m'(m_0+)} + \sum_{i=m'(m_0)}^{n}\frac{\delta_0}{i\log n}\\
&\leq \frac{\delta_0}{\log n} - \frac{m(m_0)}{m'(m_0+)} + \frac{\delta_0}{\log n}(\log n + 1)\\
&\leq 2\delta_0.
\end{aligned}$$

So let $q' = (q_1, q_2, \ldots, q_{d-1}, x')$, and we have $\text{level}(q')/f(q') < \delta_0^{-1}\log n$. In the proof of Lemma 4, we have established that there always exists a $S_{0,j}$ at the top for some j. And the top point q_t in that set has the property that $\text{level}(q_t)/f(q_t) < \delta_0^{-1}\log n$. This means q' is in or below T_0. Let $q' \in S_{i,j}$, we have $2^{i+j}l_i/F \leq \text{level}(q')/f(q') < \delta_0^{-1}\log n$ because $\text{level}(q') \leq 2^j l_i$ and $f(q') > 2^{-i}F$. □

We remark that with a slight modification to the proof above, it is easy to show that for any vertical line in the dual space, consider the collection of intervals $[p_1, q_1], [p_2, q_2], \ldots, [p_k, q_k]$ where $h'(x)/h(x) \geq \delta_0^{-1}\log n$ in each interval, we have $q_i - p_i \leq (\delta_0/\log n)\sum_{i=h'(p_i)}^{h'(q_i)} 1/i$. And thus $\sum_{i=1}^{k}(q_i - p_i) \leq 2\delta_0$. This implies that over all the queries that intersect the minimum enclosing ball of P, only $O(\delta)$ fraction of the queries will be answered with another query.

Lemma 15 (Restatement of Lemma 7) *For a query H, if $H^* \in U'_{i,j}$ for some non-ignored $U_{i,j}$ and H^* is below $\widehat{T}_i$ and above $\widehat{T}_{i+1}$, $(1 - \sqrt{d}\varepsilon_0)\frac{F}{2^i}k_{i,j}(\frac{2^{i+j+2}l_i\sqrt{d}}{F}H^*) \leq f(q) \leq (1 + 4\sqrt{d}\varepsilon_0)\frac{F}{2^i}k_{i,j}(\frac{2^{i+j+2}l_i\sqrt{d}}{F}H^*)$, where q is either H^* or a point vertically below H^* and within distance $\varepsilon_0\delta_0\sqrt{d}/\log n$.*

Proof. Let $X(q) = \frac{F}{2^{i+j+2}l_i\sqrt{d}}q$. We consider two cases:

$H^* \in U_{i,j}$: $k_{i,j}$ is a lower approximation of $g_{i,j}$, so we have:

$$\frac{F}{2^i}k_{i,j}(X^{-1}(H^*)) \leq \frac{F}{2^i}g_{i,j}(X^{-1}(H^*)) = f(H^*)$$

From Equation 1, we can calculate the gradient of $g_{i,j}$:

$$\nabla g_{i,j}(q) = \nabla \frac{2^i}{F} f(X(q))$$
$$= -\frac{1}{2^{j+2}l_i}\left(\sum_{p\in Z_\downarrow(X(q))} p_1, \ldots, \sum_{p\in Z_\downarrow(X(q))} p_{d-1}, -\operatorname{level}(X(q))\right)$$

Since $-1 \le p_i \le 1$, we have $\left|\sum_{p\in Z_\downarrow(X(q))} p_i\right| \le \operatorname{level}(X(q))$. Moreover, for $p \in U_{i,j}$, we have $\operatorname{level}(p) < 2^{j+1}l_i$. So for $q \in (2^{i+j+2}l_i\sqrt{d}/F)U_{i,j}$, we have $\nabla g_{i,j}(q) \in [-1,1]^{d-1}\times[0,1]$. So $|\nabla g_{i,j}(q)| \le \sqrt{d}$. Since the Hausdorff distance between $k_{i,j}$ and $g_{i,j}$ is at most ε_0, we have $g_{i,j}(x) - k_{i,j}(x) \le \varepsilon_0\sqrt{|\nabla g_{i,j}|^2+1} \le \varepsilon_0\sqrt{d+1}$.

Since H^* is above $\widehat{T}_{i+1}$, $f(H^*) \ge 2^{-(i+1)}F$, so $g_{i,j}(X(H^*)) \ge 1/2$, so we have $g_{i,j}(X(H^*)) \le k_{i,j}(X(H^*)) + 2\varepsilon_0\sqrt{d+1}g_{i,j}(X(H^*))$. And it follows that $f(H^*) \le (1+4\sqrt{d}\varepsilon_0)\frac{F}{2^i}k_{i,j}(\frac{2^{i+j+2}l_i\sqrt{d}}{F}H^*)$.

***$H^* \notin U_{i,j}$*:** H^* is within a Hausdorff distance of $\varepsilon_0\delta_0/\log n$ to the upper envelope of $U_{i,j}$. Note that the upper envelope of $U_{i,j}$ consists either hyperplane from $\operatorname{conv}(\{q\in\mathbb{R}^d : \operatorname{level}(q) < 2^{j+1}l_i\})$ or from $\{q \in \mathbb{R}^d : f(q) < 2^{-i}F\}$.

Since the upper envelope of $\{q\in\mathbb{R}^d : \operatorname{level}(q) < 2^{j+1}l_i\}$ is the $(2^{j+1}l_i)$th level of P^*, it consists of hyperplanes from P^*. By Mean Value Theorem, the surface normals of $\operatorname{conv}(\{q\in\mathbb{R}^d : \operatorname{level}(q) < 2^{j+1}l_i\})$ are bounded by the normals in P^*. So the surface normal is contained in $[-1,1]^{d-1}\times\{1\}$. If a point has a Hausdorff distance of z to the set, the vertical distance is at most $z\sqrt{d}$.

The upper envelope of $\{q\in\mathbb{R}^d : f(q) < 2^{-i}F\}$ is the function value level set of f. Take any q on the level set. The normal vector of the tangent hyperplane of the level set through q is $\nabla f(q)$, which is contained in $[-\operatorname{level}(q), \operatorname{level}(q)]^{d-1}\times\{\operatorname{level}(q)\}$ as we have shown before. If a point has a Hausdorff distance of z to the set, the vertical distance is at most $z\sqrt{d}$.

Combining this we conclude the vertical distance between H^* and $U_{i,j}$ is at most $\varepsilon_0\delta_0\sqrt{d}/\log n$. Now consider $h(x) = f((H_1^*, H_2^*, \ldots, H_{d-1}^*, x))$, similar to the h we define in the proof of Lemma 5. Let $q = (H_1^*, H_2^*, \ldots, H_{d-1}^*, x_0)$ be the point on the upper envelope of $U_{i,j}$ vertically below H^*. Since $S_{i,j}$ is not ignored, we have $h'(x_0)/h(x_0) \le 2^{i+j+2}l_i/F < 4\delta_0^{-1}\log n$. Hence we have

$$\frac{F}{2^i}\left(k_{i,j}(X^{-1}(H^*)) - k_{i,j}(X^{-1}(q))\right) \le h'(x_0)(H_d^* - x_0)$$
$$\le (h(x_0)\delta_0^{-1}\log n)(\varepsilon_0\delta_0\sqrt{d}/\log n)$$
$$= \sqrt{d}\varepsilon_0 h(x_0)$$
$$\le \sqrt{d}\varepsilon_0\frac{F}{2^i}k_{i,j}(X^{-1}(H^*)).$$

Hence

$$(1-\sqrt{d}\varepsilon_0)\frac{F}{2^i}k_{i,j}(X^{-1}(H^*))$$
$$\le f(q) \le (1+4\sqrt{d}\varepsilon_0)\frac{F}{2^i}k_{i,j}(X^{-1}(q))$$
$$\le (1+4\sqrt{d}\varepsilon_0)\frac{F}{2^i}k_{i,j}(X^{-1}(H^*)).$$

□

Lemma 16 (Restateemnt of Lemma 11) *There exists a set of hyperspherical caps with polar angle $\pi/4$, such that:*

- *The set covers the unit $(d-1)$-sphere, and*
- *The size of the set is $O(1)$.*

Proof. Consider a maximal packing C of hyperspherical caps with polar angle $\pi/4$ on the unit $(d-1)$-sphere. So $\forall c_1, c_2 \in C, c_1 \ne c_2 \implies c_1 \cap c_2 = \emptyset$ (disjoint). And for any point q outside the union of caps in C, there is no way to place another cap at q (maximal). The latter condition can also be phrased as: for any point q outside the union of caps in C, there exists a cap $c \in C$, such that there exists a point $q' \in c$ where the polar angle between q and q' is smaller than $\pi/4$.

Now, if we replace each cap in C by a cap with polar angle $\pi/2$ instead, it forms a cover due to the maximality of C.

We claim $|C| = O(1)$. Li [9] shows the surface area of a hyperspherical cap on a unit $(d-1)$-sphere with polar angle ϕ is:

$$A_{d-1}^{\text{cap}}(\phi) = \frac{2\pi^{(d-1)/2}}{\Gamma((d-1)/2)}\int_0^\phi \sin^{d-2}(\theta)\,\mathrm{d}\theta \tag{2}$$

We can derive a crude lower-bound for $\phi = \pi/4$:

$$A_{d-1}^{\text{cap}}(\pi/4) \ge \frac{2\pi^{(d-1)/2}}{\Gamma((d-1)/2)}\int_{\pi/8}^{\pi/4} \sin^{d-2}(\theta)\,\mathrm{d}\theta$$
$$\ge \frac{2\pi^{(d-1)/2}}{\Gamma((d-1)/2)}\int_{\pi/8}^{\pi/4} \sin^{d-2}(\pi/8)\,\mathrm{d}\theta$$
$$= \frac{\pi^{(d-1)/2}\sin^{d-2}(\pi/8)}{4\Gamma((d-1)/2)}$$

On the other hand, the surface area of a unit $(d-1)$-sphere is:

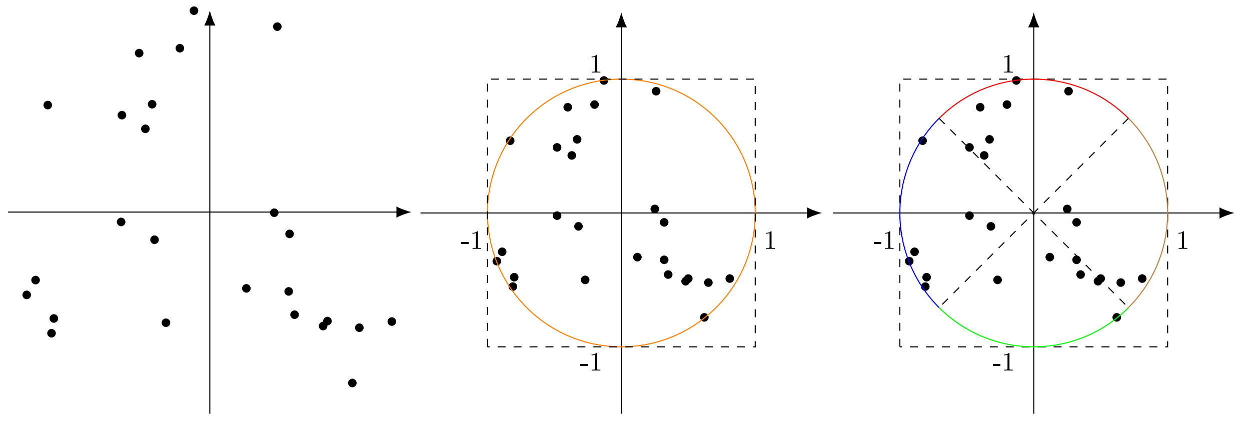
$$A_{d-1} = \frac{2\pi^{d/2}}{\Gamma(d/2)} \tag{3}$$

Since caps in C are disjoint, we have:

$$|C| \le \frac{A_{d-1}}{A_{d-1}^{\text{cap}}(\pi/4)} \le \frac{8\sqrt{\pi}\Gamma((d-1)/2)}{\sin^{d-2}(\pi/8)\Gamma(d/2)}$$
$$\le \frac{8\sqrt{\pi}}{\sin^{d-2}(\pi/8)} = O(1)$$

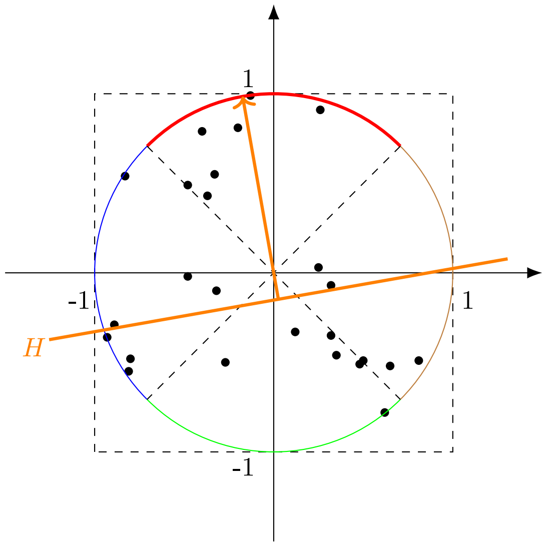
□

B Deferred Figures



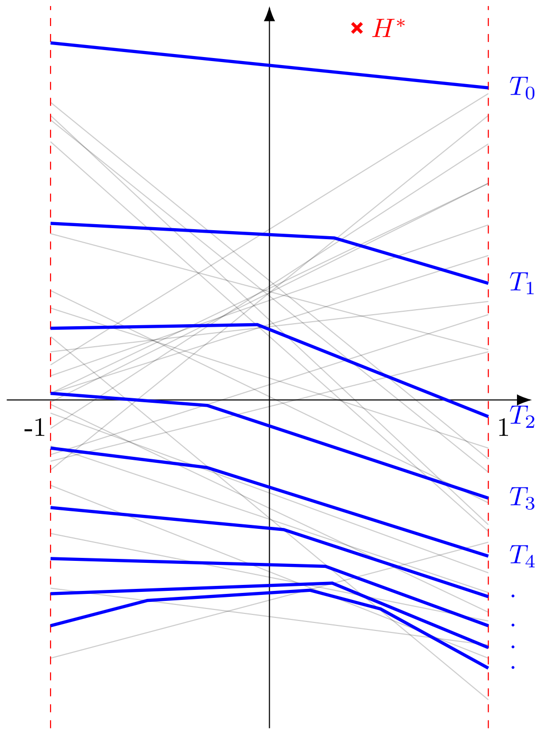
(a) Original point set (b) Scaled and translated to $[-1, 1]^d$ (c) Spherical cap covering

Figure 1: Preprocessing.

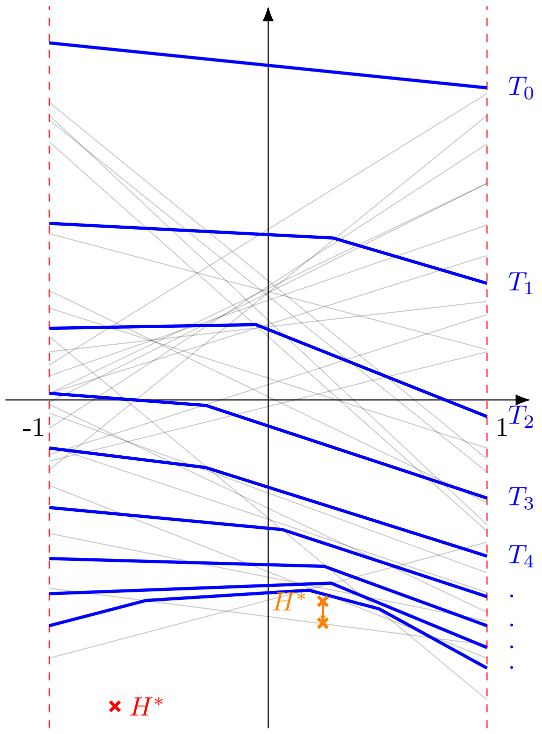


(a) locate covering cap

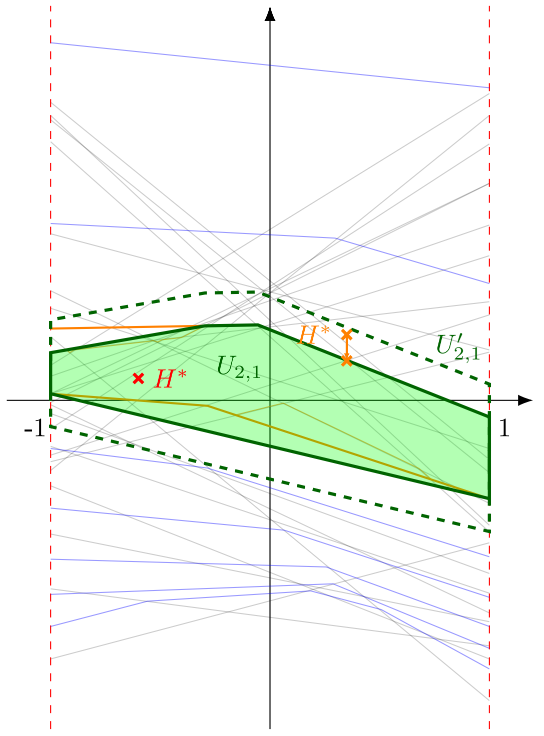
Figure 2: Query processing.



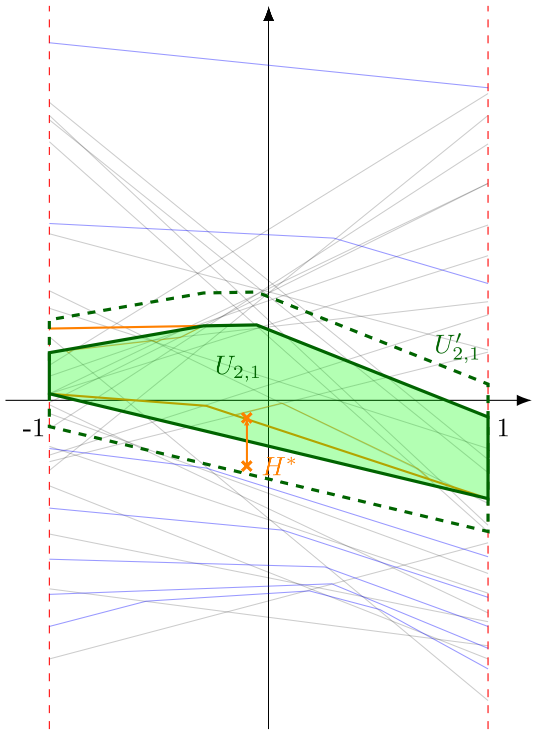
(b) query is above all T_i



(c) query is below all T_i. orange: the query that gets moved

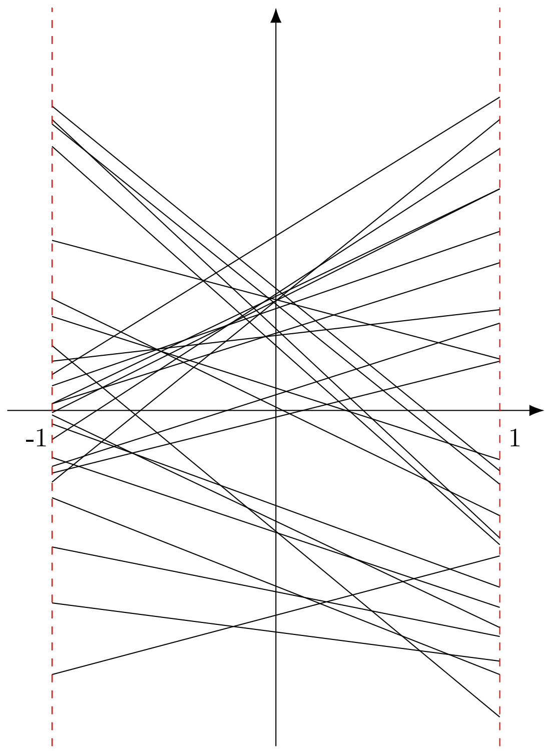


(d) query is in T_i and inside $U'_{i,j}$. orange: the query that gets moved

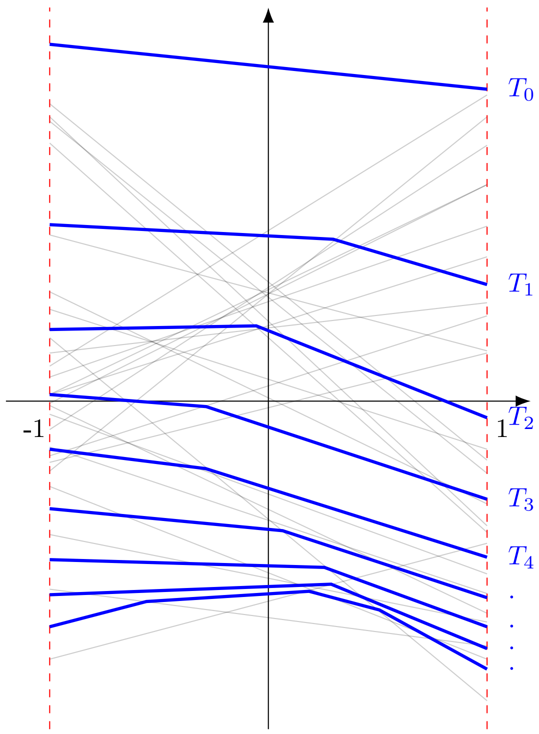


(e) query is in T_i but not inside any $U'_{i,j}$, moved to the first T_k with non-ignored $U_{k,l}$

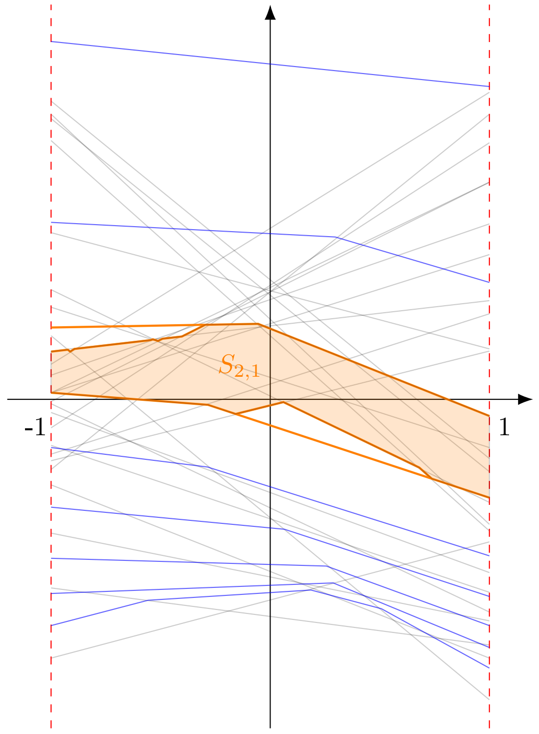
Figure 2: Query processing. (cont.)



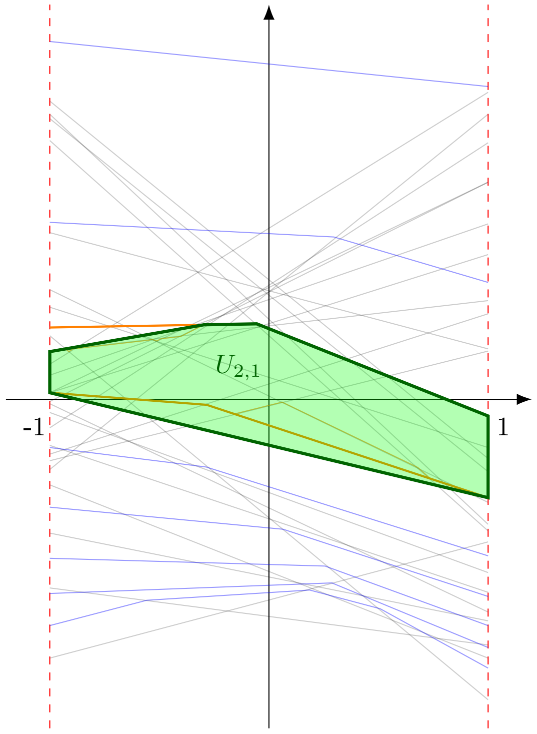
(a) Dual space



(b) Function value level sets T_i



(c) Arrangement level sets $S_{i,j}$



(d) $U_{i,j}$ as the domain of $k_{i,j}$

Figure 3: Construction.

Reconstructing Bounded Treelength Graphs with Linearithmic Shortest Path Distance Queries

Chirag Kaudan* Amir Nayyeri†‡

Abstract

We consider the following graph reconstruction problem: given an unweighted connected graph $G = (V, E)$ with visible vertex set V and an oracle which takes two vertices $u, v \in V$ and returns the shortest path distance between u and v, how many queries are needed to reconstruct E?

Specifically, we consider bounded degree Δ and bounded treelength tl connected graphs and show that reconstruction can be done in $O_{\Delta,\mathrm{tl}}(n \log n)$ queries with a deterministic algorithm. This result improves over the best known algorithm (deterministic or randomized) for this graph class by a $\log n$ factor and matches the known lower bound for the class of graphs with bounded chordality, which is a subclass of bounded treelength graphs.

1 Introduction

In the graph reconstruction problem, only the vertex set V of a hidden graph $G = (V, E)$ is visible, and the goal is to reconstruct the edge set E by querying an oracle. A graph reconstruction algorithm aims to discover the edge set with as few adaptively chosen queries as possible. The reconstruction task is considered complete once there is a unique graph consistent with all answers to the queries. Several query models have been studied for graph reconstruction, including edge existence queries, edge counting, effective resistance queries, and shortest path queries [1–3, 14, 19].

In the Shortest Path (SP) distance query model, we have access to a distance oracle: for any two vertices $u, v \in V$, a query returns the length of the shortest path between u and v in G and returns ∞ if no such path exists. The study of network reconstruction using shortest path queries is motivated by applications such as Internet topology reconstruction and evolutionary tree reconstruction [4, 5, 10, 13, 20].

Any (unweighted) graph can trivially be reconstructed using $\binom{n}{2}$ SP queries, simply by querying every pair of vertices to check whether they are adjacent. Unfortunately, this upper bound is tight even for trees, once we allow unbounded degree [19]. Moreover, the bound is tight if one allows disconnected graphs–reconstructing a graph with n vertices and 1 edge requires $\Omega(n^2)$ queries. In light of these observations, all non-trivial algorithmic results on graph reconstruction focus on connected bounded degree graphs.

Mathieu et al. [16, 17] showed that general bounded-degree graphs can be reconstructed using $\tilde{O}_\Delta(n^{3/2})$ expected queries via a randomized algorithm, where $\tilde{O}(f(n))$ is shorthand for $O(f(n) \cdot \mathrm{polylog}(n))$. This is still the best known general upper bound today. It is unknown whether there exists a deterministic algorithm to reconstruct a bounded-degree graph in $\tilde{O}_\Delta(n^{3/2})$ queries. Mathieu et al. further described a randomized algorithm that reconstructs chordal graphs using only $O_\Delta(n \log^3 n)$ queries. Building on this, Bastide and Groenland [6] strengthened their result by obtaining a reconstruction algorithm for bounded degree k-chordal graphs with $O_{\Delta,k}(n \log n)$ queries (See also Rong et al. [18] for other nearly linear reconstruction algorithms of nearly chordal graphs).[1]

In a more recent work, Bastide and Groenland [7] describe a randomized $O_{\Delta,\tau}(n \log^2 n)$-query algorithm for reconstructing graphs with treelength at most τ. Note that bounded chordality implies bounded treelength [11], but the converse is not true. Thus, their latter result applies to a broader class of graphs, though the algorithm is slightly slower and randomized.

In this paper, we present a deterministic $O_{\Delta,\tau}(n \log n)$-query algorithm for reconstructing graphs of bounded degree Δ and treelength at most τ–our algorithm is deterministic and more efficient by a factor of $\log n$ compared to Bastide and Groenland [7].

Theorem 1 *Given a hidden graph G on n vertices with maximum degree at most Δ and treelength at most τ, there is an algorithm that reconstructs G using at most $O(\Delta^{3\tau+2} \cdot n \log n)$ SP queries.*

In particular, our result implies a deterministic $O_{\Delta,k}(n \log n)$-query algorithm for reconstructing bounded degree graphs with bounded chordality, matching the optimal result for reconstructing graphs with

*Oregon State University, kaudanc@oregonstate.edu

†Oregon State University, amir.nayyeri@oregonstate.edu

‡The authors were supported by NSF grants CCF-1941086 and CCF-2311180.

[1]A graph is k-chordal if all its induced cycles have length at most k; chordal graphs are 3-chordal.

bounded k-chordality [6]. Note that this theorem assumes the parameters Δ and τ are given.

2 Background and notation

In this paper, all graphs are simple, connected, unweighted and undirected. For a graph $G = (V, E)$ and vertices $u, v \in V(G)$, we use $d_G(u, v)$ to denote the length of the shortest path between u and v in G. For a graph $G = (V, E)$ and a vertex $v \in V(G)$, we use $N_G(v)$ to denote the set of all neighbors of v in the graph G and $N_G[v] = N_G(v) \cup \{v\}$, the open and closed neighborhood of v respectively. Analogously, for a graph $G = (V, E)$ and a set $S \subseteq V(G)$, we let $N_G[S] = \{u \in V(G) \mid \exists a \in S, d_G(a, u) \leq 1\}$ and $N_G(S) = N_G[S] \setminus S$ be the closed neighborhood and open neighborhood of S in G respectively. For a graph $G = (V, E)$ and a path $\pi = v_0 v_1 \ldots v_k$ in G, we use $\pi[v_i, v_j]$ to denote the subpath of π starting at v_i and ending at v_j for $0 \leq i \leq j \leq k$. Further, we denote the concatenation of paths $P = p_0, \ldots, p_n$ and $Q = q_0, \ldots, q_m$ where $p_n = q_0$ in graph G with $P \circ Q = p_0, \ldots, p_n, q_0, \ldots, q_n$ resulting in a p_0-q_n walk in G.

Tree decomposition and treelength. A **tree decomposition** of $G = (V, E)$ is an ordered pair $(T, \{V_t \mid t \in V(T)\})$ where T is a tree and $V_t \subseteq V(G)$ is a subset associated with each vertex of T. The tree T and the collection of subsets $\{V_t \mid t \in V(T)\}$, which are often referred to as **bags** of the tree decomposition, must satisfy the following properties.

- Every vertex of G is contained in at least one bag V_t i.e. $V(G) = \bigcup_{t \in V(T)} V_t$.
- For each edge e of G, there is a bag V_t which contains both endpoints of e.
- Let t_1, t_2 and t_3 be three vertices of T such that t_2 lies on the unique path from t_1 to t_3 in T. Then, if a vertex v of G belongs to both V_{t_1} and V_{t_3}, it also belongs to V_{t_2}.

The **treelength** of a graph $G = (V, E)$, denoted $\mathrm{tl}(G)$, is the minimum integer j such that there exists a tree decomposition $(T, \{V_t \mid t \in V(T)\})$ where $d_G(u, v) \leq j$ for all $u, v \in V_t$ for $t \in V(T)$. That is, any two vertices in the same bag of the tree decomposition are at distance at most j.

BFS layers and the layering tree. A key tool that our algorithm uses is the notion of layering tree as used by Chepoi and Dragan [8]. The same notion has appeared in the literature of graph theory and computational geometry and topology under different names such as merge tree, join tree or split tree (see these references: [9, 21]).

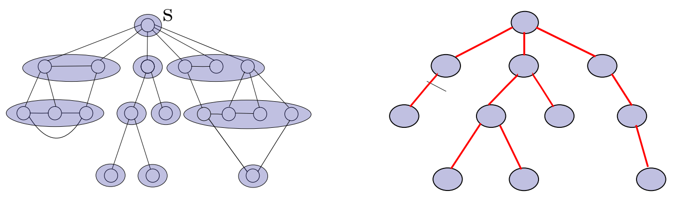


Figure 1: A graph G with its layering tree; parts shaded in purple

Let $G = (V, E)$ be a connected graph, and let $s \in V$ be an arbitrary vertex. We define (BFS) **layers** L_i to be the set of vertices at shortest path distance i from s, i.e. $L_i = \{v \mid d_G(s, v) = i\}$. In particular, $L_0 = \{s\}$. We use notations, $L_{\leq k} := L_0 \cup \ldots \cup L_k$, and $L_{\geq k} := L_k \cup \ldots \cup L_{n-1}$.

For any $i \geq 0$, let $S_i = \{S_i^1, \ldots, S_i^{s_i}\}$ be the set of connected components of $G \setminus L_{\leq i-1}$, noting that $L_{-1} = \emptyset$. In turn, for any $1 \leq j \leq s_i$, let $P_i^j = S_i^j \cap L_i$, the set of vertices in S_i^j that belong to L_i. We call the elements $\mathcal{P}_i = \{P_i^1, \ldots, P_i^{s_i}\}$ the **parts** of G at layer i. It follows that these parts partition L_i. Moreover, the set of all parts, $\mathcal{P} = \bigcup_{0 \leq i} \mathcal{P}_i$, partitions $V(G)$.

We define the **layering tree** $\mathcal{T} = (\mathcal{P}, \mathcal{E})$ of G with respect to s to be the graph whose vertices are the parts of G. For two parts $P, P' \in \mathcal{P}$, $(P, P') \in \mathcal{E}$ if and only if there exists $u \in P$ and $u' \in P'$ such that $(u, u') \in E(G)$.

In the paper we refer to parts both as vertices of the layering tree and as subsets of $V(G)$, but it should be clear from the context which one we refer to. Our algorithm needs to consider the subtrees induced by subsets of $\mathcal{P}$. We use $\mathcal{T}_k$ to refer to the induced subtree $\mathcal{T}[L_{\leq k-1}]$.

Further we define the **length** of a layering tree, denoted $\ell(\mathcal{T})$, to be the maximum diameter of any of its parts, i.e. $\ell(\mathcal{T}) = \max_{P \in \mathcal{P}}(\max_{u,v \in P}(d_G(u, v)))$. By a simple counting argument, the size of each part of a layering tree can be bounded.

Lemma 2 *Let $G = (V, E)$ be a graph with maximum degree Δ, and let $\mathcal{T} = (\mathcal{P}, \mathcal{E})$ be any of its layering trees. The size of each part in $\mathcal{P}$ is at most $\frac{\Delta^{\ell(\mathcal{T})+1}-1}{\Delta-1} \leq \Delta^{\ell(\mathcal{T})+1}$*

Our definition of layering tree is equivalent to the definition used by Dourisbourne and Gavoille [11]–in their definition, two vertices in L_i belong to the same part if and only if there is a path between them in $G[L_{\geq i}]$. They show, in Theorem 8 of their paper [11], the following bound on the length of a layering tree of G with respect to its treelength.

Lemma 3 (Dourisbourne and Gavoille [11]) *Let $G = (V, E)$ be a connected graph with treelength $\mathrm{tl}(G)$, and let $\mathcal{T} = (\mathcal{P}, \mathcal{E})$ be its layering tree (with respect to any vertex s). Then $\ell(\mathcal{T}) \leq 3\mathrm{tl}(G)$.*

Chepoi and Dragan [8] show that the layering tree of a graph can be computed in linear time. In this paper, we focus on query complexity rather than time complexity. However, it is not hard to show that the asymptotic time complexity of our algorithm is the same as its query complexity once the maximum degree and treelength are treated as constants.

3 Reconstruction Algorithm

Let $G = (V, E)$ be a hidden graph with a visible vertex set and a shortest path query oracle. Our algorithm first selects an arbitrary vertex $s \in V(G)$ and uses $n-1$ queries to find $d_G(s, v)$ for all $v \in V(G) \setminus \{s\}$. Based on this information, our algorithm can build the BFS layers $L_0, L_1, \ldots, L_{n-1}$.

Next, our algorithm reconstructs G layer by layer. Specifically, for each vertex $v \in L_i$ our algorithm uses SP queries to find the neighbors of v in $L_i \cup L_{i-1}$.

Let $G_i = G[L_{\leq i-1}]$ be the subgraph of G induced by the first i layers. Also, recall that $\mathcal{T}$ is the layering tree rooted at s, and let ℓ be an upper bound on its length, i.e. $\ell \geq \ell(\mathcal{T})$. Similarly, let $\mathcal{T}_k$ be the subtree of $\mathcal{T}$ induced by the first $k = i - \ell - 2$ layers of $\mathcal{T}$, the layering tree. Our algorithm alternates between extending G_i to G_{i+1} and $\mathcal{T}_k$ to $\mathcal{T}_{k+1}$. Specifically, given G_i, we construct $\mathcal{T}_k$ with no shortest path queries. Then, given $\mathcal{T}_k$ and G_i, we construct G_{i+1} with $|L_i| \cdot \Delta^{O(\ell)} \cdot \log n$ shortest path queries. Our latter algorithm is composed of a binary search type algorithm in $\mathcal{T}_k$ and an exhaustive search within layers k to i, which is similar in spirit to Bastide and Groenland [6] reconstruction for k-chordal graphs.

3.1 Extending the layering tree

We first show that given G_i, one can reconstruct the first $k = i - \ell - 2$ layers of $\mathcal{T}$, denoted $\mathcal{T}_k$. The following lemma is a key structural insight for our construction, showing that if two vertices belong to the same part at layer k then there is a path between them within the next $O(\ell)$ layers of G.

Lemma 4 *Let $G = (V, E)$, let $\mathcal{T} = (\mathcal{P}, \mathcal{E})$ be a layering tree of G, and let $\ell \geq \ell(\mathcal{T})$. Suppose $u, v \in V(G)$ belong to the same part P of $\mathcal{T}$ at layer k. Then, there is a path between u and v in G whose vertices are in $L_{\leq k+\ell+1} \cap L_{\geq k}$, i.e. u and v are connected in $G \setminus (L_{\leq k-1} \cup L_{\geq k+\ell+2})$.*

Proof. Suppose there is no u-v path in $G \setminus (L_{\leq k-1} \cup L_{\geq k+\ell+2})$. Since $u, v \in P$, by definition there exists a u-v path in $G \setminus L_{\leq k-1}$. Let π be a shortest such u-v path. Then π has a vertex t in $L_{k+\ell+2}$. Let $\mu = \lfloor \frac{\ell}{2} \rfloor$, and let x be the last vertex in $L_{k+\mu}$ on the subpath $\pi[u, t]$ and y be the first vertex in $L_{k+\mu}$ on the subpath $\pi[t, v]$. Note that x and y are both in $L_{k+\mu}$ and there exists a path $\pi[x, y]$ in $G \setminus L_{\leq k+\mu-1}$ that is composed of $\pi[x, t]$ and $\pi[t, y]$, both in $G \setminus L_{\leq k+\mu-1}$. Therefore, x and y belong to the same part of $\mathcal{T}$. By Lemma 3, it follows that $d_G(x, y) \leq \ell(\mathcal{T}) \leq \ell$, as the diameter of each part is at most ℓ.

By definition of π, the path $F := \pi[x, t] \circ \pi[t, y]$ is the shortest x-y path whose vertices are in $L_{\geq k}$. Thus, any x-y path Q either has length at least the length of F or contains a vertex of layer L_{k-1}. That is, $d_G(x, y) = \min\{2(\ell + 2 - \mu), 2(\mu + 1)\} \geq \ell + 1$, a contradiction. □

We use Lemma 4 to construct $\mathcal{T}_k$ from G_i. Note that to construct $\mathcal{T}_k$, one only needs to be able to check if two vertices $u, v \in L_k$ are connected in the graph $G \setminus L_{\leq k-1}$. By Lemma 4, to check this connectivity we only need to consider G_i.

Corollary 1 *Let $G = (V, E)$, and let $\mathcal{T} = (\mathcal{P}, \mathcal{E})$ be a layering tree of G. For any $i \geq \ell(\mathcal{T}) + 2$ and $k \leq i - \ell(\mathcal{T}) - 2$, there is an algorithm to compute $\mathcal{T}_k$ from G_i (and no more information about G) with no shortest path queries.*

3.2 Extending graph reconstruction

Next, we show we can reconstruct G_{i+1} given G_i and $\mathcal{T}_k$ for $k = i - \ell - 2$. That is, we can find all the edges that are in G_{i+1} but not in G_i, namely the edges with both endpoints in L_{i+1} or one endpoint in L_i and one in L_{i+1}.

Our algorithm takes two steps to achieve this reconstruction. First, for each vertex $v \in L_i$, it finds the connected component of v in $G \setminus L_{\leq k-1}$; more specifically, the part at layer k that is the ancestor of the part that contains v. This step requires $\Delta^{O(\ell)} \cdot \log n$ shortest path queries. Once this connected component is discovered, our algorithm exhaustively searches for all neighbors of v within this connected component at layers L_i and L_{i-1} to find all its neighbors. This step takes $\Delta^{O(\ell)}$ queries per vertex in L_i.

To prepare for a more formal description of our method, we provide a few definitions:

- For a vertex $v \in V(G)$, we define $\mathbf{part}(v)$ to be the part of the layering tree $\mathcal{T}$ that contains v.
- For a vertex $v \in L_i$ and an integer $k \in \{0, 1, \ldots, i\}$, we define $\mathbf{anc}(v, k)$ to be the part of $\mathcal{T}$ that is the ancestor of $\mathrm{part}(v)$ at depth k.
- For a part P of $\mathcal{T}$, we define $\mathbf{comp}(P)$ to be the subset of vertices contained in P and all of the descendants of P in $\mathcal{T}$. That is, the subset of $V(G)$

that is contained in the subtree of $\mathcal{T}$ rooted at P. Note, if P is a part at layer k then $\mathbf{comp}(P)$ is the connected component of $G \setminus L_{\leq k-1}$ that contains the vertices in P.

Logarithmic search for the connected component. Let $v \in L_i$, and let $k \in \{0, 1, \dots, i\}$. The first step of our algorithm is a logarithmic search to find $\mathrm{anc}(v, k)$. To that end, we use the following classic result on vertex separators of trees [15]. A **(1/2)-cut vertex** of graph G is a vertex whose removal results in connected components whose size is at most $\frac{1}{2}|V(G)|$.

Lemma 5 *Every tree has a $\frac{1}{2}$-cut vertex.*

Lemma 6 *Let $G = (V, E)$, let $\mathcal{T} = (\mathcal{P}, \mathcal{E})$ be a layering tree of G, and let $\ell \geq \ell(\mathcal{T})$. Also, let $i \geq k \geq 0$, and suppose $\mathcal{T}_k$ and G_i are known. For any $x \in L_i$, one can find $\mathrm{anc}(x, k)$ with $O(\Delta^{\ell+2} \cdot \log n)$ queries.*

Proof. Since $\mathcal{T}_k$ is a tree, by Lemma 5, it contains a (1/2)-cut vertex. The proof constructs a sequence of subtrees of $\mathcal{T}_k$, namely $(\mathcal{T}_k^j)_{j=0}^f$ and a sequence of (1/2)-cut vertices $(P_j)_{j=0}^{f-1}$ for these subtrees such that (1) $f \leq \lceil \log n \rceil$, and (2) $\mathcal{T}_k^f$ has only one vertex: $\mathrm{anc}(x, k)$. Initially, $\mathcal{T}_k^0 = \mathcal{T}_k$. The tree $\mathcal{T}_k^{j+1}$ is defined as the connected component of $T_k^j \setminus \{P_j\}$ that contains $\mathrm{anc}(x, k)$.

Since $P_j \in V(\mathcal{T}_k^j)$ is a separator, $\mathcal{T}_k^{j+1}$ is the subtree of $\mathcal{T}_k^j$ that contains the neighbor of P_j which contains the vertex closest to x among all vertices in $N_G(P_j)$, the set of all vertices of $V(G)$ contained in a neighbor of P_j. Therefore, to find $\mathcal{T}_k^{j+1}$ it suffices to query the distance from x to all the vertices in $N_G(P_j)$.

By Lemma 2 there are at most $\Delta^{\ell+1}$ vertices in P_j. Since each vertex in $N_G(P_j)$ has at least one neighbor in P_j and G has maximum degree Δ, we have that

$$|N_G(P_j)| \leq \Delta \cdot |P_j| = \Delta^{\ell+2}$$

Since $\mathcal{T}_k$ has at most n parts and we eliminate at least half of the parts at each step, we know that this search takes at most $\lceil \log_2 n \rceil$ steps, therefore the total query complexity is $\Delta^{\ell+2} \cdot \lceil \log_2 n \rceil = O(\Delta^{\ell+2} \cdot \log n)$ □

Based on the lemma above, we can compute the connected component of each $v \in L_i$ in $G \setminus L_{\leq k-1}$.

Lemma 7 *Let $G = (V, E)$, let $\mathcal{T} = (\mathcal{P}, \mathcal{E})$ be a layering tree of G and let $\ell \geq \ell(\mathcal{T})$. Also, let $i \geq k \geq 0$, and suppose $\mathcal{T}_k$ and G_i are known. There is an algorithm to compute $\mathrm{comp}(P) \cap L_{\leq i}$ for all P at depth k with $O(|L_i| \cdot \Delta^{\ell+2} \cdot \log n)$ queries.*

Proof. To compute $\mathrm{comp}(P) \cap L_{\leq i}$ for all P in $\mathcal{T}$ at depth k, it suffices to find $\mathrm{anc}(u, k)$ for all $u \in L_k \cup \ldots \cup L_i$. Since we have G_i, for each $u \in L_k \cup \ldots \cup L_{i-1}$ we have its shortest path to L_k as part of the BFS tree, therefore we can find $\mathrm{anc}(u, k)$ by checking which part at depth k is connected to u in $G \setminus L_{\leq k-1}$. We do not need any shortest path queries for this step as G_i is given. For vertices $u \in L_i$, we use Lemma 6 to find $\mathrm{anc}(u, k)$ with $O(\Delta^{\ell+2} \cdot \log n)$ queries. Since there are $|L_i|$ vertices at layer i, the total number of queries is as stated in the lemma. □

Brute force search to find neighbors. So far, we have computed $\mathrm{comp}(P) \cap L_{\leq i}$ for all P at depth k. We bound the size of each of these vertex sets.

Lemma 8 *Let $G = (V, E)$, let $\mathcal{T} = (\mathcal{P}, \mathcal{E})$ be a layering tree of G, and let $\ell \geq \ell(\mathcal{T})$. Also, let $P \in \mathcal{P}$ be a part of $\mathcal{T}$ with depth k. Also, let $i \geq k$. Then $|\mathrm{comp}(P) \cap L_{\leq i}| \leq \Delta^{\ell+i-k+2}$.*

Proof. Each vertex in $\mathrm{comp}(P) \cap L_{\leq i}$ is at distance at most $i - k$ from a vertex in P. Since the graph has maximum degree Δ, it follows that there are at most $|P| \cdot \Delta^{i-k+1}$ such vertices. By Lemma 2, $|P| \leq \Delta^{\ell+1}$, which implies the bound of the lemma. □

Putting together Lemma 7 and Lemma 8 our algorithm constructs G_{i+1} from G_i and $\mathcal{T}_k$.

Lemma 9 *Let $G = (V, E)$, let $\mathcal{T} = (\mathcal{P}, \mathcal{E})$ be a layering tree of G, and let $\ell \geq \ell(\mathcal{T})$. Also, let $k \geq 0$ and $i = k+\ell+2$. Suppose G_i and $\mathcal{T}_k$ are given. There is an algorithm to compute G_{i+1} with $O(|L_i|(\Delta^{\ell+2} \log n + \Delta^{4\ell+8}))$ queries.*

Proof. By Lemma 7, $\mathrm{comp}(P) \cap L_{\leq i}$ can be computed for all P in $\mathcal{T}$ at depth k with $O(|L_i| \cdot \Delta^{\ell+2} \cdot \log n)$ queries. To compute edges between L_i and L_{i-1} and within L_i, it suffices to check whether every vertex $v \in L_i$ is adjacent to vertices in the same connected component as v in $G \setminus L_{\leq k-1}$. To do that we check the set $\mathrm{comp}(\mathrm{anc}(v, k)) \cap L_{\leq i}$ for possible neighbors of v. By Lemma 8, these sets are bounded by $\Delta^{\ell+i-k+2} = \Delta^{2\ell+4}$. Overall, we have at most $|L_i| \cdot \Delta^{4\ell+8}$ queries of the latter type. Hence, the total number of queries is $O(|L_i|(\Delta^{\ell+2} \log n + \Delta^{4\ell+8}))$ □

3.3 Proof of the main result

Finally, we are ready to prove our main result.

Proof. [Proof of Theorem 1] First, we select an arbitrary vertex s in $V(G)$ and we query the distance from s to every $v \in V(G) \setminus \{s\}$ to build the layers $L_0, \ldots, L_{n-1}$; this takes $n - 1$ queries. Let $\mathcal{T}$ be the layering tree of G rooted at $\{s\}$ (note, we have not yet computed $\mathcal{T}$). By Lemma 3, we know $\ell(\mathcal{T}) \leq 3\mathrm{tl}(G)$. We set $\ell := 3\tau \geq 3\mathrm{tl}(G)$, where τ is from the theorem statement. Hence $\ell \geq \ell(\mathcal{T})$.

We compute $G_{\ell+2}$ by querying every pair of vertices in $L_{\leq \ell+1}$, taking at most $\Delta^{2\ell+4}$ queries.

For $i \geq \ell+2$, we compute G_{i+1} from G_i in two steps. First, we compute $\mathcal{T}_k$, for $k = i-(\ell+2)$ using no queries, by the algorithm of Corollary 1. Then, by Lemma 9, we compute G_{i+1} using at most $|L_i|(\Delta^{\ell+2} \cdot \log n + \Delta^{4\ell+8})$ queries. Summing over all layers, this quantity is $n(\Delta^{\ell+2} \cdot \log n + \Delta^{4\ell+8})$. Therefore, we reconstruct G with at most

$$\begin{aligned} n - 1 + \Delta^{2\ell+4} + n(\Delta^{\ell+2} \cdot \log n + \Delta^{4\ell+8}) \\ = O(\Delta^{\ell+2} \cdot n \log n) = O(\Delta^{3\tau+2} \cdot n \log n) \end{aligned}$$

queries. □

References

[1] Noga Alon and Vera Asodi. Learning a hidden subgraph. SIAM Journal on Discrete Mathematics, 18(4):697–712, 2005.

[2] Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. SIAM Journal on Computing, 33(2):487–501, 2004.

[3] Dana Angluin and Jiang Chen. Learning a hidden graph using o(log n) queries per edge. In John Shawe-Taylor and Yoram Singer, editors, Learning Theory, pages 210–223, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[4] Dimitris Achlioptas, Aaron Clauset, David Kempe, and Cristopher Moore. On the bias of traceroute sampling: Or, power-law degree distributions in regular graphs. J. ACM, 56(4), July 2009.

[5] Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matúš Mihaľák, and L. Shankar Ram. Network discovery and verification. In Dieter Kratsch, editor, Graph-Theoretic Concepts in Computer Science, pages 127–138, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[6] Paul Bastide and Carla Groenland. Optimal distance query reconstruction for graphs without long induced cycles, 2024.

[7] Paul Bastide and Carla Groenland. Quasi-linear distance query reconstruction for graphs of bounded treelength. In Édouard Bonnet and Pawel Rzazewski, editors, 19th International Symposium on Parameterized and Exact Computation, IPEC 2024, September 4-6, 2024, Royal Holloway, University of London, Egham, United Kingdom, volume 321 of LIPIcs, pages 20:1–20:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[8] Victor Chepoi and Feodor Dragan. A note on distance approximating trees in graphs. European Journal of Combinatorics, 21(6):761–766, 2000.

[9] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. Computational Geometry, 24(2):75–94, 2003. Special Issue on the Fourth CGC Workshop on Computational Geometry.

[10] Luca Dall'Asta, Ignacio Alvarez-Hamelin, Alain Barrat, Alexei Vázquez, and Alessandro Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. Theoretical Computer Science, 355(1):6–24, 2006. Complex Networks.

[11] Yon Dourisboure and Cyril Gavoille. Tree-decompositions with bags of small diameter. Discret. Math., 307(16):2008–2029, 2007.

[12] W. Ellens, F.M. Spieksma, P. Van Mieghem, A. Jamakovic, and R.E. Kooij. Effective graph resistance. Linear Algebra and its Applications, 435(10):2491–2506, 2011. Special Issue in Honor of Dragos Cvetkovic.

[13] Jotun J. Hein. An optimal algorithm to reconstruct trees from additive distance data. Bulletin of Mathematical Biology, 51(5):597–603, 1989.

[14] Jeremy Hoskins, Cameron Musco, Christopher Musco, and Babis Tsourakakis. Inferring networks from random walk-based node similarities. In Advances in Neural Information Processing Systems, volume 31, 2018.

[15] Camille Jordan. Sur les assemblages de lignes. Journal für die reine und angewandte Mathematik, 70:185–190, 1869.

[16] Sampath Kannan, Claire Mathieu, and Hang Zhou. Graph reconstruction and verification. ACM Trans. Algorithms, 14(4), August 2018.

[17] Claire Mathieu and Hang Zhou. Graph reconstruction via distance oracles. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, Automata, Languages, and Programming, pages 733–744, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[18] Guozhen Rong, Wenjun Li, Yongjie Yang, and Jianxin Wang. Reconstruction and verification of chordal graphs with a distance oracle. Theoretical

Computer Science, 859:48–56, 2021.

[19] Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In Marcus Hutter, Rocco A. Servedio, and Eiji Takimoto, editors, Algorithmic Learning Theory, pages 285–297, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[20] M.S. Waterman, T.F. Smith, M. Singh, and W.A. Beyer. Additive evolutionary trees. Journal of Theoretical Biology, 64(2):199–213, 1977.

[21] Suyi Wang, Yusu Wang, and Rephael Wenger. The js-graphs of join and split trees. In Proceedings of the Thirtieth Annual Symposium on Computational Geometry, SOCG'14, page 539–548, New York, NY, USA, 2014. Association for Computing Machinery.

The Marco Polo Problem: A Combinatorial Approach to Geometric Localization*

Ofek Gila† Michael T. Goodrich† Zahra Hadizadeh‡ Daniel S. Hirschberg† Shayan Taherijam†

Abstract

We introduce and study the ***Marco Polo problem***, which is a combinatorial approach to geometric localization. In this problem, we are told there are one or more points of interest (POIs) within distance n of the origin that we wish to localize. Given a mobile search point, Δ, that is initially at the origin, a localization algorithm is a strategy to move Δ to be within a distance of 1 of a POI. In the combinatorial localization problem we study, the only tool we can use is reminiscent of the children's game, "Marco Polo," in that Δ can issue a ***probe*** signal out a specified distance, d, and the search algorithm learns whether or not there is a POI within distance d of Δ. For example, we could imagine that POIs are one or more hikers lost in a forest and we need to design a search-and-rescue (SAR) strategy to find them using radio signal probes to a response device that hikers carry. Unlike other known localization algorithms, probe responses do not inform our search algorithm of the direction or distance to a POI. The optimization problem is to minimize the number of probes and/or POI responses, as well as possibly minimizing the distance traveled by Δ. We describe a number of efficient combinatorial Marco Polo localization strategies and we analyze each one in terms of the size, n, of the search domain. Moreover, we derive strong bounds for the constant factors for the search costs for our algorithms, which in some cases involve computer-assisted proofs. We also show how to extend these strategies to find all POIs using a simple, memoryless search algorithm, traveling a distance that is $\mathcal{O}(\log k)$-competitive with the optimal traveling salesperson (TSP) tour for k POIs.

1 Introduction

In the children's game, "Marco Polo," a group of children are playing in a swimming pool. One player is chosen as "it," who closes their eyes and tries to find and tag one of the other players. The "it" player periodically calls out "Macro" and the other players who can hear this call must respond with "Polo." The "it" player moves based on this "Marco-Polo" call-and-response protocol until getting close enough to another player to tag them, which ends this player's turn being "it." See, e.g., [20].

In this paper, we introduce and study the ***Marco Polo problem***, which is a combinatorial approach to geometric localization motivated from the "Marco Polo" children's game. In this problem, we start with a search point, Δ, at the origin, with one or more points of interest (POIs) at a distance n from the origin and our goal is to move Δ to be within distance 1 of a POI.[1] We may periodically send probes out a specified distance, d, and we learn whether or not there is a POI within distance d from Δ. Optimization goals include minimizing the number of probes and minimizing the total distance traveled by Δ.

We can motivate the Marco Polo problem, for example, from a search-and-rescue (SAR) scenario. Suppose a hiker, who we'll call "Alice," is lost and stationary at a point of interest (POI) in a large forest and we would like to find her using an efficient SAR strategy. Assume Alice has a wireless device, similar to an Apple AirTag, which can respond to probes sent from a searcher, Δ. In particular, suppose Δ can send a probe at a given power level, which sends an omni-directional signal out to a known radius (depending on the power level), and if Alice is present inside the circle determined by this probe, then Δ will receive a positive response. Such probes use up power, however, both for Δ and for Alice's tracking device; hence, the goal is to devise a sequence of probes that minimizes the number of probes needed to locate Alice to a specified accuracy. (See Figure 1.)

We also consider generalizations of this problem, such as if there were multiple POIs. One can imagine other applications besides search and rescue for the combinatorial searching problem, including locating animals wearing tracking collars, finding radioactive sources, or identifying anomalous readings in wireless sensor networks.

Related Prior Work. Although we are not familiar with any prior work on the Marco Polo problem itself,

*This research supported in part by NSF grant 2212129.

†University of California, Irvine, {ogila,goodrich,dhirschb,staherij}@uci.edu
‡University of Rochester, zhadizadeh99@gmail.com

[1]This formulation is made without loss of generality, as we could just as easily normalize the search problem so that there is a POI within distance 1 of Δ and we are interested in moving Δ to be within distance ε of a POI, for a given $\varepsilon > 0$.

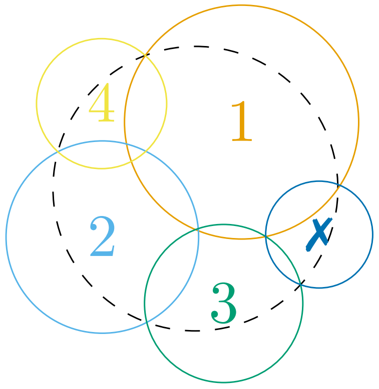


Figure 1: An example sequence of probes for an instance of the Marco Polo problem. In this example, the first four probes are negative and the POI (marked with an ✗) is found to within distance 1 on the fifth probe.

our work nevertheless falls into a rich area of study known as ***localization algorithms***; see, e.g., the survey by Han, Xu, Duong, Jiang, and Hara [12]. We discuss how our work compares to a wealth of existing prior work in Section A.

Problem Definition. In the ***Marco Polo problem***, there are $k \geq 1$ points of interest (POIs) with unknown positions, with at least one that is within a distance, n, of a point, O, called the ***origin***, which is the initial position of a mobile search point, Δ. A ***probe***, $p(x, y, d)$, is a query that asks if there are any POIs within distance d of the current location, (x, y), for Δ. The goal is to design a search strategy to move Δ to be within a distance of 1 of a POI. Given this setup, there are a number of possible constraints that define instances of the Marco Polo problem, including:

- A search algorithm can be ***incremental***, which finds all k POIs one at a time, or ***coordinated***, which finds multiple POIs in a coordinated fashion.
- An algorithm is ***memoryless*** with respect to the current search area if its state is restricted to the area determined by the previous successful probe.

There are a number of metrics we can use to measure the effectiveness of the strategy, including the following:

- $P(n)$: the number of probes issued.
- $R_{\max}$: the maximum number of times a POI responds to a probe.
- $D(n)$: the total distance traveled by Δ.

These efficiency measures can conflict, of course, in that a strategy that minimizes, say, $R_{\max}$, may have poor bounds for $P(n)$ and/or $D(n)$. Such a trade-off may nevertheless be worthwhile, however, such as in an SAR scenario where the batteries are running out in a hiker's device and responses are costly.

Our Results. In this paper, we provide a number of efficient algorithms for solving instances of the Marco Polo problem. In Section D we show a simple double-binary-search algorithm that uses $2\lceil \log n \rceil + \mathcal{O}(1)$ probes,[2] but which is based on unrealistic assumptions. Instead, under more realistic assumptions about conditions regarding the search space, we provide a sequence of algorithms, starting with simple algorithms based on hexagon geometries, which we call "hexagonal algorithms," and progressing to more sophisticated recursive strategies based on progressively shrinking probes at each level of recursion. This ultimately results in an algorithm that makes at most $3.34\lceil \log n \rceil$ probes using a monotonically spiraling search strategy at each recursive level. Using computer-assisted proof techniques, we then show that it is possible to find a POI using $2.53\lceil \log n \rceil$ probes. Although this strategy sacrifices the simplicity of a monotonically spiraling search, it performs competitively with our proven lower bound of $2.4\lceil \log n \rceil$ probes for progressive shrinking algorithms. We also provide various algorithms that reach different trade-offs between the number of probes made and the total distance traveled by Δ, include one algorithm that travels a total distance of at most $6.02n$, which is less than the circumference of the original search area. We then provide a family of algorithms that are able to restrict the total number of POI responses, $R_{\max}$, to any desired value from 1 to $\lceil \log n \rceil$ while minimizing the number of probes made. Finally, we present a strategy to extend our incremental algorithms to find all POIs while traveling a distance that is $\mathcal{O}(\log k)$-competitive with an optimal traveling salesperson (TSP) tour. We include experiments supporting all our results in Section F.

2 Finding One POI

We describe a series of progressively more efficient algorithms that minimize the number of probes needed for finding a single POI. We assume that there may be multiple POIs, either within the search region of radius n or outside of it, but we are initially interested only in finding one of them. Later we will discuss how to extend these algorithms to find all POIs.

Hexagonal Algorithms. Our first algorithms, which we call ***hexagonal algorithms***, are defined in terms of a tiling of our search area with hexagons of radius $n/2$. There are seven such hexagons, which can each be probed with radius-$n/2$ probes until a probe succeeds, which then allows us to make a recursive call to an $n/2$-sized subproblem. Such an algorithm will take $7\lceil \log n \rceil$ probes in the worst case where POIs are always in the last hexagon probed in each recursive level. We can improve this to $6\lceil \log n \rceil$ probes by not probing the last hexagon, since a POI ***must*** be there

[2] All of the logarithms used in this paper are base 2.

if the other six probes fail. We refer to this hexagonal algorithm as Algorithm 1.

There is a better hexagonal algorithm, however, which involves first probing the upper two quadrants, which can be done with radius $n/\sqrt{2}$ probes, eliminating 3 hexagons, and then probing 3 of the 4 remaining hexagons as before. We refer to this modified hexagonal algorithm as Algorithm 2. See Figure 2. If POIs are in one of the two larger probes, we reduce the problem less than before, only by a factor of $\sqrt{2}$, but with fewer probes (at most 2). This turns out to be a better tradeoff, so that in the worst case where POIs are all in the last hexagon probed at each level and our algorithm makes at most $5\lceil\log n\rceil$ probes.

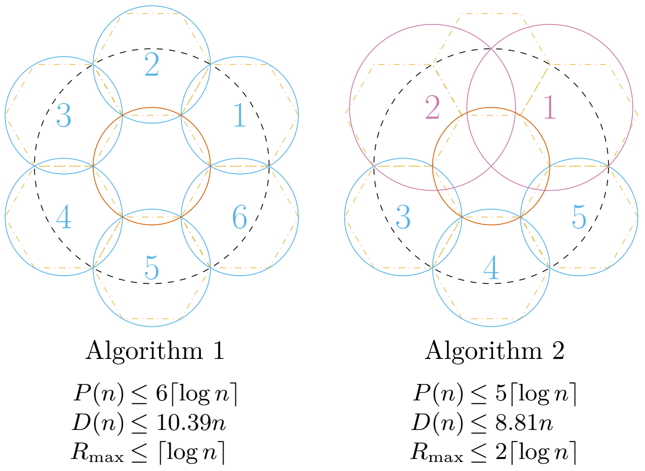

2
3
1
4
6
5
Algorithm 1
$P(n) \le 6\lceil\log n\rceil$
$D(n) \le 10.39n$
$R_{\max} \le \lceil\log n\rceil$
2
1
3
5
4
Algorithm 2
$P(n) \le 5\lceil\log n\rceil$
$D(n) \le 8.81n$
$R_{\max} \le 2\lceil\log n\rceil$

Figure 2: Two simple hexagonal algorithms. Algorithm 1 performs probes of radius $n/2$ along the center of each of the 6 outer hexagons, while Algorithm 2 first performs two radius $n/\sqrt{2}$ probes in the upper two quadrants before performing the remaining probes of Algorithm 1.

Distance Traveled. To determine the total distance traveled by Δ, $D(n)$, in any probe-sequence algorithms, including our hexagonal algorithms, let us consider the first layer of probes, i.e., those performed when the search radius is n. In particular, consider the k-th probe (starting from 1) as having radius $r_k = \rho_k n$, where ρ_k represents the proportionality factor for the size of each probe relative to the (current) search radius. Let d_k denote the distance Δ travels to perform the k-th probe. Assuming we always first succeed on our k-th probe, the total distance traveled is determined by $D(n) = d_k + D(\rho_k n)$, where we assume that $D(n) = b_k n$, for some constant b_k. Solving this, we find that if indeed the first probe that succeeds is always the k-th probe, then $b_k = d_k/(1-\rho_k)$. Using this, we determine that:

$$D(n) \le \max_k \frac{d_k}{1-\rho_k} n. \qquad (1)$$

To determine a good bound for this maximum, we use a computer-assisted proof technique to compute this value for each algorithm.[3] The maximum number of responses is achieved when the POI is always inside the largest probe, $\rho_{\max}$, resulting in:

$$R_{\max} \le \lceil \log_{1/\rho_{\max}} n\rceil \le -\frac{1}{\log \rho_{\max}}\lceil\log n\rceil. \qquad (2)$$

Progressively Shrinking Probes. As shown in Algorithm 2, we may be able to achieve better results by probing from differently sized circles before ever receiving a positive response. Indeed, we can do better by using probes that get progressively smaller, so that if we spend many probes to be able to recurse to a smaller search area, we should at least reduce the remaining area by a larger factor. The question remains how to choose the sizes of the probes.

Let the total number of probes required to find a POI starting with a search area of radius n be $P(n)$, and let the k-th probe (starting from 1) have radius $r_k = \rho_k n$, where ρ_k represents the proportionality factor for the size of each probe relative to current search radius. Assuming we always first succeed on our k-th probe, $P(n)$ is determined by the recurrence relation, $P(n) = k + P(\rho_k n)$, where $P(n) = c_k \log n$, for some constant, $c_k > 0$, and our total number of probes will be $k + P(\rho_k n)$, resulting in

$$\begin{aligned} P(n) &= k + P(\rho_k n) = k + c_k \log(\rho_k n) \\ &= k + c_k \log \rho_k + c_k \log n \\ &= k + c_k \log \rho_k + P(n). \end{aligned}$$

Hence, $0 = k + c_k \log \rho_k \implies c_k = -\frac{k}{\log \rho_k}$.

In the worst case, all POIs will be in the k-th probe such that the total number of probes is maximized, i.e., such that c_k is maximized, so we pick c_k such that it is the same for all k. This is done by setting $\rho_k = \rho_1^k$, where ρ_1 is the proportionality constant of the first probe, resulting in the overall number of probes being[4]

$$P(n) \le -\frac{1}{\log \rho_1} \log n. \qquad (3)$$

Note that our equations for the maximum number of probes and responses, Equations (2) and (3), respectively, are nearly identical—the latter occurring when the POI is always in the first probe.

It is worth noting that all the above calculations make no assumptions about the correctness of the algorithm, i.e., the ability of the algorithm to always find a POI when one exists. Any *correct* algorithm must be able to cover the entire search area, including its perimeter, a fact that we will use to derive a lower bound on the number of probes required to find a POI.

Further, unlike the calculations for Equation (3) which assumed that the final probe locating the POI

[3] We provide pseudocode for this algorithm in Section B.
[4] The inequality arises from the omission of the last probe.

has radius which is exactly 1, it is possible for the final probe to have a smaller radius, i.e., that the algorithm obtained some excess precision. As we will discuss in more detail in Section C, this excess precision results in at most an additional constant number of probes, where the constant is dependent on the maximum number of probes the algorithm performs per layer, which we refer to as $k_{\max}$, obtaining:

Lemma 1 *A correct progressively shrinking algorithm may require at most* $-\frac{1}{\log \rho_1} \log n + \mathbf{k}_{max} - \mathbf{1}$ *probes to find a POI in a search area of radius n.*

The proof, as well as examples and more details are provided in Section C. For simplicity, we will ignore this additional constant in the main text.

A Lower Bound for Progressive Shrinking. With this result in mind, we can determine a lower bound on the number of probes required to find a POI. Since it is always possible for all POIs to be along the area's perimeter, any correct algorithm must at least probe the perimeter of the search area. To maximize the perimeter coverage of each probe, we place it such that its diameter is a chord of the circle, as shown in Figure 3, and determine the minimum value of ρ_1 required to probe the entire perimeter of the circle to be approximately 0.74915. This results in a lower bound of $P(n) > 2.40001 \log n$ probes.[5]

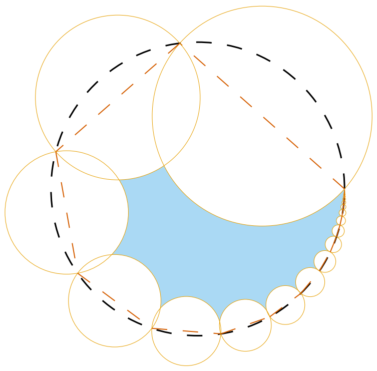


Figure 3: The optimal placement of progressively decreasing probes of radius proportional to ρ_1^k in order to cover the circumference of the search area. ρ_1 must be at least 0.74915 to fully cover the perimeter, as shown. There exists uncovered search area, depicted in blue.

Chord-Based Shrinking Algorithms. The placement strategy for Δ used to show the lower bound of roughly $2.4 \log n$ probes for progressive shrinking algorithms is not a valid approach for an upper-bound for an algorithm, due to all the internal uncovered area, see Figure 3. Nevertheless, the approach of placing diameters of each probe as chords of a circle can work with large enough probes, which we can determine by tuning ρ_1. We refer to our next algorithm, which is a progressive shrinking algorithm, as Algorithm 3. In this algorithm, we numerically determine, using a computer-assisted proof,[6] that the minimum value of ρ_1 that leaves no uncovered area is approximately 0.844, reducing the number of probes to $P(n) < 4.08 \log n$. See Figure 4 (left). Δ's route maintains the property that it is monotonic in a counterclockwise orientation.

However, if we allow for nonmonotonic routes, we can place the two largest probes side by side and alternate the next two probes on either side, which leads to significant improvements, which we refer to as Algorithm 4. This algorithm is able to further reduce the number of probes to $P(n) < 3.54 \log n$, and is depicted in Figure 4 (right).

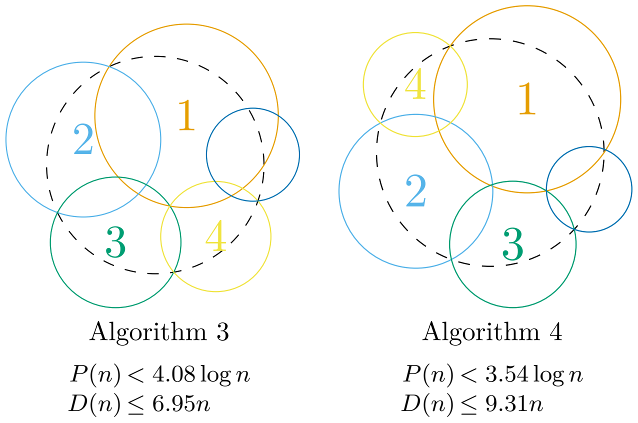


Figure 4: Algorithms 3 and 4 both place probes such that their diameters are chords of the search area circle. Algorithm 3 places the probes in order of decreasing size going counter-clockwise, while Algorithm 4 places the probes such that they overlap as little as possible.

An optimization can be made to the total distance traveled by Δ for Algorithms 3 and 4. If a POI is determined to be in the last area, because previous probes in a recursive level are all negative, Δ can travel directly to the center of the first probe within the next recursive layer instead of the center of the last probe, since the last probe does not need to be performed. Accordingly, Algorithms 3 and 4 tradeoff the number of probes for flight distance. See Figure 4.

Higher-count Monotonic-path Algorithms. Both Algorithms 3 and 4 use very few probes (i.e., 5) at each recursive layer, while we know from Equation (3) that we can introduce more probes with geometrically decreasing radii without increasing $P(n)$. However, decreasing the value of ρ_1 in Algorithm 4 will not only introduce a gap in the perimeter but would also introduce an internal gap, which would require another placement scheme entirely to fill. Our next idea is to

[5]The exact coefficient c can be determined numerically by solving the following equation: $\sum_{k=1}^{\infty} \sin^{-1} 2^{-\frac{k}{c}} = \pi$, which we approximated using Wolfram Mathematica.

[6]The code determining ρ_1 and the maximum distance traveled is available at `https://github.com/ofekih/DroneSearching`.

begin with one large central probe before placing the remaining probes along the perimeter monotonically as in Algorithm 3, which should also have good flight-path performance for Δ. Intuitively, the large central probe greatly reduces the probe radius required to avoid internal gaps, allowing for more probes to be placed. Indeed, by placing the remaining probes such that their diameters are chords of the search area, as was done in Algorithm 3, leads to Algorithm 5, which uses up to 8 probes at each recursive level. While this algorithm improves upon Algorithm 3, requiring only $P(n) < 3.83 \log n$ probes, and improved flight distance, it performs worse than Algorithm 4 in its total number of probes. See Figure 6. In order to take advantage of even more probes, we observe that we cover the circumference of the search area at a much faster rate than the circumference of the central probe. In other words, if we reduce the probe radii, we would still be able to cover the search radius circumference, but we would introduce internal gaps between the outer probes and the central probe. Ideally, we would like the rate at which they cover the inner and the outer circumferences to be the same, such that the chords made with the outer and inner circles cover the same angle. The geometric reasoning is shown in Figure 5.

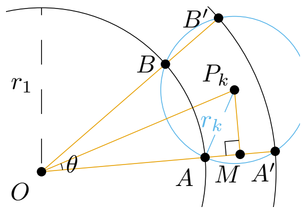
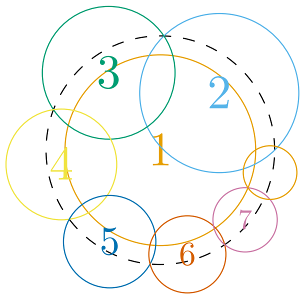
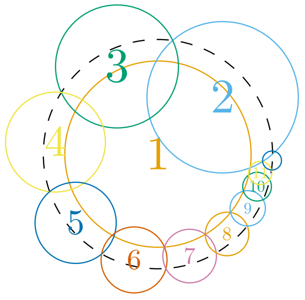


Figure 5: A diagram showing the relationship between the k-th circle, centered at P_k, the first circle, centered at O, and the search radius, also centered at O.

In particular, we must determine at what position, P_k, to place the center C_k of the k-th probe with radius r_k. For simplicity, we assume that the search area is a unit circle centered at the origin, O, and that the first probe, C_1, has radius $r_1 < 1$. Let A, B, A', and B' refer to the points where C_k intersects C_1 and the search area, respectively, and let M be the midpoint of $\overline{AA'}$. Covering both the inner and outer circumferences at the same rate implies that $\angle AOP_k = \angle A'OP_k$, i.e., that $\overline{OA}$ and $\overline{OA'}$ are colinear. Since $\triangle AA'P_k$ is isosceles, we know that $\overline{P_kM}$ is perpendicular to $\overline{AA'}$ so $|\overline{P_kM}| = \sqrt{r_k^2 - (\frac{1-r_1}{2})^2}$ and consequently $\theta = \arctan(\frac{\sqrt{r_k^2-(\frac{1-r_1}{2})^2}}{1-\frac{1-r_1}{2}})$. Thus, by moving the outer probes inward, towards the origin, to their new centers as described above, we are able to "turn it up to 11" and achieve Algorithm 6. See Figure 6.

Darting Non-Monotonic Algorithms. Up to this point, our best two algorithms for minimizing $P(n)$ are Algorithm 4, which is non-monotonic and uses its few probes very efficiently, and Algorithm 6, which is monotonic with a counterclockwise spiral of probes such that, despite having significant overlap, it is able to squeeze in nearly three times as many probes and achieve better performance. The question remains whether it is possible to achieve even better query performance at the expense of monotonicity, giving up on optimizing Δ's flight distance so as to achieve even better probe complexity. We refer to such algorithms as being ***darting algorithms*** and discuss them in Section E. The best two darting algorithms are highlighted in Figure 7.





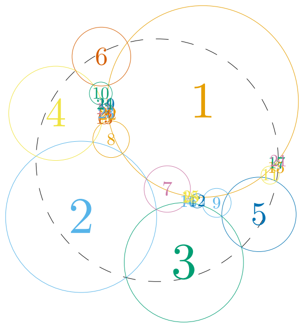
Algorithm 5

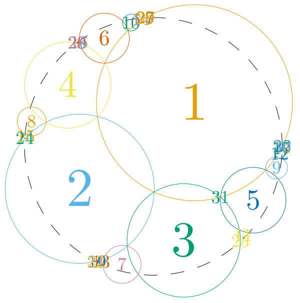
$P(n) < 3.83 \log n$
$D(n) \leq 6.72n$

Algorithm 6

$P(n) < 3.34 \log n$
$D(n) \leq 6.02n$

Figure 6: Algorithms 5 and 6 both perform probes counter-clockwise along the circumference of a central probe. Algorithm 5 places the probes so that diameters are chords of the search area, while Algorithm 6 balances the coverage rate of the inner and outer circumferences. Out of all our algorithms, Δ travels the least distance (in the worst case) using Algorithm 6.





Algorithm 7

$P(n) < 2.93 \log n$
$D(n) \leq 25.8n$

Algorithm 8

$P(n) < 2.53 \log n$
$D(n) \leq 45.4n$

Figure 7: Algorithms 7 and 8 both use computer-assisted probe placement to efficiently cover the search area. Algorithm 7 begins with Algorithm 4, removing the final probe, while Algorithm 8 uses a differential evolution algorithm to place the initial six probes. The remaining probes are placed greedily as discussed in an appendix. Algorithm 8 achieves our best probe results.

Theorem 2 *Progressive shrinking algorithms for Δ searching a circular region of radius n have a lower bound of $2.4\log n$ for $P(n)$, and we can achieve upper bounds as shown in Figures 2, 4, 6, and 7.*

Reducing The POIs' Responses. Our exploration has so far focused on reducing the total number of probes, $P(n)$, required to find a POI. In this section, we consider the case where POIs are limited in their number of responses $R_{\max}$ to probes, e.g., due to battery constraints. Namely, consider a scenario where $R_{\max}$ is a fixed value where $R_{\max} \geq 1$. Our goal is to design a search strategy that minimizes the number of probes, $P(n)$, while ensuring that the number of responses from the POIs does not exceed $R_{\max}$. Recalling from Equation (2), the amount of responses is determined by the size of the largest probe at each recursive layer, so there is no benefit to using differently sized probes. We present a family of hexagonal algorithms which probe the search space using hexagonal lattices with L layers of hexagonal rings. See Figure 8.

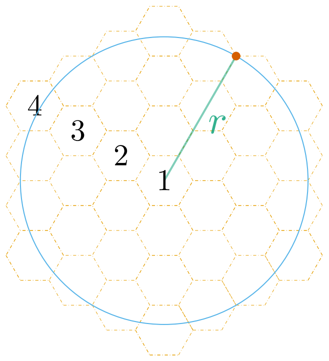


Figure 8: A hexagonal lattice with $L = 4$ layers of rings.

Increasing the number of rings increases the number of hexagons per lattice, which in turn reduces each hexagon's size. These smaller hexagons require smaller probes, reducing the number of responses from the POI but increasing the (worst case) total number of probes. Algorithm 1 can be thought of as one such algorithm where $L = 2$. We describe our family of algorithms by the following routine:

1. Cover the search area with an L-layer lattice.
2. Sequentially probe each hexagon in the lattice until receiving a positive response.
3. Repeat until reaching an area with radius 1.

Theorem 3 *If a POI is only allowed to respond at most $1 \leq R_{max} \leq \lceil \log n \rceil$ times, then a hexagonal algorithm is able to find them using at most*

$$P(n) \leq 6R_{max}\binom{\lceil \frac{2n^{\frac{1}{R_{max}}}+2}{3} \rceil}{2} \text{ probes,} \tag{4}$$

by using a hexagonal lattice with

$$L = \lceil \frac{2n^{\frac{1}{R_{max}}}+2}{3} \rceil \text{ layers.} \tag{5}$$

Fact 4 *A hexagon with side length s is covered by a circumscribed circle with radius s.*

Lemma 5 *A circle with radius r can be covered by an L-layer lattice of hexagons with side length $s = \frac{2r}{3L-2}$.*

Proof. See Section C. ☐

Fact 6 *An L-layer lattice has $1+6\binom{L}{2}$ hexagons, determined by the L-th centered hexagonal number.*

Proof. (of Theorem 3) During step 1, we cover the radius r search area with a lattice of hexagons of side length s, where $s = \frac{2r}{3L-2}$ from Lemma 5. From Fact 4, we probe each of these hexagons with a circle of radius s, reducing the search area of the next recursive layer by a factor of $\frac{3L}{2} - 1$. As the lattice grows, we reduce the search area by a greater factor. In order to finally reach a circle of radius 1, we require $\lceil \log_{\frac{3L}{2}-1}(n) \rceil$ recursive rounds, each requiring one response from the POI. Solving for L, we obtain Equation (5). Finally, since we can probe all the hexagons but one per round, we probe at most $6\binom{L}{2}$ hexagons per round (see Fact 6), and since there are $R_{\max}$ rounds, the total number of probes is bounded by $P(n) \leq 6R_{\max}\binom{L}{2}$. ☐

Corollary 7 *The total number of probes required to find a POI is bounded as follows:*

1. *If $R_{max} = 1$, then $P(n) \leq \frac{4n^2}{3} + 6n + 6 = \mathcal{O}(n^2)$.*
2. *If $R_{max} = 2$, then $P(n) \leq \frac{8n}{3} + 12\sqrt{n} + 12 = \mathcal{O}(n)$.*
3. *If $R_{max} = \lceil \log n \rceil$, then $P(n) \leq 6\lceil \log n \rceil$.*[7]

3 Finding All POIs

Once one POI is found, we shut off the tracking device so that it stops responding to Δ's probes, yet the question remains—how should Δ search for the rest of the POIs? Since Δ is stateless with respect to its current search area, no knowledge is gained about other POIs from the search for the first POI. Even if Δ was able to retain its search path so far, the result of probes are binary; any previous positive probe result cannot be relied on. And even if the probe were able to determine the exact *quantity* of POIs within the area, it is possible for them to be in different search areas early on, resulting in the probe needing to perform its search almost entirely from scratch. This can trivially happen if the POIs are far from each other, but may also happen if the POIs are

[7]This follows from the fact that $n^{\frac{1}{\lceil \log n \rceil}} \leq 2$.

close to each other but on opposite sides of a search area boundary. Does there exist a ***coordinated*** search strategy that performs better than the *incremental* strategy of independently searching for each POI?

Let us assume we have a search algorithm $\mathcal{A}(n)$ that is able to find a single POI in $P(n) \leq c\lceil \log n \rceil$ probes, traveling a distance of $D(n) \leq dn$. We use $\mathcal{A}$ as a subroutine of the following strategy:

1. Find an arbitrary POI using $\mathcal{A}(n)$.
2. Shut off the tracking device of the found POI.
3. Without moving Δ, re-probe the area at radius 2, 4, 8, etc., until a probe returns a positive result (i.e., another POI is found).
4. Invoke $\mathcal{A}$ using this new radius to find another POI.
5. Repeat steps 2 to 4 until all POIs are found.

Assuming there are k POIs, we have:

Theorem 8 *The total number of probes (P_{tot}) and the total distance traveled by Δ (D_{tot}) during the memoryless search algorithm for all k POIs is at most:*

$$P_{tot} \leq c\lceil \log n \rceil + (c+1)(k-1)\lceil \log \overline{e} \rceil,$$
$$D_{tot} \leq dn + 2dE,$$

where $E < OPT(\lceil \log k \rceil + 1)$, $\overline{e} = \frac{E}{k-1}$, and OPT is the optimal tour length for the traveling salesperson problem (TSP) on the k POIs.

Proof. See Section C. □

References

[1] Esther M Arkin, Michael A Bender, Sándor P Fekete, Joseph SB Mitchell, and Martin Skutella. The freeze-tag problem: how to wake up a swarm of robots. *Algorithmica*, 46:193–221, 2006. doi:10.1007/s00453-006-1206-1.

[2] Esther M. Arkin, Michael A. Bender, and Dongdong Ge. Improved approximation algorithms for the freeze-tag problem. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, page 295–303, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/777412.777465.

[3] Nicolas Bonichon, Arnaud Casteigts, Cyril Gavoille, and Nicolas Hanusse. Freeze-tag in L_1 has wake-up time five, 2024. arXiv:2402.03258.

[4] Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and Saeed Odak. Euclidean freeze-tag problem on plane. In *36th Canadian Conference on Computational Geometry (CCCG)*, pages 199–205, 2024.

[5] Judith Brecklinghaus and Stefan Hougardy. The approximation ratio of the greedy algorithm for the metric traveling salesman problem. *Operations Research Letters*, 43(3):259–261, 2015. doi:10.1016/j.orl.2015.02.009.

[6] Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943. URL: http://www.jstor.org/stable/2235930.

[7] Ding-Zhu Du and Frank Kwang-Ming Hwang. *Combinatorial Group Testing and Its Applications.* World Scientific, 2nd edition, 2000. doi:10.1142/4252.

[8] Jens Niklas Eberhardt, Nikolas Peter Breuckmann, and Christiane Sigrid Eberhardt. Multi-stage group testing improves efficiency of large-scale COVID-19 screening. *Journal of Clinical Virology*, 128:104382, 2020. doi:10.1016/j.jcv.2020.104382.

[9] David Eppstein, Michael T Goodrich, and Daniel S Hirschberg. Improved combinatorial group testing algorithms for real-world problem sizes. *SIAM Journal on Computing*, 36(5):1360–1375, 2007. doi:10.1137/050631847.

[10] Michael T Goodrich and Daniel S Hirschberg. Improved adaptive group testing algorithms with applications to multiple access channels and dead sensor diagnosis. *Journal of Combinatorial Optimization*, 15:95–121, 2008. doi:10.1007/s10878-007-9087-z.

[11] Mikael Hammar, Bengt J Nilsson, and Mia Persson. The online freeze-tag problem. In *LATIN 2006: Theoretical Informatics: 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006. Proceedings 7*, pages 569–579. Springer, 2006. doi:10.1007/11682462_53.

[12] Guangjie Han, Huihui Xu, Trung Q Duong, Jinfang Jiang, and Takahiro Hara. Localization algorithms of wireless sensor networks: a survey. *Telecommunication Systems*, 52:2419–2436, 2013. doi:10.1007/s11235-011-9564-7.

[13] Stefan Hougardy and Mirko Wilde. On the nearest neighbor rule for the metric traveling salesman problem. *Discrete Applied Mathematics*, 195:101–103, 2015. 12th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW 2013). doi:10.1016/j.dam.2014.03.012.

[14] Jyh-How Huang, Saqib Amjad, and Shivakant Mishra. CenWits: a sensor-based loosely coupled search and rescue system using witnesses. In *3rd ACM International Conference on Embedded Networked Sensor Systems*, SenSys '05, page 180–191, 2005. doi:10.1145/1098918.1098938.

[15] Richard Kershner. The number of circles covering a set. *American Journal of Mathematics*, 61(3):665–671, 1939. doi:10.2307/2371320.

[16] E.B. Martinson and F. Dellaert. Marco Polo localization. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 1960–1965, 2003. doi:10.1109/ROBOT.2003.1241881.

[17] Lehilton Lelis Chaves Pedrosa and Lucas de Oliveira Silva. Freeze-tag is NP-hard in 3d with L_1 distance. *Procedia Computer Science*, 223:360–366, 2023. doi:10.1016/j.procs.2023.08.248.

[18] Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977. doi:10.1137/0206041.

[19] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997. doi:10.1023/A:1008202821328.

[20] Wikipedia contributors. Marco Polo (game), 2025. URL: https://en.wikipedia.org/wiki/Marco_Polo_(game).

[21] Wikipedia contributors. Tag (game) – freeze tag, 2025. URL: https://en.wikipedia.org/wiki/Tag_(game)#Freeze_tag.

A Additional Related Work

As mentioned in the introduction, the Marco Polo problem falls into a rich area of study known as ***localization algorithms***; see, e.g., the survey by Han, Xu, Duong, Jiang, and Hara [12]. Our approach differs from the approaches used in this prior work, however, in that we are interested in strictly combinatorial strategies, where all we learn is a single in-or-out result from each probe, rather than, say, range and/or directional results, such as in the work by Martinson and Dellaert [16].

The Marco Polo problem is related to combinatorial group testing, see, e.g., [7, 9, 10], which was originally directed at identifying WWII soldiers with syphilis [6] and was recently applied to COVID-19 testing [8]. In this problem, one is given a set of n items, at most d of which are "defective." Subsets of the items (such as blood samples) can be pooled and tested as a group, such that if one of the items in the pool is defective, then the test for the pool will be positive. Tests can be organized to efficiently identify the defective items based on the outcomes of the tests. The Marco Polo problem differs from combinatorial group testing, however, in that the search space for the Marco Polo problem is a geometric region and tests must be connected geometric shapes (i.e., radius-d balls using the Euclidean metric), whereas the search space in combinatorial group testing is defined by a discrete set of n items and tests can be arbitrary subsets of these items.

There is also some work on SAR algorithms that use call-and-response protocols, such as the CenWits system by Huang, Amjad, and Mishra [14], which uses RF-based sensors for the search and rescue of people, such as hikers, who are carrying mobile wireless communication devices in wilderness areas.

In the context of computational geometry, the Marco Polo problem is somewhat related to the Freeze-Tag problem [1–4, 11, 17], which involves "waking up" a collection for moving robots that are initially at given points in the plane via a strategy motivated from the children's game, "Freeze Tag" [21].

We stress that we are interested in solutions to the Marco Polo problem that are adaptive, where the i-th probe can depend on the results of the probes that came earlier. A non-adaptive solution to the Marco Polo problem would be related to a constructive solution to a classic disk covering problem, which asks for the minimum number of disks of radius $\varepsilon > 0$ that can cover a region in the plane [15].

B Supplemental Pseudocode

Pseudocode bounding the distance traveled by the search point, Δ, is provided in Figure 9, and is implemented in https://github.com/ofekih/DroneSearching/blob/main/src/algorithm_utils.py.

Algorithm 1 Bounding the distance traveled by Δ

```
Require: A probe placement as a list of tuples $(x, y, \rho)$
Ensure: Upper bound on the total distance traveled
 1: $d_k \leftarrow 0$
 2: $p_{\text{curr}} \leftarrow (0, 0)$                         ▷ Current position
 3: $b_{\max} \leftarrow 0$                                 ▷ Maximum bound
 4: for each probe $(x, y, \rho_k)$ in placement do
 5:     $(c_x, c_y) \leftarrow p_{\text{curr}}$
 6:     $d_k \leftarrow d_k + \sqrt{(x - c_x)^2 + (y - c_y)^2}$
 7:     if probe is last in placement then
 8:         ▷ Skip round trip to final probe center
 9:         $(n_x, n_y) \leftarrow$ first probe in placement
10:         $d_1 \leftarrow \sqrt{n_x^2 + n_y^2}$
11:         $d_k \leftarrow d_k - 2d_1\rho_k$                 ▷ Subtract round trip
12:     $b_{\max} \leftarrow \max(b_{\max}, \frac{d_k}{1-\rho_k})$
13:     $p_{\text{curr}} \leftarrow (x, y)$
14: return $b_{\max}$
```

Figure 9: Algorithm for bounding the distance traveled by Δ. When the POIs are determined to be in the last probe area, Δ can move directly to the first probe of the next search area, saving distance. If the first probe is at distance d_1 from the origin of the original search area, it is at distance $d_1\rho_k$ from the origin of the new search area. The probe placement within the new search area can be rotated such that the first probe is as close to Δ as possible, saving an entire d_1-length round trip to the origin and back.[8]

C Omitted Proofs

In this appendix, we provide proofs that were omitted in the body of this paper.

[8]This assumes there is not significant overlap between the last two probes, which holds for all our algorithms.

Lemma 9 (Same as Lemma 1) *A correct progressively shrinking algorithm may require at most* $-\frac{1}{\log \rho_1}\log n + \mathbf{k}_{max} - \mathbf{1}$ *probes to find a POI in a search area of radius* n.

Proof. Consider for example the execution of our progressively shrinking Algorithm 6—where the first probe has a proportionality constant $\rho_1 = 0.812$, and, when ignoring constant additive factors as in Equation (3), finds a POI in at most $3.34 \log n$ probes. Algorithm 6 is able to fully cover the search area with its progressively shrinking probes, making it a *correct* progressively shrinking algorithm, and does so by subdividing the search area into 12 progressively shrinking probe regions, as shown in Figure 10.

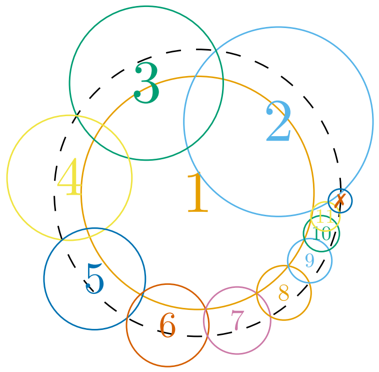


Figure 10: The execution of Algorithm 6 where the POI is at the last region probed by Δ on some layer.

Let us assume that, on the final layer, the POI is in the last search region, as shown in Figure 10, and let us consider three cases:

1. $n = 1$. In this case, we have trivially found the POI without requiring any probes. We also expected to spend at most $3.34 \log 1 = 0$ probes.
2. $n = 12.17$. In this case, we first perform 11 probes, and after they all return negative results, we determine that the POI is in the last region. The size of the last region is $12.17 \times \rho_1^{12} = 1.00$, successfully locating the POI. We spent 11 probes to find the POI, while we expected to spend at most $3.34 \log 12.17 = 12.04$ probes.
3. $n = 1.01$. Similar to the last case, we first perform 11 probes, however this time, the last region is of size $1.01 \times \rho_1^{12} = 0.08$. Not only did we locate the POI within distance 1 of it, we located it to within a much greater precision. Just like the previous case, we spent 11 probes to find the POI, while this time we expected to spend at most only $3.34 \log 1.01 = 0.05$ probes.

All three cases are able to successfully locate the POI, as expected from any correct algorithm. Moreover, the first two cases are able to locate the POI using at most the maximum number of probes expected by $P(n)$. Only the third case, which locates the POI to excess precision, requires more probes, specifically 11 more probes, than expected. Our result follows from the fact that this inefficiency can only occur on the last layer of the algorithm, and that the maximum number of probes we perform on each layer is $k_{\max} - 1$. □

If this added constant is of concern, it can be reduced by increasing the size of all probes to be at least 1. Such optimizations are outside the scope of this paper, however.

It is worth pointing out that Algorithms 1 and 2, which are not progressively shrinking, are able to express this added constant by using $\lceil \log n \rceil$ instead of $\log n$ in the bound for $P(n)$. This is because their value of $k_{\max} - 1$ exactly matches their $\log n$ coefficients of 6 and 5, respectively.

Lemma 10 (Same as Lemma 5) *A circle with radius* r *can be covered by an* L*-layer lattice of hexagons with side length* $s = \frac{2r}{3L-2}$.

Proof. Let us consider the case of a hexagonal lattice with an even number of layers, as in Figure 8. The closest point from the center of the lattice to its boundary is obtained by moving diagonally along the center hexagons maximal diameter. Consider the distance, r, to that point. Every even-numbered layer of hexagons contributes one side length s, while every odd numbered layer contributes its maximal diameter $2s$, except for the first layer which contributes s. Thus, the total circumradius $r = \frac{3L}{2}s - s = (\frac{3L}{2} - 1)s$.[9] □

Theorem 11 (Same as Theorem 8) *The total number of probes* (P_{tot}) *and the total distance traveled by* Δ (D_{tot}) *during the memoryless search algorithm for all* k *POIs is at most:*

$$P_{tot} \le c\lceil \log n \rceil + (c+1)(k-1)\lceil \log \bar{e} \rceil,$$
$$D_{tot} \le dn + 2dE,$$

where $E < OPT(\lceil \log k \rceil + 1)$, $\bar{e} = \frac{E}{k-1}$, *and* OPT *is the optimal tour length for the traveling salesperson problem (TSP) on the* k *POIs.*

Proof. We re-iterate the steps of our stateless below:

1. Find an arbitrary POI using $\mathcal{A}(n)$.
2. Shut off the tracking device of the found POI.
3. Without moving Δ, re-probe the area at radius 2, 4, 8, etc., until a probe returns a positive result (i.e., another POI is found).
4. Invoke $\mathcal{A}$ using this new radius to find another POI.
5. Repeat steps 2 to 4 until all POIs are found.

Other than step 1, the performance of the search strategy depends by the relative positions of the POIs. Let the first POI found be POI 0, and the second POI found in step 4 be POI 1. Let the distance between the two POIs be denoted as e_1, and the distance between POI $i-1$ and POI i be denoted as e_i, for $i \in \{1, 2, \ldots, k\}$. The total number of probes in step 3 required to find a large enough search area containing

[9]This bound is tight for lattices with an even number of layers, and only improves for lattices with an odd number of layers.

POI i is $\lceil \log e_i \rceil$, where the size of the search area is $2^{\lceil \log e_i \rceil}$. Finding the POI within this search area (step 4) will take $P(2^{\lceil \log e_i \rceil}) = c\lceil \log 2^{\lceil \log e_i \rceil} \rceil = c\lceil \log e_i \rceil$ probes. Adding on the initial probe to find POI 0, the total number of probes required to find all POIs is at most:

$$P_{\text{tot}} \le c\lceil \log n \rceil + \sum_{i=1}^{k-1} (c+1)\lceil \log e_i \rceil.$$

Let $E = \sum_{i=1}^{k-1} e_i$, and $\bar{e} = \frac{E}{k-1}$. At each step of the algorithm, we find a POI that is within a factor of two of the closest still-undiscovered POI to the last found POI. This algorithm will therefore perform at worst a factor of two approximation of the nearest neighbor tour for the traveling salesperson problem (TSP), e.g., see [5, 13]. In Euclidean space, the (greedy) nearest neighbor tour of n salespeople performs at worst a factor of $\frac{1}{2}(\lceil \log n \rceil + 1)$ of the optimal TSP tour length (OPT), see [18]. Applying to our case, $E <$ OPT$(\lceil \log k \rceil + 1)$. Since $\log e$ is a concave function, it follows from Jensen's inequality that $\sum_{i=1}^{k-1} \lceil \log e_i \rceil \le (k-1)\lceil \log \bar{e} \rceil$, where $\bar{e}$ is the average value of e, and we obtain our desired result.

Regarding the total distance traveled, note that Δ only moves during steps 1 and 4. For the first POI, Δ travels $D(n) \le dn$. Using similar reasoning as above, for the i-th POI, Δ travels $D(2^{\lceil \log e_i \rceil}) \le d(2e_i)$. Overall, the total distance traveled is at most: $dn + 2d\sum_{i=1}^{k-1} e_i = dn + 2dE$, and our result follows. $\square$

D Algorithm Assumptions

In this section we justify some assumptions regarding the possible probes and locations of the POIs that we make in the main body of the paper. Namely, we assume that:

- The maximum allowed probe distance, d, is n.
- There may be multiple POIs, either within the initial search region or slightly outside of it.

More specifically, we show how, by relaxing these assumptions, a relatively simple solution is able to solve the problem using an optimal number of probes.

Simple Case: Exactly One POI at Full Distance. Let us begin with a simple (but admittedly unrealistic) scenario, where there is exactly one POI within distance n of the origin and we can perform arbitrarily large probes. In this case, where we allow unbounded probe distances, we can find the sole POI using two binary searches, one in each dimension. We place Δ at a very large distance from the origin, such that the intersection for the probe with the original search region is nearly a straight line, and perform one binary search using $\lceil \log n \rceil + 1$ probes to find a region of width at most 1 in the first dimension. We then perform a similar binary search placing Δ in an orthogonal direction to find the POI in the second dimension. We are left with a region of side length 1, which is fully contained within a single, final, radius-1 probe. This solution thus uses $2\lceil \log n \rceil + \mathcal{O}(1)$ probes, but it relies on Δ traveling very far away and at high altitude and the tracking device to have very strong signal strength, which are not reasonable assumptions. This can be improved somewhat by restricting the probe distance to n, as we explore in the next section.

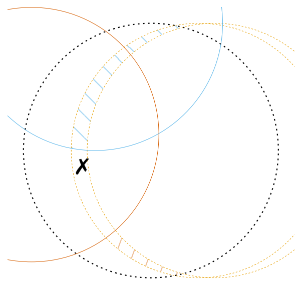


Figure 11: A search for one POI (✗), first reducing the horizontal dimension to a width of 1 (in the dashed orange probes), then searching along the remaining arc (with the solid blue and orange probes).

A Slight Improvement to a Bad Algorithm. While the previous algorithm allows for unbounded probe distances, we now show how to achieve a similar result while restricting the probe size to be at most n. In this case, we reduce the horizontal dimension this time first to a 1-width arc, and then cut the length of the arc in half for each subsequent probe. Note that the arc may be up to πn units long, which is greater than the $2n$ original region diameter, resulting in up to one more probe. This probe can be removed by reducing the remaining *area* by two per probe in the first dimension rather than the arc width. See Figure 11.

Note that this algorithm may require Δ to travel outside of our original search region. This can be avoided with a final algorithm. We can place Δ at the center of the original search region and perform a binary search by changing the radius of the probes, resulting in at most a width-1 shell of the original search region, requiring at most $\lceil \log n \rceil$ probes.[10] This shell of outer-radius r is $2\pi r$ units long, where r can be up to n. We can reduce this shell using a similar binary search, by moving Δ along the outer edge of the shell, performing probes with radius r. See Figure 12 for an example of this algorithm. Note that the first of these probes will only reduce one third of the shell, rather than a factor of 2, resulting in up to one more probe overall. We can then perform probes to reduce this shell to a width-1 square, which could be performed in $\lceil \log n \rceil + 2$ probes.

E Darting Non-Monotonic Algorithms

In this section we discuss the darting algorithms, which are non-monotonic and place many more probes at each recursive level by first placing several carefully-placed probes and then greedily adding more to fill in smaller and smaller gaps, albeit at the expense of darting from side to side in the search space to do so. That is, the approach for our darting algorithms is as follows: after an initial placement of several probes, we determine if there are any uncovered internal areas. If so, we find the largest such area and place a probe

[10]Note that reducing the remaining *area* by a factor of two at this stage would reduce a constant number of probes.

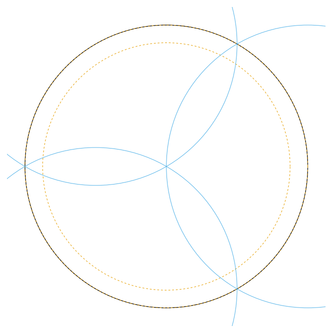


Figure 12: A depiction of the final "simple" search strategy where it is known that there is only one POI within the search region. In this final strategy, not only is the probe distance limited to n, but Δ is also restricted to the original search region.

such that it intersects two points on the area's convex hull. If there are multiple possible placements, we repeatedly choose the one that reduces the remaining area the most in a greedy fashion. See Figure 13. Subsequent probes are placed in the same manner, until either the entire search area is covered, or the probes become sufficiently small as to be unable to cover the remaining area.

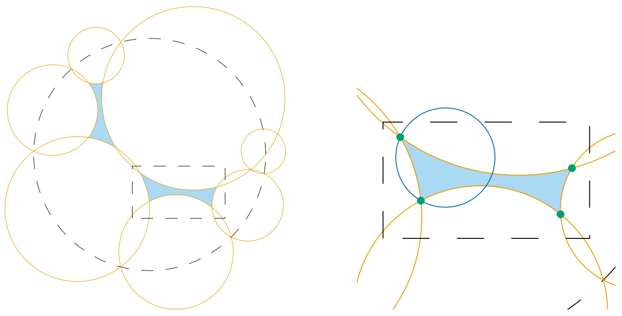


Figure 13: A method to greedily add an arbitrary amount of probes to any initial placement of probes. On the left, Algorithm 4 given an insufficient ρ_1 value fails to cover the search area, leaving some uncovered internal area (in blue). On the right, the largest uncovered internal area's convex hull (in green) is identified, and a probe is placed such that it intersects two points on the hull.

We note that the final probe of Algorithm 4 is often the least efficient; see Figure 13 (left) and Figure 4. If we run Algorithm 4, remove the final probe, and then apply the greedy method described above to place the remaining probes, we obtain Algorithm 7, which is able to significantly reduce the number of probes required to find a POI to $P(n) < 2.93 \log n$.

Exploiting this approach further, however, strains our ability to reason about regions that are uncovered after performing many probes; hence, for an even further improved algorithm, Algorithm 8, its probe sequence for each recursive level is determined using computer-assisted proof. More specifically, the placement of the first six probes is determined by a differential evolution algorithm, which is a type of genetic algorithm that optimizes a function by iteratively improving a population of candidate solutions, see [19]. Next, we fill in the gaps according to the aforementioned greedy method. The resulting Algorithm 8 is able to achieve our best probe results, with $P(n) < 2.53 \log n$, albeit with a very large value for $D(n)$. See Figure 7.

F Experiments

We implemented our 8 algorithms and tested them. The data were obtained by placing a POI at a random location, determined from a uniformly random angle and a random distance from the center of the search area. When the POI was in the last probe's search area, the last probe was not executed, and Δ proceeded directly to the first probe of the next search area. Each algorithm was executed 40 million times, where $n = 2^{20}$, and normalized by dividing by either $\lceil \log n \rceil$ or by n. Our code is publicly available at https://github.com/ofekih/DroneSearching. Both Algorithms 1 and 2, despite having a poor worst-case probe complexity, perform well in practice, finding the POI using fewer probes on average than Algorithms 3–6. Our computer-assisted algorithms (7 and 8), however, outperform them. Interestingly, the progressively shrinking algorithms all have a non-zero variability in their probe counts. See Figure 14. While Algorithms 1 and 2 perform well in terms of average distance traveled, they are outperformed by Algorithms 5 and 6, which each have the best distance-traveled guarantees. See Figure 15. Overall, the best methods are Algorithm 8 if number of probes is a priority, Algorithm 6 if distance traveled is a priority, and Algorithm 2 for a good balance.

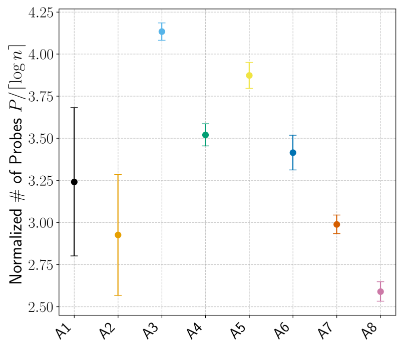


Figure 14: Simulation results for $P/\lceil \log n \rceil$. Error bars represent one standard deviation from the mean.

Number of Probes Made. We make two interesting observations regarding the number of probes made, P, for our progressive probe algorithms.

Observation 1 *The progressive probe algorithms, Algorithms 3–8, exhibit a non-zero variance in P experimentally.*

And perhaps more surprisingly, in Table 1 we observe:

Observation 2 *Progressive probe algorithms appear to perform more probes than their theoretical upper bounds.*

		Probes $(P/\log n)$				Total Distance (D/n)				Responses $(R/\log n)$		
Category	Alg. #	Min	Avg	Max	Bound	Min	Avg	Max	Bound	Avg	Max	Bound
Hexagonal	Alg. 1	**1.00**	3.24	5.70	6.00	**0.00**	3.35	10.39	10.39	**0.89**	**1.00**	**1.00**
	Alg. 2	**1.00**	2.93	4.80	5.00	**0.00**	2.65	8.81	8.81	1.11	1.45	2.00
Chord-Based	Alg. 3	3.85	4.13	4.25	4.08	4.69	5.46	6.56	6.95	1.99	2.40	4.08
	Alg. 4	3.10	3.52	3.70	3.54	4.30	5.38	9.00	9.31	1.94	2.50	3.54
Monotonic	Alg. 5	3.55	3.87	4.15	3.83	**0.00**	**1.92**	6.72	6.72	2.49	3.85	3.83
	Alg. 6	3.25	3.41	3.85	3.34	**0.00**	1.96	**6.01**	**6.02**	1.96	3.35	3.34
Darting	Alg. 7	2.90	2.99	3.65	2.93	3.86	5.97	25.74	25.80	1.39	2.15	2.93
	Alg. 8	2.55	**2.59**	**3.20**	**2.53**	2.44	4.05	42.58	45.40	1.31	1.85	2.53

Table 1: A numerical comparison of simulation results for our 8 algorithms on three normalized performance metrics, namely the number of probes made (P), the total distance traveled by Δ (D), and the number of POI responses (R). The best values are highlighted in bold. The category names used are crude abbreviations; see the main paper for their proper names.

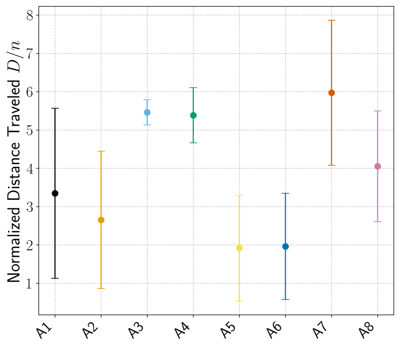


Figure 15: Simulation results for D/n, where $n = 2^{20}$.

		Probes $(P/\log n)$	
Category	Alg. #	Max	True Bound
Hexagonal	Alg. 1	5.70	6.00
	Alg. 2	4.80	5.00
Chord-Based	Alg. 3	4.25	4.28
	Alg. 4	3.70	3.9
Monotonic	Alg. 5	4.15	4.18
	Alg. 6	3.85	3.89
Darting	Alg. 7	3.65	4.18
	Alg. 8	3.20	4.13

Table 2: A table comparing the maximum number of probes observed in our experiments with the true upper bounds for each algorithm when $n = 2^{20}$.

Both observations are primarily explained by the last recursive layer in each of our algorithms. In the proof of Lemma 1 in Section C, we show how the number of probes required in the last layer depends significantly on the size of the last layer as well as the placement of the POI within it. Specifically, it can vary by up to the maximum number of probes performed per layer of the algorithm, $k_{\max} - 1$, and thus the true maximum number of probes performed can be up to $k_{\max} - 1$ more than our simplified bounds as shown in Table 1 (see Lemma 1). To obtain more precise upper bounds for any specific n, we can expect our normalized number of probes made, $P(n)/\log n$, to be increased by up to $\frac{k_{\max}-1}{\log n}$. As expected, these adjusted bounds now hold for all algorithms. See Table 2. As n increases, the $\log n$ coefficient in $P(n)$ dominates this constant number of extra probes, so our theoretical upper bounds will hold.

Total Distance Traveled by the Search Point. As expected, both our higher-count monotonic-path (HM) algorithms, Algorithms 5 and 6, minimize the total distance traveled by Δ. Algorithms 1–3 also have monotonic counter-clockwise paths, but their less efficient probe sequences result in worse average distance traveled. See Table 1. The hexagonal and HM algorithms each start with a central probe,[11] and consequently have a minimum of 0 distance traveled. While all algorithms experienced an instance where Δ traveled nearly as much as their worst-case bound, reassuringly, most algorithms performed significantly better on average, with the hexagonal and HM algorithms performing ~3 times better, and the darting non-monotonic-path algorithms, Algorithms 7 and 8, performing ~4 and ~11 times better, respectively. They are still likely not best suited for time-critical rescue operations since they travel 2-3 times more than the HM algorithms on average.

Number of POI Responses. Finally, we compare the number of POI responses, R, for our algorithms, to see which are best suited for the case where the POI has battery constraints. We find that, Algorithm 1 performs by far the best, with not only the fewest responses across the board, with its maximum number of responses being lower than the average of any other algorithm, but also with the smallest standard deviation. See Figure 16. This result is not surprising, because in Corollary 7 (3), we learn that Algorithm 1 is part of a larger family of response-efficient algorithms. The next best algorithm is Algorithm 2, also partly based on a hexagonal grid, and then the darting algorithms, Algorithms 7 and 8, which perform only a small number of probes in general.

[11]Recall that the hexagonal algorithms can be modified to start with the first probe instead of ending with it.

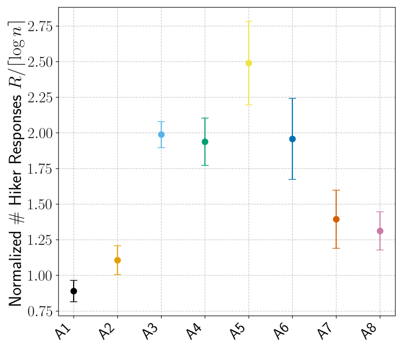


Figure 16: Simulation results for $R/\log n$

The Rectilinear Marco Polo Problem*

Ofek Gila[†] Michael T. Goodrich[†] Zahra Hadizadeh[‡] Daniel S. Hirschberg[†] Shayan Taherijam[†]

Abstract

We study the ***rectilinear Marco Polo problem***, which generalizes the Euclidean version of the Marco Polo problem for performing geometric localization to rectilinear search environments, such as in geometries motivated from urban settings, and to higher dimensions. In the rectilinear Marco Polo problem, there is at least one ***point of interest*** (POI) within distance n, in either the L_1 or L_∞ metric, from the origin. Motivated from a search-and-rescue application, our goal is to move a ***search point***, Δ, from the origin to a location within distance 1 of a POI. We periodically issue ***probes*** from Δ out a given distance (in either the L_1 or L_∞ metric) and if a POI is within the specified distance of Δ, then we learn this (but no other location information). Optimization goals are to minimize the number of probes and the distance traveled by Δ. We describe a number of efficient search strategies for rectilinear Marco Polo problems and we analyze each one in terms of the size, n, of the search domain, as defined by the maximum distance to a POI.

1 Introduction

Gila, Goodrich, Hadizadeh, Hirschberg, and Taherijam [13] introduce the Marco Polo problem, which they motivate in terms of one or more points of interest (POIs), thought of as hikers lost in a forest, that we would like to localize using a mobile search point, Δ. Each lost hiker is assumed to have a wireless device that can respond to probes sent from Δ to a specified distance such that if a lost hiker is within that distance of Δ, then the search algorithm will receive a positive response. Such probes use up power, of course, both for Δ and for a POI's tracking device; hence, the goal is to devise a search algorithm for Δ and a sequence of probes that minimizes the number of probes needed to locate a POI to within a distance of 1.

In the formulation of Gila *et al.* [13], the underlying geometry for the Marco Polo problem is Euclidean, such that Δ can move unrestrictedly in any direction and probes are circles, which seems reasonable for searching in a forest but not for searching in an urban environment where distance is more accurately abstracted as being rectilinear. In this paper, we are interested in studying a rectilinear version of the Marco Polo problem.

As a colorful motivation of the two-dimensional version of the rectilinear Marco Polo problem, suppose one or more people have been kidnapped and are being held in one or more secret points of interest (POIs) in a city (like New York, Chicago, or Toronto) whose streets are essentially grids. A mobile search point, Δ, can move to search for them that is restricted to flying or driving along rectilinear paths (since it cannot fly or drive through buildings). Each kidnap victim at a POI has a hidden electronic device that can respond to probes from Δ, which can issue probes to specified rectilinear distances such that if there is a kidnap victim within this distance, then our search algorithm will learn this. But the search algorithm does not learn the direction or distance to the kidnap victim. The optimization problem is to minimize the number of probes and/or victim responses, as well as possibly minimizing the travel distance for Δ. We are therefore interested in efficient searching strategies for rectilinear Marco Polo problems with analyses in terms of the size, n, of the search domain.

We can therefore formulate the rectilinear Marco Polo problem as a computational geometry problem, where we have at least one ***point of interest*** (POI) at distance at most n from the origin, and we want to move a ***search point***, Δ, to within distance 1 of a POI, guided by probes. A probe is a query specified by Δ's position and a distance, d, such that we learn whether or not a POI is within distance d from Δ, in either the L_1 or L_∞ metric. Our optimization goals are to minimize the number of probes and distance for Δ to travel to find a POI.

Related Prior Work. We are not familiar with any prior work on the rectilinear Marco Polo problem. As mentioned above, Gila, Goodrich, Hadizadeh, Hirschberg, and Taherijam [13] introduce the Euclidean version of the Marco Polo problem, where search paths are not restricted and travel distances and probe distances are measured with the Euclidean L_2 metric. For example, they provide a number of carefully choreographed travel patterns and probe strategies, including one that finds a POI with $3.34\lceil\log n\rceil$ probes and flight distance $6.02n$ and a strategy that uses

*This research was supported in part by NSF grant 2212129.

[†]University of California, Irvine, {ogila,goodrich,dhirschb,staherij}@uci.edu

[‡]University of Rochester, zhadizadeh99@gmail.com

$2.53\lceil \log n \rceil$ probes and flight distance $45.4n$.

The Marco Polo problem is related to combinatorial group testing, see, e.g., [8–11,14]. In this problem, one is given a set of n items, at most d of which are "defective." Subsets of the items can be pooled and tested as a group, such that if one of the items in the pool is defective, then the test for the pool will be positive. Tests can be organized either adaptively or non-adaptively to efficiently identify the defective items based on the outcomes of the tests. The Marco Polo problem differs from combinatorial group testing, however, in that the search space for the Marco Polo problem is a geometric region and tests must be connected geometric shapes (like squares), whereas the search space in combinatorial group testing is for a discrete set of n items and tests can be arbitrary subsets.

Another related problem in computational geometry is the freeze tag problem [2, 3, 7, 15], which has also been studied in the rectilinear setting [6,19], where one is interested in moving robot points in the plane to "wake up" a collection of robots. Also, another related rectilinear combinatorial optimization problem is the optimization problem abstracted from the Minesweeper game; see, e.g., [17]. There is also considerable prior work on search-and-rescue algorithms focused on non-combinatorial solutions, including the use of continuous monitoring, sophisticated cameras, and non-adaptive travel patterns; see, e.g., [1,4,5,16,18,20–22].

Problem definition. In the ***rectilinear Marco Polo*** problem, there are $k \geq 1$ entities, which we'll call ***points of interest*** (POIs), with unknown positions, at least one of which are within a distance, n, in the L_1 or L_∞ distance metric, of a point, O, called the ***origin***. That is, the search region is a diamond or square in $\mathbb{R}^2$ or a octahedron or cube in $\mathbb{R}^3$.

A ***probe***, $p(x, y, d)$, is a query that asks if there is any POI within distance d of the current position, (x, y), of a search point, Δ, measured under a distance metric, since such a point is the position at which, e.g., a search algorithm would issue a probe request to a wireless device of a lost kidnap victim. The goal is to design a search strategy for Δ to localize one or more POIs to within a distance of 1. In this paper, we primarily consider the case where Δ only searches for a single POI, which can be combined with an ***incremental*** search strategy which finds POIs one at a time, for example, using the generalized algorithm of [13]. We also make no assumptions about the number of POIs, k, and their locations, besides the fact that at least one POI is within distance n from the origin, referred to the ***unbounded*** version of the problem. Finally and most importantly, unlike the paper by Gila *et al.* [13], which focuses solely on the L_2 distance metric, we consider the rectilinear metrics, L_1 and L_∞.

For any search strategy, there are a number of ways we can measure the effectiveness of the strategy, including:

- $P(n)$: the number of probes issued by Δ.
- $R_{\max}$: the maximum number of times a POI must respond to a probe.
- $D(n)$: the total distance traveled by Δ, in a chosen rectilinear metric, such as L_1 or L_∞.

Our Results. In this paper, we provide a number of efficient algorithms for the solving Marco Polo problems, with algorithms that achieve optimal or near-optimal performance across all measures. We begin with a warm-up algorithm which sequentially checks each quadrant of the search area in the 2D case, and each octant in the 3D case. While this simple algorithm indeed has a poor worst-case probe performance, we show that it effectively minimizes the number of responses required by each POI to just $\lceil \log n \rceil$, regardless of the number of dimensions. We then present a pair of more sophisticated algorithms that use a domino-like recursive search pattern to achieve great probe complexities, to just $2\lceil \log n \rceil + 1$ in 2D and $3\lceil \log n \rceil + 3$ in 3D, near-optimal with respect to the lower bounds of $2\lceil \log n \rceil$ and $3\lceil \log n \rceil$, for 2D and 3D, respectively.

We then focus on minimizing the distance traveled by Δ with respect to the distance to the nearest POI, $\delta_{\min}$, presenting an algorithm which performs a binary search for each dimension, which we call ***central binary search*** (CBS). A search point, Δ, following this algorithm will travel a distance of at most $2\delta_{\min} + O(1)$ in 2D, $3\delta_{\min} + O(1)$ in 3D, while still maintaining a near-optimal probe complexity. Afterwards, we show how to extend our algorithms to higher dimensions, achieving the orthant algorithm that yields good POI response performance for all dimensions, and achieving the generalized CBS algorithm, which provides near-optimal probe complexity and instance-optimal distance performance. Finally, we present a method to make the probe and response performance of any algorithm instance-optimal with respect to $\delta_{\min}$. In Section B, we include experiments that support our results.

For simplicity, we primarily focus on instances of the rectilinear Marco Polo problem in the L_∞ metric in this paper, representing the search area as a square or cube rather than as a diamond or octahedron in our figures. Likewise, we refer to the search area as a hypercube in general case rather than as a cross-polytope. Nevertheless, our results apply equally well to the L_1 metric except where explicitly specified.

2 Rectilinear Searching Strategies

We first introduce a number of algorithms for the 2D and 3D rectilinear searching problem, including more natural algorithms and some more involved ones, and show how each are optimal with respect to a different metric. In general, we assume that there may be multiple POIs, yet we are only interested in finding one of them, and that Δ starts its search from the origin. We do note, however, that each of our algorithms can be used as a subroutine in the general method of finding all targets as described in [13]. We generalize several algorithms to higher dimensions in a later section.

Quadrant and Octant Algorithms. Perhaps the first algorithms that come to mind are those that divide the original search area into quadrants and octants. We probe each quadrant from its center, with a probe of half the radius of the parent region. Since a POI is known to exist within the search area, at most three quadrants need to be probed. And each subsequent ***layer*** has half the radius of the previous. As such, there are at most $\lceil \log n \rceil$ layers total, and since each layer may take up to three probes, the total number of probes, $P(n)$, is at most $3\lceil \log n \rceil$ probes. In the 3D case, there are eight octants in total, where at most seven must be probed, resulting in $P(n) < 7\lceil \log n \rceil$ probes. See Figure 1.

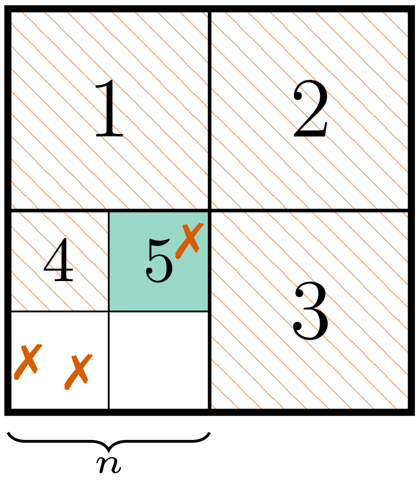


Figure 1: A simple quadrant search algorithm, showing the first 5 probes. Only three of the four quadrants must be probed, as the POIs (denoted by ✗) must be in the final quadrant if the first three probes fail. Regions with diagonal lines correspond to failed probes, while probe 5 is a successful probe. The search will continue in the 5th region. For simplicity, figures represent L_∞ probes, but our algorithms and figures translate directly to L_1 probes when considered diagonally.

While our subsequent algorithms will reduce the total number of probes, this algorithm behaves best with regards to the maximum number of times a POI must respond to a probe, $R_{\max}$. Specifically, since a POI only responds at most once per layer for both 2D and 3D, we have that $R_{\max} \leq \lceil \log n \rceil$ responses. We note the similarity between this algorithm and the hexagonal algorithms of [13].

Domino Algorithms. Our quadrant algorithm was able to find a POI using at most $3\lceil \log n \rceil$ probes, and the question remains—can we do better? For the 2D case, since we start with an area of $(2n)^2$ and end with an area of no larger than $(2)^2$, and each probe, in the worst case, at most halves the remaining area, there is a trivial lower bound of $2\lceil \log n \rceil$ probes. Similar reasoning can be used to lower bound the worst case number of probes in the 3D case to at least $3\lceil \log n \rceil$ probes. In this section, we introduce our first two algorithms, which are able to achieve within constant factors of these lower bounds; due to their structure, we refer to these as the 2D and 3D domino algorithms, respectively.

The 2D Domino Algorithm. In the 2D domino algorithm, we refer to a position as a 2-domino if it consists of two equally sized areas—a $d \times d$ area where a POI is known to exist, and an adjacent $d \times d$ area known to be empty, i.e., where *no* POI exists. Let the empty region be to the left of the remaining search area as depicted in Figure 2, without loss of generality. Our first probe has radius $d/2$, and is placed halfway between the two areas. If the probe fails, we know that the POI must be in the remaining right half of the search area. If the probe succeeds, however, we take advantage of the fact that we know that the left half of the probe is empty, and similarly reduce the search area. Regardless of the result, we perform a second probe with radius $d/4$ in one half of the remaining search area, and achieve a new 2-domino with a quarter of the area. This is optimal, since the remaining area is halved with each probe. Figure 2 depicts this procedure.

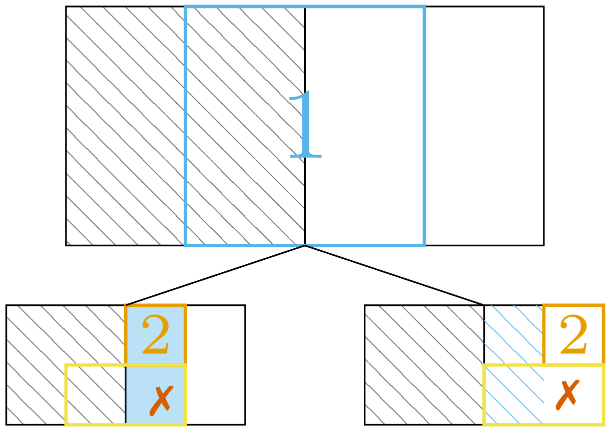


Figure 2: The recursive 2-domino procedure as used in the 2D domino algorithm. Each probe reduces the remaining area by a factor of 2. Regardless of the results of these two probes, we are left with a new 2-domino where each dimension is halved, depicted in yellow.

The question that remains, however, is how to achieve the initial 2-domino. To this end, our 2D domino algorithm performs the top layer using the simple quadrant algorithm depicted in Figure 1. In the best case, if the first probe succeeds, we have reduced the area by a factor of 4 and simply continue our algorithm recursively into this quadrant. Otherwise, if the first probe fails, any subsequent probe that succeeds is adjacent to a quadrant that is known to be empty, and we can initiate our 2-domino procedure. In the worst case, it will take all 3 initial probes to reach a domino with a quarter of the remaining search area, resulting in $P(n) \leq 2\lceil \log n \rceil + 1$ probes.

The 3D Domino Algorithm. Unlike for our quadrant algorithm, it is not as straightforward to extend our 2D domino algorithm to a 3D algorithm, and we only do so under the L_∞ metric. Our first step is to define a 4-domino as a 3D region consisting of four equally sized $d \times d \times d$ cubes, where a POI is known to exist in one, and all the rest are known to be empty. As before, the first probe has radius $d/2$, and is placed halfway between the search area and one of the two adjacent empty regions. The second probe also has the same radius, and is placed orthogonally depending on the result of the first probe as to halve the remaining volume again. Finally, we perform the final probe of radius $d/4$ in a half of the remaining search area, resulting in a new 4-domino with an eighth of the original volume. See Figure 3.

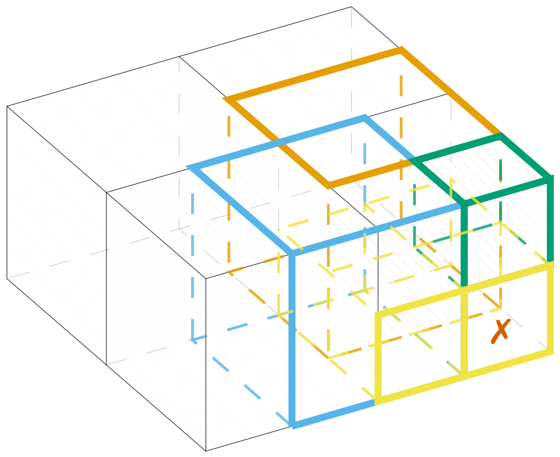


Figure 3: The recursive 4-domino procedure as used in the 3D domino algorithm. Each probe reduces the remaining volume by a factor of 2. Regardless of the results of these three probes, depicted in blue, orange, and green, respectively, we are left with a new 4-domino where each dimension is halved, depicted in yellow.

We have shown that once we reach a 4-domino shape, we are able to perform optimally, reducing the remaining volume by a factor of 2 with every probe. Unfortunately, there is no simple procedure by which we efficiently reach a 4-domino shape from the original search area. In Section A, we show how by using an additional construction, similar to the 2D 2-domino procedure, as an intermediate step, we are able to prove good results. The resulting 3D domino algorithm requires at most $3\lceil \log n \rceil + 4$ probes in the worst case.

The Central Binary Search Algorithm. Our domino algorithms were able to achieve excellent probe complexities of $2\lceil \log n \rceil + \mathcal{O}(1)$ in 2D and $3\lceil \log n \rceil + \mathcal{O}(1)$ in 3D. These algorithms, however, make no attempt to minimize the distance traveled by the search point, Δ, which may be important in practice in a real-world scenario. In this section we discuss a 2D algorithm that not only minimizes the number of probes, $P(n)$, but also minimizes the distance traveled by Δ, $D(n)$. More specifically, we will show how our algorithm is instance-optimal with respect to distance, which we define as having Δ travel a distance of at most $D(n) \in \mathcal{O}(\delta_{\min})$, where $\delta_{\min}$ is the distance of the closest POI to the origin using either the L_1 or L_∞ metrics. This algorithm performs two binary searches from the center of each dimension, so we refer to it as the 2D **central binary search** algorithm.

Our algorithm can be thought of in two distinct 'phases', one for a binary search in 2D, and another for a binary search in a '1D' edge, as depicted in Figures 4 and 5, respectively. In a later section, we will show how this algorithm can be generalized not only to three dimensions, but also beyond.

For the first phase, we perform a sequence of probes from the origin, without moving Δ, performing a binary search to find a width-1 shell containing the nearest POI, as described in Algorithm 1. This binary search takes at most $\lceil \log n \rceil$ probes. Afterwards, we determine which one of the four edges of the shell contains a POI, which can be performed with two additional probes, as depicted in Figure 4. While the probe does not need to move by much to perform these two probes, it may need to move 1 or 2 units along the x and y axes, such as to only probe a desired subset of the shell's edges. We go into more detail about this in a later section.

Algorithm 1 Binary Search for the Initial Shell Radius

```
1: Output: approximate distance $\tilde{\delta}$ to nearest POI,
   such that $\tilde{\delta} - 1 \leq \delta_{\min} \leq \tilde{\delta}$.
2: $l \leftarrow 0$, $h \leftarrow n$                ▷ lower and upper bounds
3: while $h - l > 1$ do
4:     $m \leftarrow \left\lfloor \frac{h+l}{2} \right\rfloor$
5:     if $p(0, 0, m)$ succeeds then
6:         $h \leftarrow m$                      ▷ POI is in the shell
7:     else
8:         $l \leftarrow m$                      ▷ POI is outside the shell
9: return $h$
```

After this, we have reduced the problem to a 1D search along an edge of the original shell. A similar binary search is performed to find the two width-1 squares nearest to the center of the edge which are

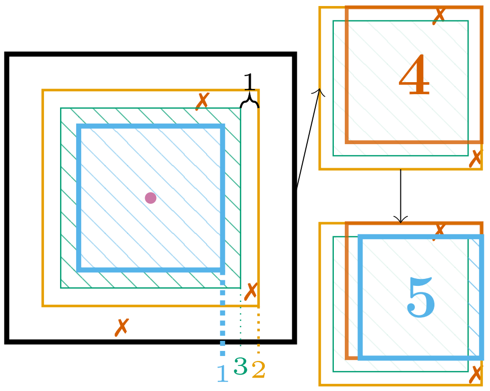


Figure 4: The 2D central binary search algorithm. The probe is always centered at the origin, depicted by a purple point, but varies in size, increasing after every failed probe, and decreasing after every successful probe. It continues performing this binary search until reaching the closest width-1 shell to the origin where a POI is known to exist. After this, only two additional probes are needed to determine which of the four edges of the shell contains a POI.

known to contain a POI. While this binary search also takes at most $\lceil \log n \rceil$ probes, it may require Δ to move back and forth along the line from the origin and the center of the edge. See Figure 5. The first probe is performed at the center of this line, and requires the probe to move $\tilde{\delta}/2$ units from the origin under both the L_1 and L_∞ metrics. The second probe will be performed $\tilde{\delta}/4$ units away, either back towards the origin in the case of a failed probe, or towards the edge in the case of a successful probe. In fact, each subsequent probe will require moving Δ half the distance as the previous, and so the total distance traveled by Δ to perform this search is at most $\tilde{\delta}$. One final probe then determines which of the two width-1 squares contains a POI, which again requires moving Δ only a small constant distance. And finally, in order to reach the location of the POI, Δ must move a final distance of at most $\tilde{\delta}$.

Our algorithm may require up to $2\lceil \log n \rceil + 3$ probes to find a width-1 square containing a POI, but recall that our objective is to reach a position within a *distance* of 1 of our POI. In truth, we can loosen our requirements to only finding a width-*2* square containing a POI, such that its center is at most 1 unit away from the edges. Each binary search will then take one fewer probe, resulting in a total of $2\lceil \log n \rceil + 1$ probes. And as desired, the total distance traveled by Δ is at most $2\delta_{\min} + \mathcal{O}(1)$, where $\delta_{\min}$ is the distance to the nearest POI.

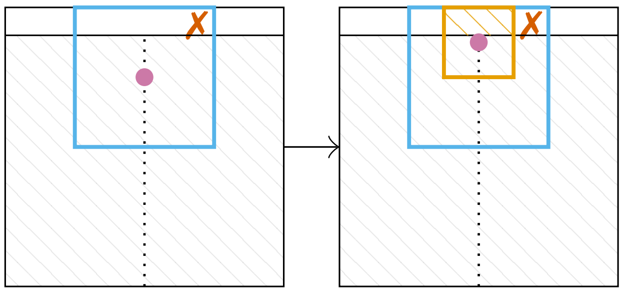


Figure 5: The second phase of the 2D CBS algorithm, where a POI is known to exist on a width-1 edge. One more binary search is conducted, where the probe moves along a line perpendicular to the center of the edge. After this, one final probe is necessary to determine which of the two 1-by-1 regions contains a POI.

3 Extended Search Strategies

In this section, we expand on the key results of the previous section and show how to generalize them to higher dimensions.

Orthant Algorithm. As a warmup, we first extend our quadrant and octant algorithms to operate in higher dimensions. In this context, an **orthant** is the higher-dimensional analogue of a quadrant (in 2D) or octant (in 3D), where each k-dimensional hypercube has 2^k orthants. By searching at most $2^k - 1$ orthants per layer, we can guarantee that we will find a POI within at most $(2^k - 1)\lceil \log n \rceil$ probes, where k is the dimension of our search space, while receiving at most $\lceil \log n \rceil$ responses. To reduce the total distance traveled, we always probe adjacent orthants, which can be done by following a Gray code, sometimes known as a single-digit code [12]. The distance between the centers of two adjacent orthants in the original hypercube is n under both the L_1 and L_∞ metrics, so in the first layer alone we may travel a distance of over $2^k n$. Certainly such an algorithm should not be used when distance traveled is of any concern.

Generalized Central Binary Search (CBS) Algorithm. In this section we extend our 2D CBS algorithm to k dimensions under the L_∞ metric. Consider our true objective of a search strategy: to find a point in space that is within distance 1 of a POI. In k-dimensional space, this means a point with coordinates $x_1, \ldots, x_k$, where each coordinate is within distance 1 of the corresponding coordinate of a POI. As with the 2D case, we will split our algorithm into separate phases, where in each phase we will fix the coordinates of at least one of the dimensions, thus effectively reducing the dimension of our search space by 1. Let phase p refer to the phase in which our search space is effectively confined to a p-dimensional subspace. After at most k phases, we will have fixed all k coordinates, and therefore we will have found a POI. As in the 2D case, each phase p starts by a binary search for the (approximate) radius, $\tilde{\delta_p}$, of the smallest cube that contains at least one POI. Recall that each binary search takes at most $\lceil \log n \rceil$ probes. While Δ is stationary in our p-dimensional subspace during each binary search, for any phase $p < k$, Δ may need to move up to $\tilde{\delta_p} < \delta_{\min} + 1$ units overall (in the original k-dimensional space) to conduct the search. Regarding distance traveled, during the first phase, when $p = k$, Δ can remain at the origin, while for all subsequent phases, Δ may need to move up to $\tilde{\delta_p} < \delta_{\min} + 1$ units. This movement is necessary since although we visualize the search as oc After this search, we are guaranteed that at least one POI contains a coordinate that is within distance 1 of $\pm\tilde{\delta_p}$.

In p dimensions, after the binary search determines our radius $\tilde{\delta_p}$, we consider a p-cube with radius $\tilde{\delta_p}$, where we know that the POI is located within distance 1 of one of the cubes' facets. In our 2D algorithm, we were able to determine not only which coordinate to set (the x or the y), but also which sign to set ($\pm$) using only two probes. The first probe eliminated two of the four edges, and the second probe eliminated one of the two remaining edges. See Figure 4 (right). Intuitively, it may seem possible to extend this idea to our p-cube, halving the number of facets we consider with each probe. Doing so would allow us determine the facet in $\log(2p)$ probes, but this is unfortunately not possible in general.

One way to build this intuition is to consider the corners of our p-cube. The distance between any two corners is $2\tilde{\delta_p}$, so any probe that tests for the presence of a POI in two corners concurrently must have a radius of at least $\tilde{\delta_p}$. However, such a probe, when initiated from the center of the cube, will encompass it entirely and not providing any new information, and when initiated from any other point, will include regions outside of it which may contain other POIs. Therefore, in the case of multiple possible POIs, we cannot probe multiple corners of the cube concurrently. This is unfortunate, since a p-cube has 2^p corners, and since each of its facets contains 2^{p-1} corners, if the POI is located within distance 1 of a corner, we may need to probe 2^{p-1} corners before determining which facet contains the POI. With this unfortunate observation in mind, not only do we need to individually probe each facet, but our probes must be smaller than $\tilde{\delta_p}$, such that each probe may not include the (lower dimensional) boundary of the facet. In the case of 3D cubes for example, each probe may not cover the edges nor corners of the face being probed. Therefore, we resort to not only individually probing each of the $2p$ facets, but also the lower-dimensional faces, as described in Algorithm 2.

The only remaining detail is the `ProbeFace(f)` operation for an arbitrary $(p - a)$-dimensional face f.

Algorithm 2 Finding Cube Face near POI

```
1: Output: highest-dimensional $p$-cube face $f$ near at least one POI.
2: for $a = 1, \dots, p$ do
3:     for $f \in (p - a)$-dimensional cube faces do
4:         if ProbeFace($f$) then
5:             return $f$                    ▷ POI is near face
```

Lemma 1 *Consider a $(p - a)$-dimensional face f of a p-cube of radius $\tilde{\delta_p}$ centered at the origin. f can be defined by a unique vector $s \in \{-1, 0, 1\}^p$ of length p, where 'a' values are fixed to ± 1, and the remaining values are 0, such that any point x of f satisfies $s \cdot x = a\tilde{\delta_p}$. `ProbeFace(f)` can be performed by moving Δ to coordinate s and conducting a probe with radius $\tilde{\delta_p} - 1$.*

Using this lemma, which we prove in Section A, we are able to probe each face by moving Δ at most one unit from the origin under the L_∞ metric (since the maximum absolute value of s is 1). Since the number of faces required to probe is independent of n, we obtain that the total number of probes required is at most

$$k\lceil \log n \rceil + g(k), \tag{1}$$

for some function $g(k)$ independent of n, and the total distance traveled is

$$D(n) \le k \cdot \delta_{\min} + 2g(k), \tag{2}$$

with the factor of 2 accounting for the fact that Δ must return to the origin after each face probe. In other words, for any constant k, we get

$$P(n) \le k\lceil \log n \rceil + \mathcal{O}(1) \quad \text{and} \quad D(n) \le k \cdot \delta_{\min} + \mathcal{O}(1).$$

In the appendix, we show that $g(k) < 3^k$ and discuss several realistic assumptions under which we can bound $g(k)$ to just $k(k+1)$, as long with experimental evidence supporting this going up to $k = 8$. The biggest bottleneck for adapting CBS for $k > 2$ dimensions to the L_1 metric is that the faces of cross-polytopes are not, in general, other cross-polytopes, although we conjecture that a similar algorithm can be developed.

Input-Sensitive Probe Complexity. There may be a scenario where the probes themselves are very expensive, and we may not have a good estimate for the distance $\delta_{\min}$ to the nearest POI. In this case, Δ can start by performing an exponential search from the origin, increasing the probe radius by a factor of 2 per probe, until the first successful probe. Doing so requires at most $\lceil \log \delta_{\min} \rceil$ probes, and would limit the initial search area radius to $\delta_{\min} \le n < 2\delta_{\min}$, from which we can use our preferred algorithm to find the POI. For example, when using the central binary search algorithm, we achieve $P(n) \le (k+1)\lceil \log \delta_{\min} \rceil + \mathcal{O}(1)$.

References

[1] Antonio Albanese, Vincenzo Sciancalepore, and Xavier Costa-Pérez. SARDO: An automated search-and-rescue drone-based solution for victims localization. *IEEE Transactions on Mobile Computing*, 21(9):3312–3325, 2022. doi:10.1109/TMC.2021.3051273.

[2] Esther M Arkin, Michael A Bender, Sándor P Fekete, Joseph SB Mitchell, and Martin Skutella. The freeze-tag problem: how to wake up a swarm of robots. *Algorithmica*, 46:193–221, 2006.

[3] Esther M Arkin, Michael A Bender, and Dongdong Ge. Improved approximation algorithms for the freeze-tag problem. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 295–303, 2003.

[4] Muhammad Atif, Rizwan Ahmad, Waqas Ahmad, Liang Zhao, and Joel J. P. C. Rodrigues. Uav-assisted wireless localization for search and rescue. *IEEE Systems Journal*, 15(3):3261–3272, 2021. doi:10.1109/JSYST.2020.3041573.

[5] Francesco Betti Sorbelli, Sajal K. Das, Cristina M. Pinotti, and Simone Silvestri. Range based algorithms for precise localization of terrestrial objects using a drone. *Pervasive and Mobile Computing*, 48:20–42, 2018. doi:10.1016/j.pmcj.2018.05.007.

[6] Nicolas Bonichon, Arnaud Casteigts, Cyril Gavoille, and Nicolas Hanusse. Freeze-tag in L_1 has wake-up time five, 2024. arXiv:2402.03258.

[7] Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and Saeed Odak. Euclidean freeze-tag problem on plane. In *36th Canadian Conference on Computational Geometry (CCCG)*, pages 199–205, 2024.

[8] Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943. URL: http://www.jstor.org/stable/2235930.

[9] Ding-Zhu Du and Frank Kwang-Ming Hwang. *Combinatorial Group Testing and Its Applications*. World Scientific, 2nd edition, 2000.

[10] Jens Niklas Eberhardt, Nikolas Peter Breuckmann, and Christiane Sigrid Eberhardt. Multi-stage group testing improves efficiency of large-scale COVID-19 screening. *Journal of Clinical Virology*, 128:104382, 2020.

[11] David Eppstein, Michael T Goodrich, and Daniel S Hirschberg. Improved combinatorial group testing algorithms for real-world problem sizes. *SIAM Journal on Computing*, 36(5):1360–1375, 2007.

[12] Gray Frank. Pulse code communication, March 17 1953. US Patent 2,632,058.

[13] Ofek Gila, Michael T. Goodrich, Zahra Hadizadeh, Daniel S. Hirschberg, and Shayan Taherijam. The Marco Polo problem: A combinatorial approach to geometric localization, 2025. arXiv:2504.17955.

[14] Michael T Goodrich and Daniel S Hirschberg. Improved adaptive group testing algorithms with applications to multiple access channels and dead sensor diagnosis. *Journal of Combinatorial Optimization*, 15:95–121, 2008.

[15] Mikael Hammar, Bengt J Nilsson, and Mia Persson. The online freeze-tag problem. In *LATIN 2006: Theoretical Informatics: 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006. Proceedings 7*, pages 569–579. Springer, 2006.

[16] Jyh-How Huang, Saqib Amjad, and Shivakant Mishra. CenWits: a sensor-based loosely coupled search and rescue system using witnesses. In *3rd ACM International Conference on Embedded Networked Sensor Systems*, SenSys '05, page 180–191, 2005. doi:10.1145/1098918.1098938.

[17] Richard Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.

[18] Balmukund Mishra, Deepak Garg, Pratik Narang, and Vipul Mishra. Drone-surveillance for search and rescue in natural disaster. *Computer Communications*, 156:1–10, 2020. doi:10.1016/j.comcom.2020.03.012.

[19] Lehilton Lelis Chaves Pedrosa and Lucas de Oliveira Silva. Freeze-tag is NP-hard in 3d with L_1 distance. *Procedia Computer Science*, 223:360–366, 2023.

[20] D. C. Schedl, I. Kurmi, and O. Bimber. An autonomous drone for search and rescue in forests using airborne optical sectioning. *Science Robotics*, 6(55):eabg1188, 2021. doi:10.1126/scirobotics.abg1188.

[21] Yunpeng Sun, Xiangming Wen, Zhaoming Lu, Tao Lei, and Shan Jiang. Localization of wifi devices using unmanned aerial vehicles in search and rescue. In *IEEE/CIC International Conference on Communications in China*, pages 147–152, 2018. doi:10.1109/ICCChinaW.2018.8674518.

[22] Yulun Tian, Katherine Liu, Kyel Ok, Loc Tran, Danette Allen, Nicholas Roy, and Jonathan P. How. Search and rescue under the forest canopy using multiple UAVs. *The International Journal of Robotics Research*, 39(10-11):1201–1221, 2020. doi:10.1177/0278364920929398.

A Omitted Details and Results

In this section, we provide more details and results omitted from the main text.

The 3D Domino Algorithm. In the main text, we described how the 3D domino algorithm can indefinitely half the remaining search volume for every probe once reaching a configuration we refer to as a 4-domino. In this configuration, we have four equally sized $d \times d \times d$ cubes, where a POI is known to exist in one, and all the rest are known to be empty, with the cubes arranged in a 2x2x1 grid. See Figure 6. The key step glossed over in the main text is how to efficiently reach this configuration.

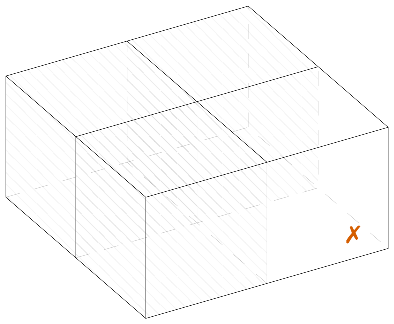


Figure 6: The configuration referred to as a 4-domino. After reaching this configuration, we can indefinitely halve the remaining search volume with every probe.

Perhaps the most natural idea that comes to mind is to split the initial search cube into its 8 equally sized octants, and then sequentially probe each of them until we reach a 4-domino configuration. However, since the 4-domino configuration relies on the fact that 3 regions are known to be empty, it is possible that no such configuration exists, e.g., if there exists a POI in every octant. Recall that our objective with the 3D domino algorithm is to minimize the number of probes to $3\lceil\log n\rceil + \mathcal{O}(1)$, meaning that we must in general reduce the search volume by a factor of 8 after every 3 probes. Each octant has an eighth of the volume of the original search cube, and thus, as long as one of the first 3 probes succeeds, we still succeed in reducing the search volume by a factor of 8. The case of a POI in every octant is therefore a very lucky case, since we are able to reduce the search volume by a factor of 8 on every probe. Our goal therefore, is to conduct the first 3 probes in such a way that even if all three of them fail, we are guaranteed to reach a 4-domino configuration. Unfortunately, this is not possible. Figure 7 depicts one possible scenario. In fact, no matter where the first 3 probes are conducted, POIs may be placed adversarially such that after those probes no 4-domino configuration exists.

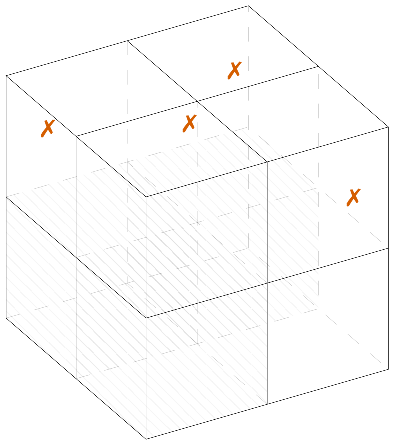


Figure 7: One possible configuration of the search cube after the first 3 probes of the 3D domino algorithm. Since there exists a POI in every octant in the top half of the cube, there does not exist any 4-domino configuration in this layer.

This poses a problem, since if we only discover a POI on our fourth probe and continue the algorithm recursively from that octant, we have spent 4 probes to reduce the search volume by a factor of 8, which, if done repeatedly would lead to a probe complexity of $P(n) = 4\lceil\log n\rceil$. The 3D domino algorithm accounts for this by considering another intermediate configuration, which is very similar to the 2-domino configuration of the 2D domino algorithm. In this configuration, we have two equally sized $d \times d \times d$ cubes, where a POI is known to exist in one, and the other is known to be empty. From this configuration, while we are generally able to reduce the search volume by a factor of 8 with every 3 probes, returning to another 2-domino configuration. However, if we get unlucky, these 3 probes in the 2-domino configuration may fail to reduce the volume as desired—but the only way for this to occur would induce a valid 4-domino configuration, from which point we can recurse indefinitely. Thus, such a failure can only occur once throughout the course of our algorithm. See Figure 8 for a depiction of the first 3 probes in the 3D 2-domino procedure. Note that this figure assumes that the first probe is successful without loss of generality. If either the second or third probe is successful, we can simply recurse into another 2-domino configuration with an eighth of the volume. Otherwise, we must just perform one more probe in either of the remaining two octants to reach a 4-domino configuration.

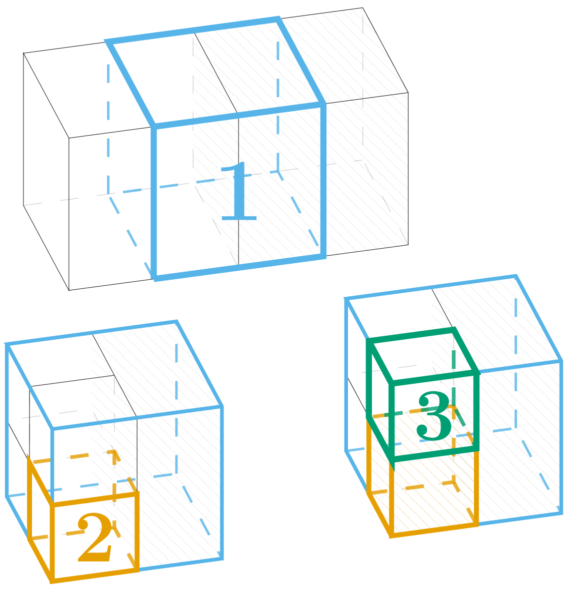


Figure 8: A depiction of the first 3 probes in the 3D 2-domino procedure. If all 3 probes fail, we are guaranteed to reach a 4-domino configuration.

Reaching this 2-domino configuration is easy, since, just like in the 2D domino algorithm, we can simply probe adjacent octants. Thus, our 3D domino algorithm begins by probing adjacent octants until one succeeds. If the first probe is successful, we simply recurse into that octant. If a 4-domino configuration is reached, we can recurse indefinitely. Finally, if a 4-domino configuration is not possible, since at least one probe failed and we probe adjacent octants, we are guaranteed to reach a 2-domino configuration, which we recurse into either indefinitely or until it becomes a 4-domino configuration. Overall, the worst case scenario is where we probe 7 octants before determining the location of the POI, after which we recurse optimally in a 4-domino configuration indefinitely. Such a scenario would lead to a probe complexity of $P(n) = 7 + 3\lceil \log n/2 \rceil = 3\lceil \log n \rceil + 4$.

Reflecting on this algorithm, we see how, unlike the 2D domino algorithm which required a single domino configuration which was easily reachable, the 3D domino algorithm required a second intermediate configuration with a separate algorithm on how to move from one configuration to the other. This more tailored approach was not necessary for the generalized CBS algorithm, which is broken up into simple, binary search-like phases, independent of the search space dimension.

Nevertheless, it might be interesting to consider whether more domino-like configurations could be used to extend our domino algorithms to higher dimensions. It is straightforward to see that for a k-dimensional search space, there exists a 2^{k-1}-domino configuration, where all but one of the 2^{k-1} cubes are known to be empty, after which point we can recurse indefinitely, halving the search volume with every probe, such as the 2-domino configuration in the 2D case and the 4-domino configuration in the 3D case. Further, you can consider a 2^{k-2}-domino configuration, a 2^{k-3}-domino, and so on, until reaching a 2-domino configuration. While it is trivial to reach a 2-domino configuration from the initial hypercube in any dimension, it is not clear whether a transition algorithm exists from a 2-domino to a 4-domino configuration for dimensions higher than 3. In fact, we conjecture that this transition breaks down in dimensions higher than 3, ending such a generalization, but we leave this as an open question.

The Generalized CBS Algorithm. In the Generalized CBS algorithm, after the binary search of each phase which determines an approximate remaining distance, $\tilde{\delta_p}$ to the nearest POI, we must determine which of the k coordinates to set to $\pm\tilde{\delta_p}$. In Algorithm 2, we describe a procedure to do this by iteratively probing smaller and smaller faces of our p-cube using a procedure we refer to as `ProbeFace`, which relies on the following lemma.

Lemma 2 *(same as Lemma 1) Consider a* $(p-a)$*-dimensional face* f *of a* p*-cube of radius* $\tilde{\delta_p}$ *centered at the origin.* f *can be defined by a unique vector* $s \in \{-1, 0, 1\}^p$ *of length* p*, where 'a' values are fixed to* ± 1*, and the remaining values are 0, such that any point* x *of* f *satisfies* $s \cdot x = a\tilde{\delta_p}$*.* `ProbeFace(f)` *can be performed by moving* Δ *to coordinate* s *and conducting a probe with radius* $\tilde{\delta_p} - 1$*.*

Proof. Let us briefly consider which qualities we require from our probe.

1. The probe must not include any regions outside of our p-cube.

2. The probe should include the entire face f, besides at most 1 unit of padding from its boundary which will be covered by subsequent probes of lower-dimensional faces.

3. The probe for a $(p-a)$-dimensional face f should not include any other regions that are not already known to be empty.

We prove the first property by contradiction. Assume, for the sake of contradiction, that the probe includes a point x that is outside the p-cube of radius $\tilde{\delta_p}$ centered at the origin. A point x is outside this p-cube if, for at least one coordinate i, its absolute value $|x_i|$ is greater than $\tilde{\delta_p}$. Without loss of generality, let us assume $x_i > \tilde{\delta_p}$ for this specific coordinate i. The probe is centered at s (where $s_j \in \{-1, 0, 1\}$, so $s_j \leq 1$ for all coordinates j) and has a radius of $\tilde{\delta_p} - 1$. For a point x to be included in this probe (which is itself a p-cube), it must satisfy $|x_j - s_j| \leq \tilde{\delta_p} - 1$ for all coordinates j.

Focusing on our specific coordinate i: We have $x_i > \tilde{\delta_p}$. We also know that $s_i \leq 1$. Consider the difference $x_i - s_i$. Since $x_i > \tilde{\delta_p}$ it follows that: $x_i - s_i > \tilde{\delta_p} - s_i$. Given $s_i \leq 1$, we have $\tilde{\delta_p} - s_i \geq \tilde{\delta_p} - 1$. Therefore, $x_i - s_i > \tilde{\delta_p} - 1$ and consequently $|x_i - s_i| > \tilde{\delta_p} - 1$. This result directly contradicts the condition for x to be inside the probe.

We now turn to the second property. The probe P is centered at the coordinate s and has a radius of $\tilde{\delta_p} - 1$ in each dimension. Thus, a point y is within this probe if it satisfies $|y_j - s_j| \leq \tilde{\delta_p} - 1$ for all coordinates $j = 1, \ldots, p$.

Let $I_{\text{free}} = \{j \mid s_j = 0\}$ be the set of indices for coordinates where s_j is zero. These are the "free" coordinates along which the face f extends. For any $x \in f$, $x_j \in [-\tilde{\delta_p}, \tilde{\delta_p}]$ for $j \in I_{\text{free}}$. Similarly, let $I_{\text{fixed}} = \{j \mid s_j \in \{-1, 1\}\}$ be the set of indices for coordinates where s_j is non-zero. These are the "fixed" coordinates. For any point $x \in f$, its j-th coordinate is determined by s_j: $x_j = s_j \tilde{\delta_p}$ if $j \in I_{\text{fixed}}$.

Let us consider an arbitrary point $x \in f$. We wish to show that x satisfies the condition $|x_j - s_j| \leq \tilde{\delta_p} - 1$ for each coordinate j, considering two cases:

- If $j \in I_{\text{fixed}}$, then $x_j = s_j \tilde{\delta_p}$ for any point $x \in f$. We examine the condition for the probe: $|x_j - s_j| = |s_j \tilde{\delta_p} - s_j| = |s_j(\tilde{\delta_p} - 1)|$. Since $s_j \in \{-1, 1\}$, we have $|s_j| = 1$. Therefore, $|s_j(\tilde{\delta_p} - 1)| = |\tilde{\delta_p} - 1|$. Assuming $\tilde{\delta_p} \geq 1$ (so that the probe radius $\tilde{\delta_p} - 1$ is non-negative), we have $|\tilde{\delta_p} - 1| = \tilde{\delta_p} - 1$. Thus, for all $j \in I_{\text{fixed}}$, the condition $|x_j - s_j| \leq \tilde{\delta_p} - 1$ is satisfied.

- If $j \in I_{\text{free}}$, then $s_j = 0$. For a point $x \in f$, its j-th coordinate x_j can range within $[-\tilde{\delta_p}, \tilde{\delta_p}]$. The condition for x to be included in the probe P with respect to this j-th coordinate is $|x_j - s_j| \leq \tilde{\delta_p} - 1$. Since $s_j = 0$, this simplifies to $|x_j| \leq \tilde{\delta_p} - 1$. This means that for coordinates $j \in I_{\text{free}}$, the probe P includes points $x \in f$ if their j-th coordinate x_j lies in the interval $[-(\tilde{\delta_p} - 1), \tilde{\delta_p} - 1]$. While this is notably *not* the entire range of x_j (which is $[-\tilde{\delta_p}, \tilde{\delta_p}]$), it includes everything besides a 1-unit "padding" from the boundary of the face f along this coordinate's axis, satisfying the second property.

Finally, we turn to the third property. We know from our algorithm, Algorithm 2, that all higher-dimensional faces have been probed and are known to be empty. Further, we know that the internal region (besides a 1-unit padding) of the p-cube is empty. Thus, the probe must only avoid probing lower or equal-dimensional faces. Consider one such $(p - a')$-dimensional face f' where $a' \geq a$, which is defined by a vector $s' \in \{-1, 0, 1\}^p$. Since f' has at least a' coordinates fixed to ± 1, it must be the case that s' has at least one coordinate j such that $s'_j = \pm 1$ and $s_j = 0$. This means that for all points in f', the j-th coordinate is fixed to $s'_j \tilde{\delta_p}$, i.e., that for any point $x' \in f'$, we have $|x_j| = \tilde{\delta_p}$. However, since our probe only has radius $\tilde{\delta_p} - 1$, and since $s_j = 0$, we have $|x'_j - s_j| = |x'_j| = \tilde{\delta_p} > \tilde{\delta_p} - 1$. Thus, our probe does not include any points in f'. □

We can now bound the total number, $g(k)$, of `ProbeFace` calls made throughout the course of the algorithm. A p-cube contains $3^p - 1$ faces. Thus, the total number of faces we must probe is technically $3^p - 2$ since we can skip the final probe.

However, since we start from higher-dimensional faces, if we take more than $2p$ probes, we are guaranteed to find a POI in a face which is lower by at least 2 dimensions, reducing p by 2 instead of just 1. It is straightforward to see that, in the worst case, a POI will only be located in the final probe in the initial, k-dimensional cube, requiring $3^k - 2$ probes. While this result is exceedingly unlikely, and is not possible if we require that all of the POIs' coordinates differ in magnitude by at least 1, it is nevertheless possible under our assumptions, and thus we only bound $g(k)$ by $3^k - 2$.

As alluded to above, however, if we do require that all of the POIs' coordinates differ in magnitude by at least 1, we know that the POIs will always be located in a facet of our p-cube in each phase, requiring at most $2p$ probes. Over the course of the algorithm, our total number of probes will therefore be bounded by

$$\sum_{p=1}^{k} 2p = k(k+1),$$

thus bounded $g(k) = \mathcal{O}(k^2)$.

Alternatively, if we only consider cases where k is small, i.e., $k = o(\log \log n)$, then we know that the total number `ProbeFace` calls made for any dimension p, 3^p, is bounded by $3^{\log \log n} = o(\log n)$. However, if we take more than $2p$ probes, decreasing p by 2, we can charge our extra probes to the binary search of the $p-1$ face we skipped, which we expected to take $\lceil \log n \rceil$ probes. Thus our worst case is the same as our previous example, and we similarly obtain that $g(k) = \mathcal{O}(k^2)$.

B Experimental Results

In this section we compare how each algorithm performs experimentally in terms of our three metrics: the number of probes made, the distance traveled, and the number of POI responses. We compared results between the domino algorithms (in 2D and 3D), the orthant algorithm (in 1D – 8D), and the generalized CBS algorithm (also in 1D – 8D). Each algorithm was executed 60 million times, where $n = 2^{20}$. The POIs were placed at uniformly random locations in the search

	Domino Algorithms				Orthant Algorithm				Generalized CBS Algorithm			
k	σ	Avg	Max	Bound	σ	Avg	Max	Bound	σ	Avg	Max	Bound
1D	—	—	—	—	**0.00**	**0.95**	**0.95**	**1.00**	0.00	**0.95**	1.00	**1.00**
2D	0.04	**1.92**	**1.95**	**2.05**	0.18	2.14	2.85	3.00	**0.03**	1.93	2.00	2.15/2.25
3D	0.11	**2.92**	**3.05**	**3.20**	0.46	4.16	6.35	7.00	**0.07**	2.96	3.10	3.40/4.10
4D	—	—	—	—	0.98	8.02	13.2	15.0	**0.10**	**4.00**	**4.25**	**4.75/7.75**
5D	—	—	—	—	2.00	15.6	26.1	31.0	**0.14**	**5.06**	**5.40**	**6.20/16.8**
6D	—	—	—	—	4.02	30.9	50.6	63.0	**0.17**	**6.15**	**6.65**	**7.75/42.0**
7D	—	—	—	—	8.05	61.3	103	127	**0.21**	**7.26**	**7.95**	**9.40/116**
8D	—	—	—	—	16.1	122	209	**255**	**0.25**	**8.40**	**9.30**	11.2/336

Table 1: Normalized number of probes ($P/\log n$) for different search algorithms across dimensions. The best (lowest) values are highlighted in bold.

area, namely such that each of their coordinates was uniformly random in the range $[0, n]$. These experiments focused on the L_∞ metric.

It is clear from our results that both the domino and CBS algorithms significantly outperform the generalized orthant algorithms for any dimension $k > 1$, where in 1D space all algorithms are equivalent. It is also possible to see that, as the dimension k increases, the normalized number of probes for the CBS algorithm gradually increases, as predicted by the $g(k)$ dependence in its probe complexity. See Figure 9.

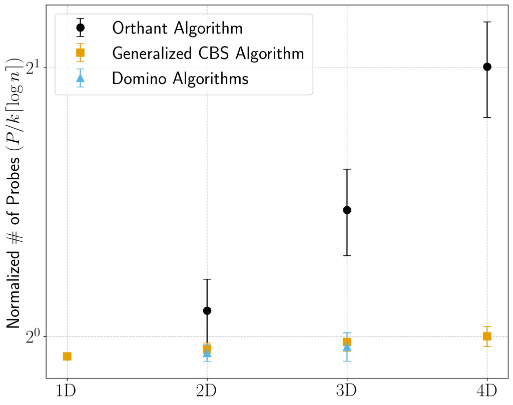


Figure 9: Simulation results for $P/k\lceil\log n\rceil$. Error bars represent one standard deviation from the mean.

Regarding the distance traveled by the search point, Δ, we see that not only does the CBS algorithm, being instance-optimal with regards to distance traveled, travel significantly less distance with respect to the nearest POI's distance on average, but it also has a much lower variability in the distance traveled than the other algorithms. See Figure 10.

Number of Probes Made (P). In Table 1 we compare the number of probes made by each algorithm operating in different dimensions. Each result is normalized by $\log n$, such that the theoretical lower bound for any algorithm after the normalization is just k for the k-dimensional case. As expected, the domino

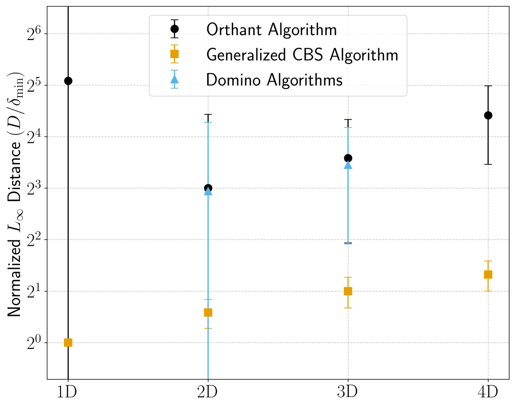


Figure 10: Simulation results for $D/\delta_{\min}$, where $n = 2^{20}$.

algorithms, specifically designed to minimize the number of probes, outperform the other algorithms in 2D and 3D across the average, maximum, and theoretical bounds. Interestingly, the generalized central binary search (CBS) algorithm, despite performing slightly worse than the domino algorithms, performed slightly more consistently, with a smaller standard deviation than the domino algorithms. The orthant algorithms, as expected, perform very poorly in this metric, with their average appearing to be, as expected, roughly $2^k/2$. It should be noted that the generalized CBS algorithm may theoretically only find a POI when probing the final face it queries in the k-dimensional space, recalling that there may be up to $g(k) = 3^k - 1$ faces. In theory, this may, in 8 dimensions and beyond, lead to a worse performance than the orthant algorithm. However, recall that only the first $2k$ faces are facets, with all the others being progressively lower dimensional. These first $2k$ facets in general will be much much larger than all the subsequent faces, meaning that, unless a POI is placed adversarially, it is most probable that the POI will be found in one of the first $2k$ faces. In fact, we can estimate this probability numerically.

Consider the case where we have a k-dimensional

	Domino Algorithms				Orthant Algorithm				Generalized CBS Algorithm			
k	σ	Avg	Max	Bound	σ	Avg	Max	Bound	σ	Avg	Max	Bound
1D	—	—	—	—	$\sim 10^4$	27.8	$\sim 10^7$	$\sim 10^6$	**0.00**	**1.00**	**1.00**	**1.00**
2D	17.2	7.68	$\sim 10^4$	$\sim 10^6$	17.1	8.00	$\sim 10^4$	$\sim 10^6$	**0.29**	**1.50**	**2.00**	**2.00**
3D	7.17	11.0	$\sim 10^3$	$\sim 10^7$	8.14	12.0	$\sim 10^3$	$\sim 10^7$	**0.41**	**2.00**	**3.00**	**3.00**
4D	—	—	—	—	10.4	21.3	$\sim 10^3$	$\sim 10^7$	**0.50**	**2.50**	**3.99**	**4.00**
5D	—	—	—	—	16.7	40.0	$\sim 10^3$	$\sim 10^7$	**0.58**	**3.00**	**4.97**	**5.00**
6D	—	—	—	—	29.7	76.8	$\sim 10^3$	$\sim 10^8$	**0.65**	**3.50**	**5.94**	**6.00**
7D	—	—	—	—	55.2	149	$\sim 10^3$	$\sim 10^8$	**0.71**	**4.00**	**6.86**	**7.00**
8D	—	—	—	—	105	293	$\sim 10^3$	$\sim 10^8$	**0.76**	**4.50**	**7.76**	**8.00**

Table 2: Normalized L_∞ distance $(D/\delta_{\min})$ for different search algorithms across dimensions.

width-1 shell with radius $\delta_{\min}$, with only one POI placed uniformly at random in the shell. We probe each facet with a radius $\delta_{\min} - 1$, covering a volume contained by that facet of $2^{k-1}(\delta_{\min} - 1)^{k-1}$. Overall, the $2k$ facet probes cover a volume of $k2^k(\delta_{\min} - 1)^{k-1}$. The total volume of the k-dimensional shell is the total volume of the hypercube, $2^k\delta_{\min}^k$, minus the volume of the inner shell, which is $2^k(\delta_{\min} - 1)^k$. Together, we get that the ratio of these volumes is

$$\frac{k(\delta_{\min} - 1)^{k-1}}{\delta_{\min}^k - (\delta_{\min} - 1)^k}.$$

Assuming that $\delta_{\min}$ is at least k, this ratio is minimized as $k \to \infty$, where it approaches $\frac{1}{e-1} \approx 0.58$, where e is Euler's number. In other words, assuming that a POI is placed uniformly at random, and that the search area radius is only moderately larger than its dimension, it is most probable that the POI will be found in one of the first $2k$ facets. Surely this is the case for our experiments where the search area radius $n = 2^{20}$ is much larger than the dimension $k \leq 8$. As such, we included not only the theoretical bounds for the generalized CBS algorithm, but also the bounds assuming we always find the POI in one of the facets we probe during each phase. Our experimental results confirm that, despite the millions of simulations performed, this bound was never violated. This further supports our claim that under reasonable conditions, $g(k)$ can be more accurately bounded by $k(k+1)$.

Distance Traveled (D). In Table 2 we compare the distance traveled by Δ for each algorithm operating in different dimensions. Each result is normalized by $\delta_{\min}$, such that the theoretical lower bound for any algorithm after the normalization is just 1. We expect any *instance-optimal* algorithm, with respect to the distance traveled, to consistently travel within a constant multiple of this distance. Our generalized CBS algorithm expects to travel a distance of at most k times the minimum distance, for example. On the other hand, non-instance-optimal algorithms, such as the orthant and domino algorithms, expect to travel a distance more dependent on the area of the search area, n, and the dimension, k, essentially ignoring the position of the nearest POI. In the worst case, the POI is placed directly at the origin, but since we disallowed this in our experiments, they would be placed at distance 1 from the origin, thus maximizing the ratio of the distance traveled to the minimum distance. While this extreme case evidently does not occur in our experiments, it is clear from our results how much better the generalized CBS algorithm performs. It outperforms the orthant and domino algorithms on all metrics and across all dimensions, often by several orders of magnitude. Reassuringly, the generalized CBS algorithm never travels a distance greater than k times the minimum distance, and seems to on average travel a distance of $(k+1)/2$ times greater, with a moderate standard deviation of roughly 20% of the average. It is worth noting that the search point, Δ, in domino algorithms appears to travel marginally less distance than a search point in the orthant algorithm.

Number of Responses (R). Up until this point, the orthant algorithm has performed poorly when compared to the domino and generalized CBS algorithms. Under the number of responses metric, however, the orthant algorithm is able to shine, performing the best on all metrics and across all dimensions, as shown in Table 3, tying with the generalized CBS algorithm only in 1D. This supports our claim that the orthant algorithm is a good choice when POI responses either carry a high cost or pose a high risk. It is worth noting that while the generalized CBS algorithm certainly performs worse, especially in higher dimensions, in 1–3D it performs comparably. Our results support the claim that the generalized CBS algorithm is a good default choice, performing competitively with regards to the number of probes and responses, while performing by far the best in terms of distance traveled.

	Domino Algorithms				Orthant Algorithm				Generalized CBS Algorithm			
k	σ	Avg	Max	Bound	σ	Avg	Max	Bound	σ	Avg	Max	Bound
1D	—	—	—	—	0.11	0.47	**0.95**	**1.00**	**0.11**	**0.47**	**0.95**	1.05
2D	0.15	0.93	1.70	2.05	**0.09**	**0.71**	**0.95**	**1.00**	0.15	0.94	1.75	2.10
3D	0.18	1.39	2.35	3.20	**0.07**	**0.83**	**0.95**	**1.00**	0.18	1.42	2.45	3.15
4D	—	—	—	—	**0.05**	**0.89**	**0.95**	**1.00**	0.21	1.88	3.00	4.20
5D	—	—	—	—	**0.04**	**0.92**	**0.95**	**1.00**	0.23	2.33	3.50	5.25
6D	—	—	—	—	**0.03**	**0.94**	**0.95**	**1.00**	0.25	2.78	4.20	6.30
7D	—	—	—	—	**0.02**	**0.94**	**0.95**	**1.00**	0.27	3.22	4.65	7.35
8D	—	—	—	—	**0.01**	**0.95**	**0.95**	**1.00**	0.28	3.65	5.25	8.40

Table 3: Normalized number of responses ($R/\log n$) for different search algorithms across dimensions.

Approximating 2-Clique in Unit Disk Graphs

A. Karim Abu-Affash* Paz Carmi† Iliya Lisin‡

Abstract

A 2-clique in a connected graph G is a subset S of vertices of G, such that, for every two vertices in S, the distance between them in G is at most 2. In this paper, we consider the problem of computing a maximum 2-clique in unit disk graphs and give a $\frac{2}{3}$-approximation algorithm for the problem that runs in $O(n^{4.5})$ time. This improves the $\frac{1}{2}$-approximation algorithm by Pattillo et al. [10].

1 Introduction

Let $G = (V, E)$ be a connected graph. For two vertices u and v in V, let $d_G(u, v)$ denote the *distance* between u and v in G, i.e., the minimum number of edges on any path between u and v in G. A k-clique in G is a subset S of vertices of G, such that, for every two vertices in S, $d_G(u, v) \leq k$. In the *k-clique problem*, we are given a connected graph G and an integer $k \geq 1$, and the goal is to compute a k-clique in G of maximum cardinality. Indeed, for $k = 1$, the k-clique problem is equivalent to the maximum clique problem, which is one of the fundamental problems in theoretical computer science [5]. It is not only NP-hard but even hard to approximate the maximum clique problem within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$, unless $P = NP$ [14]. The k-clique problem is also known to have hardness of approximation of $n^{1/2-\epsilon}$, unless $P = NP$ [4].

The k-clique problem is a relaxation of the maximum clique problem and is motivated by cluster-detection needs, which arise in a wide variety of applications [2,13]. For instance, finding clusters in networks helps in understanding and analyzing the structure of the network.

Let P be a set of points in the plane. For two points $p, q \in P$, let $|pq|$ denote the Euclidean distance between p and q. The *unit disk graph* on P is the undirected graph $G = (P, E)$, such that $(p, q) \in E$ if and only if $|pq| \leq 1$. Unit disk graphs provide a graph-theoretic model and have many applications in ad hoc wireless networks [7], where two wireless nodes can communicate if they are within the unit Euclidean distance away from each other. In this paper, we consider the k-clique problem in unit disk graphs, for $k = 2$, and give a $\frac{2}{3}$-approximation algorithm for the problem that runs in $O(n^{4.5})$ time, where n is the number of the points of P.

1.1 Related works

The k-clique problem was first introduced and studied in general graphs [2,4,11,12]. Balasundaram et al. [4] proved that for any k, the k-clique problem is NP-hard. For unit disk graphs, it has been shown that, for $k = 1$, a maximum k-clique (a maximum clique) can be computed in polynomial time [7]. Pattillo et al. [10] studied the k-clique problem in unit disk graphs, for $k = 2$, and gave a $\frac{1}{2}$-approximation algorithm for the problem. However, for $k \geq 2$, to the best of our knowledge, the hardness of the k-clique problem in unit disk graphs is still open.

Another relaxation of the maximum clique problem is the k-club problem. A k-club in a graph G is a subgraph G' of G of diameter k, i.e., every two vertices in G' have a distance of at most k. The k-club problem asks to compute a k-club of maximum size. Clearly, every k-club is a k-clique, but not vice versa. Computing a maximum k-club is also known to be NP-hard. [6], and cannot be approximated within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$, unless $P = NP$ [3]. Abu-Affash et al. [1] gave a constant-factor approximation algorithm for the maximum k-club problem in unit disk graphs.

The rest of this paper is organized as follows. In Section 2, we prove some lemmas related to pairwise intersecting unit disks. In Section 3, we prove that every 2-clique in unit disk graphs is 3-dominated. In Section 4, we first show how to compute a maximum 2-dominated 2-clique in unit disk graphs, and then we use this algorithm to approximate the 2-clique problem.

2 Preliminaries

In this section, we introduce some definitions, notation, and some lemmas that will be used in the paper. Let P be a set of points in the plane and let $|P|$ denote the number of points in P. For a point $u \in P$, let D_u denote the unit disk (i.e., the disk of radius one) centered at u. For two points p and q on a circle C, let $\overline{pq}$ denote the

*Software Engineering Department, Shamoon College of Engineering, Israel, abuaa1@sce.ac.il

†Department of Computer Science, Ben-Gurion University, Israel, carmip@cs.bgu.ac.il. Work by P. Carmi was partially supported by the Lynn and William Frankel Center for Computer Sciences.

‡Software Engineering Department, Shamoon College of Engineering, Israel, lisin.i.1997@gmail.com

line segment connecting p and q, and let $\widehat{pq}$ denote the circular arc of C from p to q in clockwise order.

Remark. To avoid ambiguity in terminology, we will use the operations $\cap$, $\cup$, $\setminus$ for disks. For example, when we say $D_u \cap D_v$, we refer to the intersection region of the disks D_u and D_v.

Let $\mathcal{D}$ be the set of all the unit disks centered at the points of P. In this section, we assume that the disks of $\mathcal{D}$ are pairwise intersecting. We say that $\mathcal{D}$ is *Helly* if every three disks in $\mathcal{D}$ have a nonempty intersection; otherwise, we say that $\mathcal{D}$ is not *Helly*. We assume that every two intersecting disks D_u and D_v in $\mathcal{D}$ intersect in exactly two points, and their intersection region $D_u \cap D_v$ is a *lens* that we denote by $L(D_u, D_v)$; see Figure 1. Thus, for every three pairwise intersecting disks D_x, D_y, and D_z in $\mathcal{D}$, either (i) $D_x \cap D_y \cap D_z = \emptyset$ and their boundaries create a *concave circular triangle*, or (ii) $D_x \cap D_y \cap D_z \neq \emptyset$ and their boundaries create a *convex circular triangle* or their intersection is a lens created by pair of them; see Figure 1.

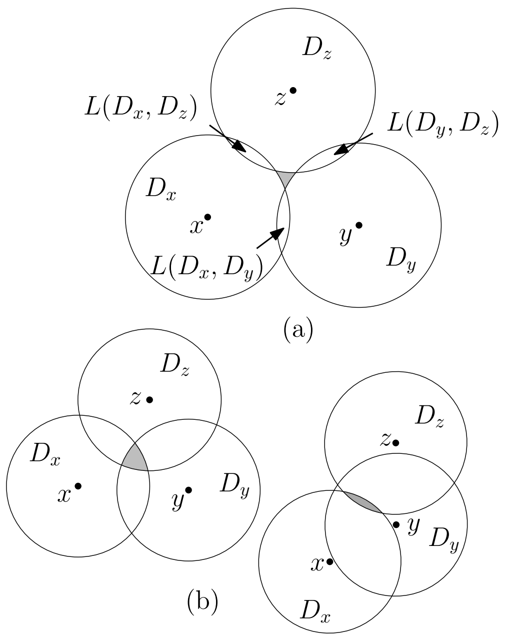


Figure 1: (a) $D_x \cap D_y \cap D_z = \emptyset$. (b) $D_x \cap D_y \cap D_z \neq \emptyset$.

Lemma 1 *Let D_a, D_b, and D_c be three disks in $\mathcal{D}$, such that $D_a \cap D_b \cap D_c = \emptyset$ and $|ab| \leq 1$. Then, for every two points $u \in L(D_a, D_c)$ and $v \in L(D_b, D_c)$, $|uv| < 1$.*

Proof. Assume, w.l.o.g., a and b are on the x-axis and c is above the x-axis. Let p be the left vertex of the lens $L(D_a, D_c)$ and let q be the right vertex of the lens $L(D_b, D_c)$; see Figure 2. Thus, it is sufficient to prove that $|pq| \leq 1$. Let t be the right vertex of $L(D_a, D_c)$. Let $D_{b'}$ be the unit disk centered at $b' \in L(D_a, D_b)$ and obtained by rotating D_b around a until it touches t; see Figure 2(a). Let q' the right vertex of $L(D_c, D_{b'})$. Since $D_a \cap D_b \cap D_c = \emptyset$, we have $|pq| < |pq'|$. We show that $|pq'| \leq 1$. Since $\overline{b'q'}$ is parallel to $\overline{tc}$ and $\overline{tc}$ is parallel to $\overline{ap}$, $\overline{b'q'}$ is parallel to $\overline{ap}$; see Figure 2(b). Moreover, since $|b'q'| = |ap| = 1$, we have $|pq'| = |ab'| = |ab| \leq 1$, which completes the proof. $\square$

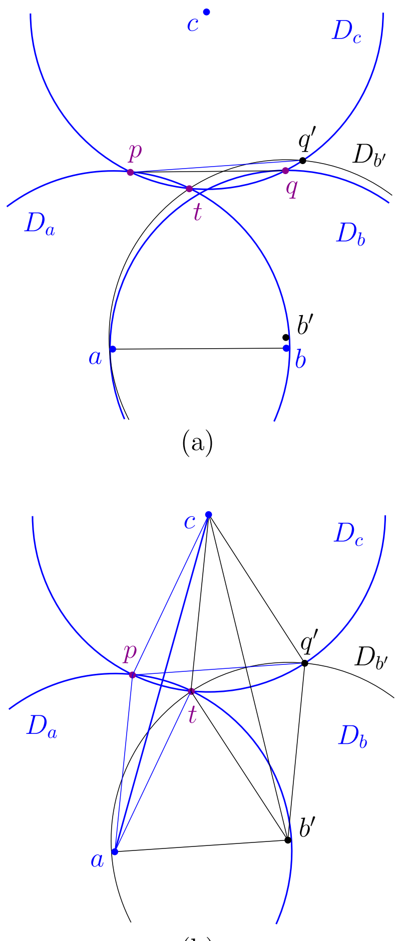


Figure 2: An illustration for the proof of Lemma 1.

The following corollary follows from the proof of Lemma 1.

Corollary 2 *Let D_x and D_y be two disks of $\mathcal{D}$ and assume that x and y are on the x-axis. Let α and β be the top and bottom vertices of $L(D_x, D_y)$, respectively; see Figure 3. Let u and v be the top vertices of $L(D_\beta, D_x)$ and $L(D_\beta, D_y)$, respectively. Let u' and v' be the top vertices of $L(D_\alpha, D_x)$ and $L(D_\alpha, D_y)$, respectively. Then, $|uv'| = |vu'| = 1$.*

3 Dominating sets in 2-cliques

Let P be a set of points in the plane and let $G = (P, E)$ be the unit disk graph of P in the containment model.

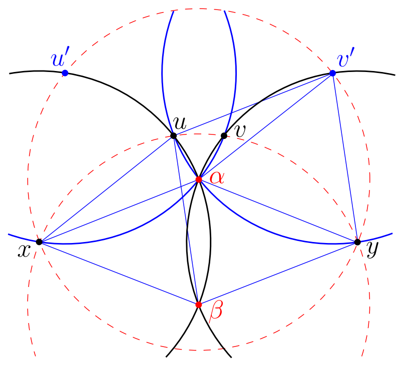


Figure 3: An illustration for Corollary 2.

That is, there is an edge between u and v in G, if v is contained in the disk D_u, i.e., $|uv| \leq 1$, where $|uv|$ denotes the Euclidean distance between u and v. Recall that $d_G(u,v)$ denotes the distance between u and v in G (i.e., the minimum number of edges on any path between u and v in G).

For a point $u \in P$, we say that u *dominates* a point $v \in P$, if $v \in D_u$. Pattillo et al. [10] proved that every 2-clique S in G is 4-dominated. That is, there are four points of P that dominate all the points in S. In this section, we prove that every 2-clique in G is 3-dominated.

Theorem 3 *Any 2-clique in G is 3-dominated.*

Proof. Let S be a 2-clique in G and let $\mathcal{D}$ be the set of the unit disks centered at the points of S. Since S is a 2-clique, the disks in $\mathcal{D}$ are pairwise intersecting and every two disks D_p and D_q in $\mathcal{D}$ contain a point of P in $L(D_p, D_q)$. We distinguish between two cases.

Case 1: $\mathcal{D}$ is not Helly.
Let D_x, D_y, and D_z be three disks in $\mathcal{D}$ that create a concave circular triangle of maximum area. Pattillo et al. [10] proved the following lemma.

Lemma 4 (Lemma 3 in [10]) *Any disk $D_q \in \mathcal{D}$ must contain at least one of the lenses $L(D_x, D_y)$, $L(D_y, D_z)$, or $L(D_x, D_z)$.*

Thus, any three points $a \in L(D_x, D_y)$, $b \in L(D_y, D_z)$, and $c \in L(D_x, D_z)$ of P dominate all the points of S.

Case 2: $\mathcal{D}$ is Helly.
In this case, every three disks in $\mathcal{D}$ have a nonempty intersection and, by Helly's theorem [8, 9], there is a nonempty region R that is contained in all the disks of $\mathcal{D}$ and its boundary is created by a set of different disks of $\mathcal{D}$. We refer to these disks as *border disks*; see Figure 4.

Pattillo et al. [10] proved that if S is not 2-dominated, then there are three border disks D_x, D_y, and D_z in $\mathcal{D}$, such that $D_x \cap D_y \cap D_z$ does not contain any point of S. Assume, w.l.o.g., that x and y are on the x-axis and z is above the x-axis.

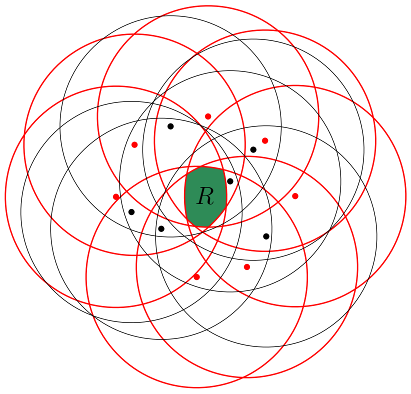


Figure 4: The common intersection region R (green) is created by the border disks (red) of $\mathcal{D}$.

Since S is a 2-clique, there are three points a, b, and c of P, such that $a \in L(D_x, D_y)$, $b \in L(D_y, D_z)$, and $c \in L(D_x, D_z)$. Notice that D_a, D_b, and D_c are pairwise intersecting, and there are no points of S lying in the outer face of $D_a \cup D_b \cup D_c$; otherwise, one of the disks D_x, D_y, and D_z is not a border disk. Thus, we have the following corollary.

Corollary 5 *If there are three points a, b, and c of P, such that $a \in L(D_x, D_y)$, $b \in L(D_y, D_z)$, $c \in L(D_x, D_z)$, and $D_a \cap D_b \cap D_c \neq \emptyset$, then a, b, and c dominate all the points of S.*

Due to space limitations, the proofs of the following two lemmas are moved to the appendix.

Lemma 6 *If $|xy| \leq 1$, then any two points $b \in L(D_y, D_z)$ and $c \in L(D_x, D_z)$ of P together with x or y dominate all the points of S.*

Lemma 7 *If there are three points a, b, and c of P, such that $a \in L(D_x, D_y)$, $b \in L(D_y, D_z)$, $c \in L(D_x, D_z)$, $D_a \cap D_b \cap D_c = \emptyset$, and $|ab| \leq 1$, then a, b, and x dominate all the points of S.*

By Lemma 6, if $\min\{|xy|, |yz|, |xz|\} \leq 1$, then S is 3-dominated, and we are done. Moreover, by Corollary 5 and Lemma 7, if there are three points $a \in L(D_x, D_y)$, $b \in L(D_y, D_z)$, and $c \in L(D_x, D_z)$ of P, such that $D_a \cap D_b \cap D_c \neq \emptyset$ or $\min\{|ab|, |bc|, |ac|\} \leq 1$, then S is 3-dominated, and we are also done. Therefore, we assume that, $\min\{|xy|, |yz|, |xz|\} > 1$ and, for every three points $a \in L(D_x, D_y)$, $b \in L(D_y, D_z)$, and $c \in L(D_x, D_z)$ of P, $D_a \cap D_b \cap D_c = \emptyset$ and $\min\{|ab|, |bc|, |ac|\} > 1$. Moreover, we also assume that there are points of S outside $D_x \cup D_y \cup D_z$; otherwise, x, y, and z dominate all the points of S, and we are done.

Let α and β be the top and the bottom vertices of $L(D_x, D_y)$, respectively. Let i be the left vertex of $L(D_y, D_z)$ and let j be the right vertex of $L(D_x, D_z)$;

see Figure 5(a). Let p be a point of S lying outside $D_x \cup D_y \cup D_z$, and assume, w.l.o.g., that p is below the x-axis. Since D_x and D_y are border disks, D_p must contain both i and j, and hence contain all the region $(D_x \cap D_y) \setminus D_z$. Thus, we have the following corollary.

Corollary 8 *For every point $a \in (D_x \cap D_y) \setminus D_z$, $|pa| \leq 1$.*

Since S is a 2-clique, there is a point of P in $L(D_z, D_p)$.

Lemma 9 *For every point $q \in P$ in $L(D_z, D_p)$, $q \in D_\beta \setminus (D_x \cup D_y)$.*

Proof. Since $|xp| > 1$, $|yp| > 1$, and D_p contains the points i and j, $L(D_p, D_\alpha)$ is contained in $L(D_\alpha, D_\beta)$; see Figure 5(a). Similarly, since $|xz| > 1$ and $|yz| > 1$, $L(D_z, D_\beta)$ is contained in $L(D_\alpha, D_\beta)$. Thus, since $L(D_z, D_p)$ is contained in both $L(D_z, D_\beta)$ and $L(D_p, D_\alpha)$, $L(D_z, D_p)$ is contained in $L(D_\alpha, D_\beta)$. Therefore, $|\alpha q| \leq 1$ and $|\beta q| \leq 1$, which implies that $q \in D_\beta$ and that $L(D_x, D_y)$ is contained in D_q.

If $q \in (D_x \cup D_y)$, then $q \in L(D_x, D_z)$ or $q \in L(D_y, D_z)$; see Figure 5(a). In both cases, since $L(D_x, D_y)$ is contained in D_q, we have $|qa| \leq 1$, for every point $a \in L(D_x, D_y)$, which contradicts the assumption that for every three points $a \in L(D_x, D_y)$, $b \in L(D_y, D_z)$, and $c \in L(D_x, D_z)$ of P, $\min\{|ab|, |bc|, |ac|\} > 1$. Therefore, $q \in D_\beta \setminus (D_x \cup D_y)$. □

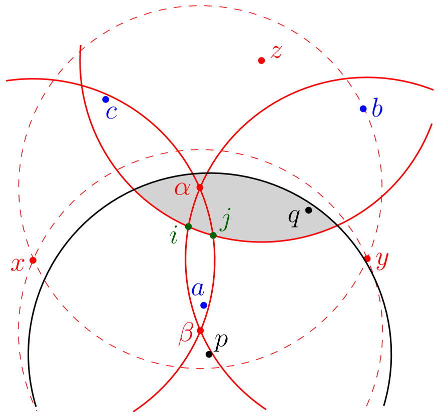
By Lemma 9, there is a point of P in the region $(D_z \cap D_\beta) \setminus (D_x \cup D_y)$. Let $q \in P$ be the lowest point in $(D_z \cap D_\beta) \setminus (D_x \cup D_y)$ (i.e., the point of smallest y-coordinate); see Figure 5(b).

Lemma 10 *For every point $t \in S$ that lies above the x-axis and outside $D_x \cup D_y$, $|tq| \leq 1$.*

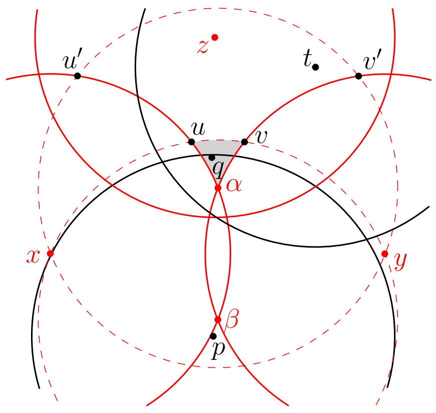
Proof. Let u and v be the top vertices of $L(D_\beta, D_x)$ and $L(D_\beta, D_y)$, respectively. Let t be a point of S that lies above the x-axis and outside $D_x \cup D_y$; see Figure 5(b). We show that D_t contains all the points in the circular triangle $\triangle \alpha uv$, including q, and thus $|tq| \leq 1$.

Since D_x, D_y, and D_z are border disks and t is above the x-axis, D_t must contain α. Thus, $t \in D_\alpha$. Let u' and v' be the top vertices of $L(D_\alpha, D_x)$ and $L(D_\alpha, D_y)$, respectively. By Corollary 2, $|uv'| = |vu'| = 1$. Thus, since $t \in D_\alpha$ and outside $D_x \cup D_y$, we have $\max\{|ut|, |vt|\} \leq |uv'| = |vu'| = 1$. Therefore, D_t contains all the points in the circular triangle $\triangle \alpha uv$. □

By Lemma 10, q dominates all the points of S that lie above the x-axis and outside $D_x \cup D_y$. Let $a \in P$ be a point in $L(D_x, D_y)$ that is closest to q, and notice that, by Corollary 8, a dominates all the points of S that lie below the x-axis and outside $D_x \cup D_y$.



(a)



(b)

Figure 5: (a) $q \in (D_x \cup D_y)$. $L(D_z, D_p)$ (gray) is contained in $L(D_\alpha, D_\beta)$. (b) $q \in (D_z \cap D_\beta) \setminus (D_x \cup D_y)$.

Thus, if q and a dominate all the points of S that are contained in D_x (resp., D_y), then q, a, and y (resp., q, a, and x) dominate all the points of S, and we are done. Therefore, we assume that there are points of S in $(D_x \cup D_y) \setminus (D_q \cup D_a)$.

Let U and V denote the sets of points of S in $D_x \setminus (D_q \cup D_a)$ and in $D_y \setminus (D_q \cup D_a)$, respectively. Observe that every point $u \in U$ (resp., $v \in V$) must contain α, otherwise this contradicts that $u \notin D_a$ (resp., $v \notin D_a$) or that x (resp., y) is a border disk.

Lemma 11 *There exist three points $u \in U$, $v \in V$, and $d \in L(D_u, D_v)$, such that $d \in D_x \setminus (D_y \cup D_z)$ and dominates all the points of U; or $d \in D_y \setminus (D_x \cup D_z)$ and dominates all the points of V.*

Proof. We select the point $u \in U$, such that the bottom intersection point u' of D_u with D_y is the highest. Similarly, we select the point $v \in V$, such that the bottom intersection point v' of D_v with D_x is the highest; see Figure 6. Thus, u' is contained in all the disks centered at the points of U and v' is contained in all the disks centered at the points of V; otherwise, this contradicts the selection of u and v. In particular, every disk centered at a point of U must contain the arc $\widehat{u'\alpha}$ of D_y and every disk centered at a point of V must contain the arc $\widehat{\alpha v'}$ of D_x.

Since S is a 2-clique, there is at least one point of P in $L(D_u, D_v)$. Moreover, since q is the lowest point of P in $D_z \cap D_\beta$ and every point of P in $L(D_u, D_v)$ is lower than q, the region $D_z \cap D_u \cap D_v$ is contained in $L(D_z, D_\beta)$ and is empty of points of P. Thus, every point of P in $L(D_u, D_v)$ must be in $(D_x \cup D_y) \setminus D_z$. In particular, since every point in $L(D_u, D_v)$ is closer to q than a, and a is the closest point of P to q in $L(D_x, D_y)$, every point of P in $L(D_u, D_v)$ must be in $D_x \setminus (D_y \cup D_z)$ or in $D_y \setminus (D_x \cup D_z)$.

Let s be the bottom vertex of $L(D_u, D_v)$, and observe that, since a is in $L(D_x, D_y)$ but not in $D_u \cup D_v$, s must be in $D_x \cup D_y$. We distinguish between two cases.

Case 2.1: If $s \in L(D_x, D_y)$, then we select an arbitrary point $d \in P$ from $L(D_u, D_v)$. Thus, either $d \in D_x \setminus (D_y \cup D_z)$ or $d \in D_y \setminus (D_x \cup D_z)$. If $d \in D_x \setminus (D_y \cup D_z)$ (resp., $d \in D_y \setminus (D_x \cup D_z)$), then, since every disk centered at a point of U (resp., of V) contains the arc $\widehat{u'\alpha}$ (resp., $\widehat{\alpha v'}$), d dominates all the points of U (resp., of V); see Figure 6(a).

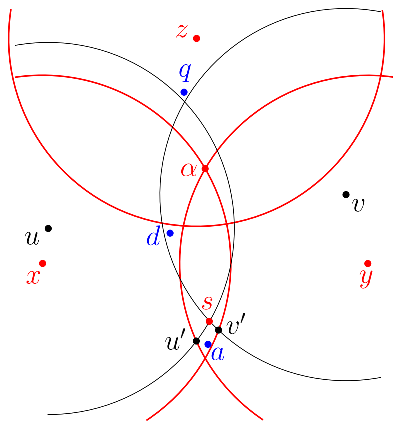
Case 2.2: If $s \notin L(D_x, D_y)$, then assume, w.l.o.g., that $s \in D_y \setminus D_x$. Thus, $L(D_v, D_x)$ is contained in $L(D_u, D_v)$; see Figure 6(b). We select an arbitrary point $d \in P$ from $L(D_v, D_x)$. Since every disk centered at a point of U contains the arc $\widehat{u'\alpha}$, $d \in D_x \setminus (D_y \cup D_z)$ and dominates all the points of U. □

Let $u \in U$, $v \in V$, and $d \in L(D_u, D_v)$ be three points satisfying Lemma 11, and assume, w.l.o.g., that d is in $D_x \setminus (D_y \cup D_z)$ and dominates all the points of U.

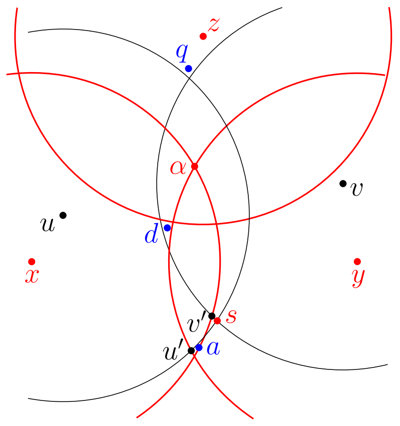
Lemma 12 *d dominates all the points of S that lie outside $D_q \cup D_y$.*

Proof. Let Q be the subset of P containing the points in $(D_z \cap D_\beta) \setminus (D_x \cup D_y)$ and recall that q is the lowest point in Q. Let Q' be the subset of S containing the points that lie outside $D_q \cup D_y$.

We consider the worst location of d. Recall that α and β are the top and the bottom vertices of $L(D_x, D_y)$, respectively, and let γ be the top vertex of $L(D_x, D_\beta)$; see Figure 7. We assume that D_z intersects D_x at the points α and γ, since $L(D_x, D_z)$ is contained in $L(D_x, D_{z'})$, for every point z' above the x-axis; otherwise $\mathcal{D}$ is not Helly or there is a point $c \in L(D_x, D_{z'})$ with $|ac| \leq 1$, which contradicts the assumption that $\min\{|ab|, |bc|, |ac|\} > 1$. We also assume that q is on the boundary of D_x and that v is on the intersection of D_q with D_β, since $L(D_x, D_{v'})$ is contained in $L(D_x, D_v)$, for every point $v' \in V$. Thus, $|\alpha q| = |\alpha q'|$. Finally, we assume that d is on the intersection of D_v with D_z, since, for every point d' in



(a)



(b)

Figure 6: An illustration for the proof of Lemma 11. (a) $s \in L(D_x, D_y)$. (b) $s \in D_y \setminus D_x$.

$L(D_x, D_v) \setminus (D_y \cup D_z)$, the points of Q' that are contained in $D_{d'}$ are also contained in D_d; see Figure 7. We show that d dominates all the points of Q'.

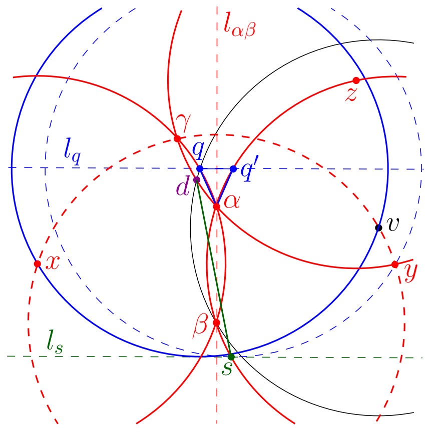
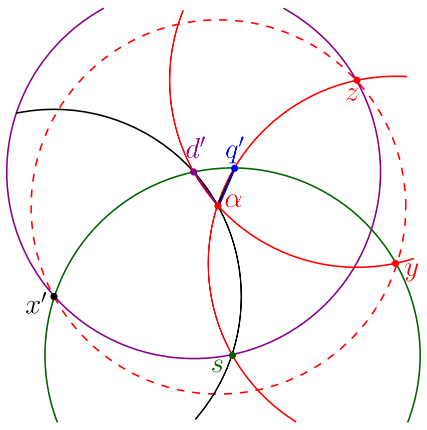


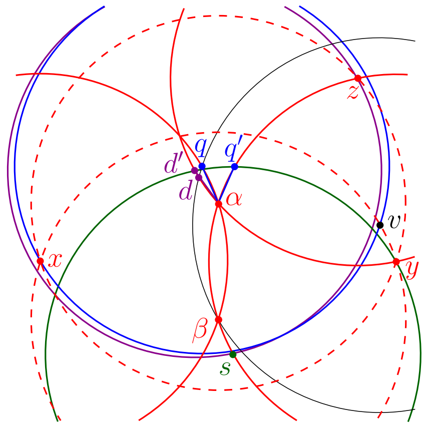
Figure 7: The worst location of d.

Let l_q be the horizontal line passing through q and let q' be the left intersection point of l_q with D_y. Let s be the bottom vertex of $L(D_{q'}, D_y)$ and let l_s be the horizontal line passing through s; see Figure 7. Thus, since q is the lowest point in Q, there are no points of Q' lying below l_s. Let $l_{\alpha\beta}$ be the vertical line passing through α and β. Since d is to the left of $l_{\alpha\beta}$ and there are no points of Q' lying below l_s, if d dominates s, then it dominates all the points of Q'. Thus, to prove the lemma, it is sufficient to prove that $|ds| \leq 1$.

Let x' be the left vertex of $L(D_s, D_\alpha)$ and let d' be the top vertex of $L(D_s, D_{x'})$; see Figure 8(a). Observe first that $|\alpha d'| = |\alpha q'|$, and, by Corollary 2, d' is on the boundary of D_z. Thus, since $\alpha \in D_s$, the arc $\widehat{\alpha d'}$ of D_z is contained in D_s. Moreover, since $v \notin D_s$ and v is on the boundary of D_q, v must be outside $D_{d'}$; see Figure 8(b). Hence, D_v intersects D_z on the arc $\widehat{\alpha d'}$, and thus $d \in D_s$. Therefore, $|ds| \leq 1$. This completes the proof of the lemma. □



(a)



(b)

Figure 8: An illustration for the proof of Lemma 12.

By Lemma 10, q dominates all the points of S that lie above the x-axis and outside $D_x \cup D_y$. By Lemma 12, d dominates all the points of S that lie outside $D_q \cup D_y$. Therefore, q, d, and y dominate all the points of S, which completes the proof of the theorem. □

4 The Algorithm

Let $G = (P, E)$ be a unit disk graph of a set P of n points in the plane. In this section, we present a polynomial-time algorithm that computes a 2-clique of G of size at least $\frac{2}{3}$ the size of a maximum 2-clique of G. We start by showing (in the appendix) a polynomial-time algorithm that computes a maximum 2-dominated 2-clique of G. (The same algorithm was used by Pattillo et al. [10]; however, we present this algorithm with more details for completeness.) Then, we use this algorithm to approximate the 2-clique problem.

4.1 Approximating the 2-clique problem

Let S^* be a maximum 2-clique of G and let S be a maximum 2-dominated 2-clique of G that is computed by the algorithm in the appendix. We prove that S contains at least $\frac{2}{3} \cdot |S^*|$ points.

Lemma 13 $|S| \geq \frac{2}{3} \cdot |S^*|$.

Proof. Since S^* is a 2-clique, by Theorem 3, S^* is 3-dominated. Let a, b, and c be three points of G that dominate S^*. Thus, at least $\frac{2}{3}$ of the points of S^* are contained in $D_a \cup D_b$, $D_b \cup D_c$, or $D_a \cup D_c$. That is, at least $\frac{2}{3}$ of the points of S^* are 2-dominated 2-clique of G. Since S is a maximum 2-dominated 2-clique of G, it contains at least $\frac{2}{3}$ of the points of S^*. □

The following theorem summarizes the result of this section.

Theorem 14 *There exists a $\frac{2}{3}$-approximation algorithm for the 2-clique problem in unit disk graphs that runs in $O(n^{4.5})$ time.*

5 Concluding Remarks

In this paper, we study the k-clique problem in unit disk graphs, for $k = 2$. We presented a polynomial-time $\frac{2}{3}$-approximation algorithm for this problem. The algorithm is based on the fact that we can find a maximum 2-dominated 2-clique in polynomial time, and its analysis is based on nice and interesting properties of the maximum (3-dominated) 2-clique in unit disk graphs.

Improving the approximation factor for $k = 2$ and solving/approximating the k-clique problem in unit disk graphs, for $k > 2$, are interesting open problems. Moreover, the complexity of computing a maximum k-clique in unit disk graphs is still open, for every $k \geq 2$. Hence, determining whether the k-clique problem in unit disk graphs is NP-hard is also an interesting open problem.

References

[1] A. K. Abu-Affash, P. Carmi, A. Maheshwari, P. Morin, M. Smid, and S. Smorodinsky. Approximating maximum diameter-bounded subgraph in unit disk graphs. *Discrete Comput. Geom.*, 66(4):1401–1414, 2021.

[2] R. D. Alba. A graph-theoretic definition of a sociometric clique. *J. Math. Sociol.*, 3:113–126, 1973.

[3] Y. Asahiro, Y. Doi, E. Miyano, K. Samizo, and H. Shimizu. Optimal approximation algorithms for maximum distance-bounded subgraph problems. *Algorithmica*, 80:1834–1856, 2018.

[4] B. Balasundaram, S. Butenko, and T. S. Novel approaches for analyzing biological networks. *J. Combin. Optim.*, 10:23–39, 2005.

[5] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. *Handbook of Combinatorial Optimization*, pages 1–74, Kluwer Academic Publishers, 1999.

[6] J.-M. Bourjolly, G. Laporte, and G. Pesant. An exact algorithm for the maximum k-club problem in an undirected graph. *European J. Oper. Res.*, 138:21–28, 2002.

[7] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Math.*, 86:165–177, 1990.

[8] E. Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresber. Dtsch. Math.-Ver.*, 32:175–176, 1923.

[9] E. Helly. Über Systeme von abgeschlossenen Mengen mit gemeinschaftlichen Punkten. *Monatshefte Math.*, 37(1):281–302, 1930.

[10] J. Pattillo, Y. Wang, and S. Butenko. Approximating 2-cliques in unit disk graphs. *Discrete Appl. Math.*, 166:178–187, 2014.

[11] J. Pattillo, N. Youssef, and S. Butenko. On clique relaxation models in network analysis. *European J. Oper. Res.*, 226:9–18, 2013.

[12] A. Veremyev and V. Boginski. Identifying large robust network clusters via new compact formulations of maximum k-club problems. *European J. Oper. Res.*, 218:316–326, 2012.

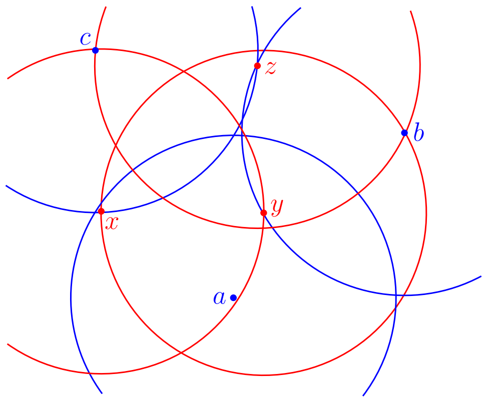
[13] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[14] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory Comput.*, 3:103–128, 2007.

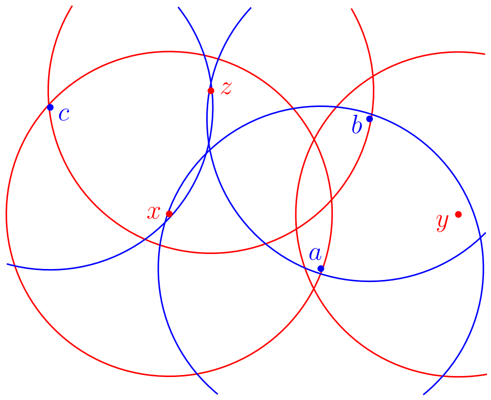
A Appendix

A.1 Proof of Lemma 6

Proof. If $|xy| \leq 1$, then both x and y are contained in $L(D_x, D_y)$. Thus, if we show that $D_x \cap D_b \cap D_c \neq \emptyset$ (resp., $D_y \cap D_b \cap D_c \neq \emptyset$), then, by Corollary 5, x, b, and c (resp., y, b, and c) dominate all the points of S. Assume, w.l.o.g., that $D_x \cap D_b \cap D_c = \emptyset$; see Figure 9(a). Since $D_x \cap D_b \cap D_c = \emptyset$ and $|cx| \leq 1$, by Lemma 1, $|yz| \leq 1$. Thus, $z \in D_y$. Moreover, $z \in D_b$ and $z \in D_c$. Therefore, $D_y \cap D_b \cap D_c \neq \emptyset$ and, by Corollary 5, y, b, and c dominate all the points of S. □



(a)



(b)

Figure 9: Three disks D_a, D_b, and D_c (blue) centered at $a \in L(D_x, D_y)$, $b \in L(D_y, D_z)$, and $c \in L(D_x, D_z)$, such that $D_a \cap D_b \cap D_c = \emptyset$. (a) $|xy| \leq 1$. (b) $|ab| \leq 1$.

A.2 Proof of Lemma 7

Proof. If $D_a \cap D_b \cap D_c = \emptyset$ and $|ab| \leq 1$, then, by Lemma 1, $|xz| \leq 1$; see Figure 9(b). Thus, $x \in L(D_x, D_z)$. Therefore, since $a \in D_x$ and $a \in D_b$, we have $D_a \cap D_b \cap D_x \neq \emptyset$ and, by Corollary 5, a, b, and x dominate all the points of S. □

A.3 Computing a maximum 2-dominated 2-clique

Let G^2 be the squared graph of G containing the same set of vertices as G, and there is an edge between two vertices u and v in G^2 if and only if $d_G(u, v) \leq 2$. Let S be a maximum 2-dominated 2-clique of G and let G^2_S be the subgraph of G^2 induced by S. Since S is a 2-clique, G^2_S is a clique. Therefore, to compute a maximum 2-dominated 2-clique of G, it is sufficient to compute a maximum clique in the subgraph of G^2 induced by the points in $D_u \cup D_v$, for every pair $u, v \in P$, and take the largest.

Let G^2_{uv} be the subgraph of G^2 induced by the points contained in $D_u \cup D_v$. We compute a maximum clique in G^2_{uv} as follows. Notice that the subgraph of G^2_{uv} induced by the points contained in D_u and the subgraph of G^2_{uv} induced by the points contained in D_v are cliques. Thus, the complement graph of G^2_{uv} is a bipartite graph. Since computing a maximum clique in G^2_{uv} is equivalent to computing a maximum independent set in the complement graph of G^2_{uv}, and computing a maximum independent set in a bipartite graph can be solved in $O(n^{2.5})$ time [7], we can find a maximum clique in G^2_{uv} in $O(n^{2.5})$ time. Since the algorithm goes over all pairs of points in P, the running time of the algorithm is $O(n^{4.5})$. The following theorem summarizes this result.

Theorem 15 *Given a unit disk graph G, one can compute in $O(n^{4.5})$ time a maximum 2-dominated 2-clique of G.*

Computing Maximum Cliques in Unit Disk Graphs [*]

Anastasiia Tkachenko[†] Haitao Wang[†]

Abstract

Given a set P of n points in the plane, the unit-disk graph $G(P)$ is a graph with P as its vertex set such that two points of P have an edge if their Euclidean distance is at most 1. We consider the problem of computing a maximum clique in $G(P)$. The previously best algorithm for the problem runs in $O(n^{7/3+o(1)})$ time. We show that the problem can be solved in $O(n \log n + nK^{4/3+o(1)})$ time, where K is the maximum clique size. The algorithm is faster than the previous one when $K = o(n)$. In addition, if P is in convex position, we give a randomized algorithm that runs in $O(n^{15/7+o(1)}) = O(n^{2.143})$ worst-case time and the algorithm can compute a maximum clique with high probability. For points in convex position, one special case we solve is when a point in the maximum clique is given; we present an $O(n^2 \log n)$ time (deterministic) algorithm for this special case.

Keywords: Cliques, unit-disk graphs, convex position

1 Introduction

A *clique* in an undirected graph is a subset of vertices such that every two vertices in the subset share an edge. Finding maximum cliques in graphs is a classical problem, which has applications across various domains, including bioinformatics [25], scheduling [39], coding theory [16], and social network analysis [5] (see also the survey [40]). Beyond its practical significance, the maximum clique problem is also closely connected to a number of important combinatorial optimization problems such as clique partitioning [33], graph clustering [30], and set packing [41]. These problems can often be directly formulated as instances of the maximum clique problem or contain subproblems that require computing a maximum clique.

In general, the maximum clique problem is NP-hard for graphs. This intractability persists even in geometrically constrained settings, such as intersection graphs of ellipses and triangles [3], hybrid models combining unit disks with axis-aligned rectangles [7, 15], and intersection graphs of rays [8]. In each of these cases, computing a maximum clique remains as hard as in general graphs.

Nevertheless, for several classes of geometric intersection graphs, the maximum clique problem becomes tractable. Polynomial time algorithms are known, for instance, for intersection graphs of axis-aligned rectangles [20], circle graphs [26], circular-arc graphs [4], and unit-disk graphs [13–15, 23]. In this paper, we focus on the maximum clique problem in unit-disk graphs. Unit-disk graphs arise naturally in modeling wireless communication networks, where devices with identical transmission ranges can interact if they are within range of each other [6, 13, 27, 28]. Relatively simple geometric structure of unit-disk graphs, paired with real-world relevance, has made them a central object of study in computational geometry and related fields.

Let P be a set of n points in the plane. The *unit-disk graph* of P, denoted $G(P)$, is the graph with vertex set P such that two points of P have an edge if their Euclidean distance is at most 1. Given P, we consider the problem of computing a maximum clique of $G(P)$.

1.1 Previous work

Clark, Colbourn, and Johnson [13] proposed the first polynomial time algorithm to compute a maximum clique in a unit-disk graph. Their algorithm runs in $O(n^{4.5})$ time, based on the following idea. For each edge (p, q) of $G(P)$ (i.e., $|pq| \leq 1$, where $|pq|$ is the distance of p and q), let $L(p, q)$ denote the intersection of the disk centered at p with radius $|pq|$ and the disk centered at q with radius $|pq|$; $L(p, q)$ is called a *lens*. A key insight of [13] was that the subgraph $G_{L(p,q)}(P)$ induced by points of P inside $L(p, q)$ is cobipartite (a graph is cobipartite if its complement graph is bipartite). Consequently, finding a maximum clique in $G_{L(p,q)}(P)$ reduces to finding a maximum independent set in its complement graph, which is bipartite. The latter problem in turn reduces to computing a maximum matching in the bipartite graph, which can be solved in $O(n^{2.5})$ time with n as the number of vertices in the graph [19]. If (p, q) is a farthest pair of points in a maximum matching M of $G(P)$, then a crucial observation is that all points of M must be in the lens $L(p, q)$. As such, by finding maximum cliques in the subgraphs for all lenses of all $O(n^2)$ of edges $G(P)$, one can compute a maximum clique in $O(n^{4.5})$ time.

Later, Aggarwal, Imai, Katoh, and Suri [2] gave an

*A full version of this paper is available on arXiv via the link: http://arxiv.org/abs/2506.21926. Omitted proofs and details can be found there.

†Kahlert School of Computing, University of Utah, Salt Lake City, UT 84112, USA.
{anastasiia.tkachenko,haitao.wang}@utah.edu

algorithm that can compute a maximum clique in a cobipartite subgraph of $G(P)$ (or equivalently compute a maximum independent set in its complement graph) in $O(n^{1.5} \log n)$ time by using the data structures in [18, 21]. Plugging this algorithm into the above framework of [13] leads to an $O(n^{3.5} \log n)$ time algorithm for computing a maximum clique in $G(P)$.

Eppstein and Erickson [14] subsequently derived an improved algorithm of $O(n^3 \log n)$ time by introducing an ordering of lenses and leveraging dynamic data structures. More specifically, the algorithm rotates a lens around each point in P, changing the corresponding cobipartite subgraph of the lens whenever a point enters or exits it. At each change, instead of computing a maximum clique from scratch, the authors developed a dynamic data structure to maintain the maximum clique by searching an alternating path in the graph, so that each point update (insertion/deletion) to the subgraph can be handled in $O(n \log n)$ time.

Recently Espenant, Keil, and Mondal [15] made a breakthrough and proposed the first subcubic time algorithm and their algorithm runs in $O(n^{2.5} \log n)$ time. It employs a divide-and-conquer approach. In the merge step, the authors discovered that it is sufficient to consider $O(n)$ lenses and compute a maximum clique for each of them, which takes $O(n^{2.5} \log n)$ time using the algorithm of [2]. Hence, the merge step can be completed in $O(n^{2.5} \log n)$ time, so is the overall algorithm. As noted in [23], Timothy Chan observed that finding a maximum clique in a cobipartite graph can be solved faster in $O(n^{4/3+o(1)})$ time by decomposing $G(P)$ into bipartite cliques [22, 36] and applying the maximum matching algorithm [11, 17]. Plugging this new algorithm into the algorithm of [15] leads to an $O(n^{7/3+o(1)})$ time algorithm to compute a maximum clique in $G(P)$.

1.2 Our results

In this paper, by using a grid and the above $O(n^{7/3+o(1)})$ time algorithm, we show that computing a maximum clique in $G(P)$ can be done in $O\big(n \log n + nK^{4/3+o(1)}\big)$ time, where K is the maximum clique size. If $K = o(n)$, then the runtime is smaller than the previous algorithm. In particular, for sufficiently small k, e.g., $k = O(\sqrt{\log n})$, the runtime is bounded by $O(n \log n)$.

We also consider a *convex position* case where points of P are in a convex position, i.e., every point of P is a vertex in the convex hull of P. In this case, we first show that if a point in a maximum clique of P is provided, then we can compute a maximum clique in $O(n^2 \log n)$ time. Combining this algorithm with the above $O\big(n \log n + nK^{4/3+o(1)}\big)$ time algorithm, we present a randomized algorithm that can compute a clique in $G(P)$ in worst-case $O(n^{15/7+o(1)}) = O(n^{2.143})$ time, and the clique is a maximum clique with high probability (i.e., with probability $1 - 1/n^c$ for an arbitrarily large constant c).

Although a constrained case, geometric problems for points in convex position have attracted much attention, e.g., Voronoi diagrams [1], k-center [12,32], independent set [31, 32], dispersion [31, 32], dominating set [32], ϵ-net [10], triangulation [24], Steiner tree [29], etc. In particular, when the disks are in convex position (e.g., every disk appears in the convex hull of all disks; but the disks may have different radii), it has been reported that a maximum clique can be computed in polynomial time [42].

Outline. The rest of the paper is organized as follows. After introducing some notation in Section 2, we present our algorithm for the general case (i.e., points of P are not necessarily in convex position) in Section 3. In Section 4, we describe the algorithm for the convex position case with the special assumption that a point in a maximum clique is provided. Section 5 finally solves the convex position case without the special assumption.

2 Preliminaries

We introduce some notation that will be used throughout the paper, in addition to those already defined in Section 1, such as P, n, and $G(P)$.

A *unit disk* is a disk of radius 1. For a point p in the plane, let D_p denote the unit disk centered at p, and ∂D_p the bounding circle of D_p. We use $x(p)$ and $y(p)$ to denote the x- and y-coordinates of p, respectively. Let $\ell_\perp(p)$ denote the vertical line through p.

For any two points p and q in the plane, let $|pq|$ represent their Euclidean distance, and $\overline{pq}$ the line segment with endpoints p and q.

For any subset $P' \subseteq P$, let $G(P')$ denote the subgraph of $G(P)$ with vertex set P'. For simplicity, we call a subset $P' \subseteq P$ a *clique* if $G(P')$ is a complete graph, i.e., the distance between every two points of P' is at most 1.

For any region R in the plane, we use $P(R) = P \cap R$ to denote the subset of points from P that lie in R.

3 The general case

In this section, we present our algorithm for computing a maximum clique in the unit-disk graph $G(P)$, without assuming that P is in convex position. We first solve a *decision problem*: Given a number k, determine whether $G(P)$ has a clique of size k (and if so, report such a clique). We show that after $O(n \log n)$ time preprocessing, given any k, the decision problem can be solved in $O(nk^{4/3+o(1)})$ time. Later we will show that a maximum clique of $G(P)$ can be computed by an exponential search and using the decision algorithm.

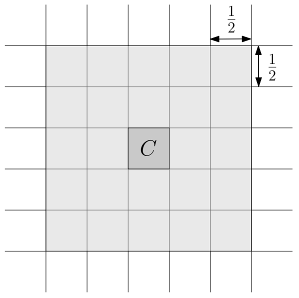


Figure 1: The gray cells are neighboring cells of C.

3.1 The decision problem

We first construct a set $\mathcal{C}$ of grid cells to capture the proximity information for the points of P. The technique of using grids has been used in various algorithms for problems in unit-disk graphs [9, 34, 35, 37, 38].

The set $\mathcal{C}$ has the following properties (see Figure 1): (1) Each cell of $\mathcal{C}$ is an axis-parallel square of side lengths 1/2. This implies that the distance between every two points in each cell is less than 1 and therefore points of P in the cell form a clique. (2) The union of all cells of $\mathcal{C}$ covers all the points of P. (3) Each cell $C \in \mathcal{C}$ is associated with a subset $N(C) \subseteq \mathcal{C}$ of $O(1)$ cells (called *neighboring cells* of C) such that for any point $q \in C$, $P(D_q) \subseteq \bigcup_{C' \in N(C)} P(C')$. (4) Each cell $C' \in \mathcal{C}$ is in $N(C)$ for $O(1)$ cells $C \in \mathcal{C}$.

An algorithm was provided in [34] for Lemma 1.

Lemma 1 [34] *The set $\mathcal{C}$ of size $O(n)$, along with $P(C)$ and $N(C)$ for all cells $C \in \mathcal{C}$, can be computed in $O(n \log n)$ time and $O(n)$ space.*

Our preprocessing step is to construct the set $\mathcal{C}$ by Lemma 1. Given a number k, below we present our algorithm to decide if $G(P)$ has a clique of size k.

For any cell $C \in \mathcal{C}$, if $|P(C)| \geq k$, we immediately find a clique of size k as the points of $P(C)$ form a clique. In the following, we assume $|P(C)| < k$ for all cells $C \in \mathcal{C}$.

Suppose $G(P)$ contains a clique M of size k. Let C be a cell that contains a point p of M. Then, $M \subseteq P(D_p)$. By the third property of $\mathcal{C}$, $P(D_p) \subseteq \bigcup_{C' \in N(C)} P(C')$. Hence, $M \subseteq \bigcup_{C' \in N(C)} P(C')$. Consequently, M can be found in $G(P_C)$ for $P_C = \bigcup_{C' \in N(C)} P(C')$. Since $N(C)$ has $O(1)$ cells C' and $|P(C')| < k$, we have $|P_C| = O(k)$. As such, applying the previous algorithm to P_C can compute M in $O(k^{7/3+o(1)})$ time.

With the above observation, our algorithm works as follows. We compute P_C for each cell $C \in \mathcal{C}$. This takes $O(n)$ time in total as $|N(C)| \leq 1$ and each cell $C' \in \mathcal{C}$ is in $N(C)$ for $O(1)$ cells $C \in \mathcal{C}$ (the fourth property of $\mathcal{C}$). If $|P_C| \geq k$, then we compute a maximum clique in $G(P_C)$, which takes $O(k^{7/3+o(1)})$ time. If the size of the computed clique is at least k, then we finish the algorithm and return the clique. The total time of the algorithm is $\sum_{C \in \mathcal{C}, |P_C| \geq k} O(k^{7/3+o(1)})$, which is $O(nk^{4/3+o(1)})$ due to the following observation.

Observation 1 $\sum\limits_{C \in \mathcal{C}, |P_C| \geq k} k^{7/3+o(1)} = O(nk^{4/3+o(1)})$.

We thus have the following lemma.

Lemma 2 *After $O(n \log n)$ time preprocessing, given any number k, we can decide whether $G(P)$ has a clique of size k (and if so, report such a clique) in $O(nk^{4/3+o(1)})$ time.*

3.2 The optimization problem: Finding a maximum clique

Using Lemma 2, the following theorem gives an algorithm to compute a maximum clique.

Theorem 3 *Given a set P of n points in the plane, we can find a maximum clique in the unit-disk graph $G(P)$ in $O(n \log n + nK^{4/3+o(1)})$ time, where K is the maximum clique size.*

Proof. We first perform the preprocessing step of Lemma 1, which takes $O(n \log n)$ time. Then, using the decision algorithm in Lemma 2, we can compute a maximum clique by doing exponential search. Specifically, we call the decision algorithm for $k = 1, 2, 4, 8$ until a value k such that the decision algorithm determines that $G(P)$ has a clique of size k but does not have a clique of size $2k$. Let $k' = k$. Hence, we have $K \in [k', 2k']$. The above calls the decision algorithm $O(\log k') = O(\log K)$ times. We then do binary search with $k \in [k', 2k']$ using the decision algorithm. This can compute a maximum clique by calling the decision algorithm another $O(\log K)$ times. As such, in total we call the decision algorithm $O(\log K)$ times, which takes $O(nK^{4/3+o(1)} \log K)$. Therefore, the time complexity of the overall algorithm is $O(n \log n + nK^{4/3+o(1)} \log K)$. Observe that the $\log K$ factor is subsumed by $K^{o(1)}$. The theorem thus follows. □

4 The convex position case with a given point

In this section, we consider the convex position case with a special assumption that a point p^* in a maximum clique of $G(P)$ is provided. With p^*, we give an algorithm that can compute a maximum clique in $O(n^2 \log n)$ time. We make a general position assumption that no three points of P lie on the same line.

Algorithm overview. Let M^* denote a maximum clique of $G(P)$ containing p^*. Our algorithm will maintain a subset $S \subseteq P$ such that $G(S)$ is cobipartite (note that $G(S)$ is cobipartite if and only S can be partitioned into two subsets S_1 and S_2 that are both cliques). During the course of the algorithm, there are $O(n)$ updates to S and each update is either to insert a point of P into S or delete a point from S, and after each update S is always cobipartite. We use the data structure of Eppstein and Erickson [14] (referred to as the *EE data structure*) to maintain a maximum clique of S so that each update can be handled in $O(n \log n)$ time (i.e., after each update, a maximum clique of S can be computed in $O(n \log n)$ time). Since there are $O(n)$ updates to S, the total time of the algorithm is $O(n^2 \log n)$. After all updates are done, among all cliques that have been computed, we return the largest one as our solution, i.e., a maximum clique for $G(P)$. To prove the correctness, we will show that at some moment during the algorithm $M^* \subseteq S$ holds (and therefore a maximum clique of that S is also a maximum clique of $G(S)$).

In what follows, we first define S in Section 4.1, and then explain the updates to S in Section 4.2. The exact definition of S will not be clear until Section 4.2, but Section 4.1 provides motivations to define S. Finally, we summarize the entire algorithm in Section 4.3.

4.1 Defining S

Since $p^* \in M^*$, we have $M^* \subseteq P(D_{p^*})$. Hence, it suffices to consider the points of P in the unit disk D_{p^*}. For notational convenience, we assume that all points of P are in D_{p^*}, i.e., $|pp^*| \leq 1$ for all points $p \in P$. Also, we rotate the plane so that p^* is the leftmost point and no two points of P have the same x- or y-coordinate.

Let $\hat{p}$ denote the rightmost point of P. Let $\hat{q}$ denote the rightmost point of M^*. Without loss of generality, we assume that $\hat{q}$ is on the upper hull of the convex hull of P (otherwise, the following discussion is symmetric; as we do not know whether $\hat{q}$ is on the upper or lower hull of P, we actually run two versions of the algorithm and then return the better solution, i.e., the larger clique found by the two versions).

Let P_u denote the set of points of P on the upper hull of P, and let $P_l = P \setminus P_u$. We assume that both p^* and $\hat{p}$ are in P_u but not in P_l.

We order the points of P_u from left to right as $p^* = p_1, p_2, \ldots, p_t = \hat{p}$. For each $p_i \in P_u$, let $S_u(i)$ denote a set of points $p \in P_u$ such that $x(p) \leq x(p_i)$ and $|pp_i| \leq 1$; similarly, define $S_l(i)$ as the set of points of $p \in P_l$ with $x(p) \leq x(p_i)$ and $|pp_i| \leq 1$. Let $S(i) = S_u(i) \cup S_l(i)$.

Suppose $\hat{q} = p_i$ for some i; then since every point p of M^* satisfies $x(p) \leq x(p_i)$ and $|pp_i| \leq 1$, it holds that $M^* \subseteq S(i)$. Our algorithm is based on this observation.

Our algorithm will scan the points of P_u from left to right. For each point $p_i \in P_u$, we will maintain a subset $S = S'(i)$ of points of P whose x-coordinates are less than or equal to $x(p_i)$ such that $S(i) \subseteq S'(i)$. We will compute a maximum clique for $G(S'(i))$ for each i, and finally return the largest clique found throughout the algorithm. This guarantees the correctness of the algorithm as $M^* \subseteq S(i) \subseteq S'(i)$. To compute maximum cliques of $G(S'(i))$ for all $1 \leq i \leq t$, we will show that each $G(S'(i))$ is cobipartite. Furthermore, if $S = S'(i)$, we will show that through a sequence of updates (point insertions/deletions) to S, we can obtain $S = S'(i+1)$ and after each update $G(S)$ is always cobipartite. In this way, we can use the EE data structure to compute the maximum cliques of $G(S'(i))$ for all $1 \leq i \leq t$. We will show that the total number of updates to S in the whole algorithm is $O(n)$. Hence, the total time of the algorithm is $O(n^2 \log n)$ since each update can be handled in $O(n \log n)$ time by the EE data structure.

To show that $S'(i)$ is cobipartite, it suffices to show that $S'(i)$ can be partitioned into two subsets S_1 and S_2 that are both cliques. To this end, we start with the following observation, which essentially shows that $G(S(i))$ is cobipartite.

Observation 2 *Both $S_u(i)$ and $S_l(i)$ are cliques.*

Remark. To define $S'(i)$ so that $G(S'(i))$ is cobipartite, due to Observation 2, it is tempted to have $S'(i) = S(i)$. However, as will be demonstrated later, having $S'(i) = S(i)$ may involve $\Omega(n^2)$ updates to $S'(i)$ for all $1 \leq i \leq t$. Fortunately, we discover a way to make a larger $S'(i)$ so that $G(S'(i))$ is still cobipartite and $O(n)$ updates to $S'(i)$ are guaranteed.

We define $S'(i) = S_u(i) \cup S'_l(i)$ for some subset $S'_l(i)$ such that $S_l(i) \subseteq S'_l(i) \subseteq P_l$. We will give the exact definition of $S'_l(i)$ later. By Observation 2, $S_u(i)$ is a clique in $G(P)$, called an *upper clique*. As will be seen later, $S'_l(i)$ is also a clique, called a *lower clique*.

4.2 Updating $S = S'(i)$

We now explain how to update $S = S'(i)$ incrementally from $i = 1$ to $i = t$. In the meanwhile, we will also give the exact definition of $S'_l(i)$.

We say that a point $p \in P$ is *inserted* into $S_u(i)$ if $p \in S_u(i)$ but $p \notin S_u(i-1)$, and p is *deleted* from $S_u(i)$ if $p \in S_u(i-1)$ but $p \notin S_u(i)$. We use the same terminology for $S_l(i)$ (and also for $S'_l(i)$, $S(i)$ and $S'(i)$).

Let h be the index in $[1, t]$ such that p_h is the point of P_u with the largest y-coordinate.

4.2.1 Maintaining the upper clique

The following lemma implies that the number of updates to the upper clique $S_u(i)$ from $i = 1$ to $i = t$ is $O(n)$.

Lemma 4 *Any point $p \in P$ can be inserted into or deleted from $S_u(i)$ at most once for all $1 \leq i \leq t$. In particular:*

1. *If $p \in S_u(i)$ for some $i \in [1, h]$, then $p \in S_u(j)$ holds for all $j \in (i, h]$.*

2. *If p is deleted from $S_u(i)$ for some $i \in [h, t]$, then $p \notin S_u(j)$ holds for all $j \in (i, t]$.*

4.2.2 Maintaining the lower clique

To define the lower clique $S'_l(i)$, which is based on $S_l(i)$, and maintain it for updates, we first make the following lemma, similar to Lemma 4(2) but for $i \in [1, h]$. The lemma implies that the number of updates to $S_l(i)$ for all $1 \le i \le h$ is $O(n)$.

Lemma 5 *Any point $p \in P$ can be deleted from $S_l(i)$ at most once for all $1 \le i \le h$. In particular, if p is deleted from $S_l(i)$ for some $i \in [1, h]$, then $p \notin S_l(j)$ holds for all $j \in (i, h]$.*

The above lemma is for indices $i \in [1, h]$. When i changes from h to t, however, the situation becomes more complicated since a point of P_l may be inserted into and deleted from $S_l(i)$ up to $\Omega(n)$ times, leading to a total of $\Omega(n^2)$ updates to $S_l(i)$ in the worst case. Consider the example illustrated in Figure 2. Let p be a point in P_l. Suppose P_u contains $O(n)$ points such that every other point from left to right lies at distance exactly $1 - \delta$ from p, while its immediate neighbors lie at distance $1 + \delta$, for some small $\delta > 0$. In this setting, when i changes from h to t, p must alternate between being inserted into and deleted from $S_l(i)$. Furthermore, if there are $O(n)$ such points p in P_l (e.g., all lying within distance δ from p), then each of them will behave similarly. In this case, the total number of updates to $S_l(i)$ for $h \le i \le t$ may reach $\Omega(n^2)$. In the following, we consider a new strategy to define and maintain a larger set $S'_l(i)$ with $S_l(i) \subseteq S'_l(i)$.

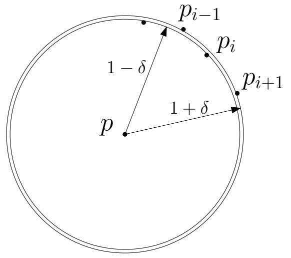


Figure 2: Illustration of a point $p \in P_l$ that may undergo $\Omega(n)$ updates to $S_l(i)$.

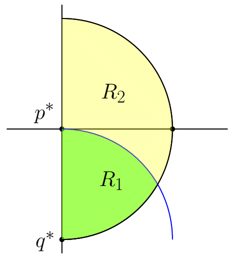


Figure 3: Illustrating the two regions R_1 and R_2 in the right half of D_{p^*}.

Let q^* be the point $(x(p^*), y(p^*)-1)$. We partition the right half of the unit disk D_{p^*} into two regions R_1 and R_2 by the boundary of D_{q^*} (see Figure 3). Specifically, define $R_1 = \{p \in D_{p^*} : p \in D_{q^*}, x(p) \ge x(p^*)\}$ and $R_2 = \{p \in D_{p^*} : p \notin D_{q^*}, x(p) \ge x(p^*)\}$. We have the following lemma for R_2.

Lemma 6 *For any point p of P_l in R_2, $|pp_i| \le 1$ holds for any $p_i \in P_u$ with $x(p) < x(p_i)$. This implies that if p is inserted into $S_l(i)$ for some $i \in [h, t]$, then p will always be in $S_l(j)$ for any $j \in (i, t]$.*

Defining $S'_l(i)$. In light of Lemma 6, when i changes from h to t, instead of maintaining $S_l(i)$, we maintain a superset $S'_l(i)$ of $S_l(i)$, defined as follows. For $1 \le i \le h$, we simply define $S'_l(i) = S_l(i)$. In general, assuming that $S'_l(i)$ has been defined for some $i \in [h, t]$, we define $S'_l(i+1)$ as follows. First, we put all points of $S'_l(i)$ to $S'_l(i+1)$. Then, for each point $p \in S_l(i+1) \setminus S'_l(i)$, we add p to $S'_l(i+1)$. Furthermore, if $p \in R_2$, then we check every point p' from $S'_l(i+1) \cap R_1$, and delete p' from $S'_l(i+1)$ if $|pp'| > 1$ (note that if $|pp'| > 1$, then p' cannot be in $S_l(i+1)$ since $S_l(i+1)$ is a clique by Observation 2; this means that no point $p \in S_l(i+1) \setminus S'_l(i)$ added to $S'_l(i+1)$ above will be later deleted; this further implies that the order we add points of $p \in S_l(i+1) \setminus S'_l(i)$ to $S'_l(i+1)$ does not matter).

The following lemma shows that $S'_l(i)$ is a "viable" alternative to $S_l(i)$.

Lemma 7 *For any $i \in [1, t]$, $S_l(i) \subseteq S'_l(i)$ and $S'_l(i)$ is a clique.*

The following lemma explains why $S'_l(i)$ is a better alternative to $S_l(i)$.

Lemma 8 *Any point $p \in P$ can be inserted into or deleted from $S'_l(i)$ at most $O(1)$ times for all $1 \le i \le t$.*

Definition 1 *Define $S'(i)$ to be $S_u(i) \cup S'_l(i)$ for all $1 \le i \le t$.*

Lemmas 4 and 8 together lead to the following.

Lemma 9 *Any point $p \in P$ can be inserted into or deleted from $S'(i)$ at most $O(1)$ times for all $1 \le i \le t$.*

4.3 Algorithm implementation and time analysis

Based on the above discussions, we implement our algorithm as follows. Initially when $i = 1$, we compute $S'(i) = S(i) = S_l(i) \cup S_u(i)$, which consists of a single point p_1. Therefore, $\{p_1\}$ is a maximum clique of $G(S'(i))$. In general, assume that we have computed $S'(i)$, whose graph $G(S'(i))$ is cobipartite, and its maximum clique $M(i)$. Then, we compute $S'(i+1)$ and its maximum clique $M(i+1)$ of $S'(i+1)$ as follows. Depending on whether $i + 1 \le h$, there are two cases.

The case $i + 1 \le h$. If $i + 1 \le h$, then we first compute $S'(i+1) = S(i+1) = S_l(i+1) \cup S_u(i+1)$. Note that $S_l(i+1)$ and $S_u(i+1)$ can be easily computed in $O(n)$ time by checking every point of P. We then compute the set $I(i+1) = S'(i+1) \setminus S'(i)$ and the set $D(i+1) = S'(i) \setminus S'(i+1)$. Notice that in order to obtain $S'(i+1)$ from $S'(i)$, $I(i+1)$ is the set of points to be inserted into $S'(i+1)$ while $D(i+1)$ is the set of points to be deleted from $S'(i)$. Given $S'(i)$ and $S'(i+1)$, the two sets $I(i+1)$ and $D(i+1)$ can be easily computed in $O(n \log n)$ time. Let $S = S'(i)$. Recall that we already know a maximum clique of S. To compute a maximum clique of $S'(i+1)$, we use the EE data structure [14] to maintain the maximum clique of S, and perform the following sequence of updates to S: First delete the points of $D(i+1)$ from S one by one in any arbitrary order and then insert the points of $I(i+1)$ into S one by one in any arbitrary order. Observe that at any moment of the sequence of the updates, S is either a subset of $S'(i)$ or a subset of $S'(i+1)$. As both $G(S'(i))$ and $G(S'(i+1))$ are cobipartite, $G(S)$ is always cobipartite during the above sequence of updates. Hence, the EE data structure can compute a maximum clique after each update in $O(n \log n)$ time. As such, the total time for computing a maximum clique $M(i+1)$ of $S'(i+1)$ is $O((|I(i+1)| + |D(i+1)| + 1) \cdot n \log n)$. Note that the "+1" is because we spend $O(n \log n)$ time computing $S'(i+1)$, $I(i+1)$, and $D(i+1)$.

The case $i+1 > h$. If $i+1 > h$, then we have $S'(i) = S_u(i) \cup S'_l(i)$ and $S'(i+1) = S_u(i+1) \cup S'_l(i+1)$. We first compute $S_u(i+1)$. Then, we compute $I_u(i+1) = S_u(i+1) \setminus S_u(i)$ and $D_u(i+1) = S_u(i) \setminus S_u(i+1)$. Next, we compute $S_l(i+1)$ and $I_l(i+1) = S_l(i+1) \setminus S'_l(i)$. Subsequently, for each point $p \in I_l(i+1)$, we check every point $p' \in S'_l(i) \cap R_1$ and add p' to $D_l(i+1)$ if $|pp'| > 1$. By definition, $S'(i+1)$ can be obtained from $S'(i)$ by deleting the points of $D(i+1) = D_l(i+1) \cup D_u(i+1)$ and inserting the points of $I(i+1) = I_l(i+1) \cup I_u(i+1)$.

Note that computing $S_u(i+1)$, $S_l(i+1)$, $I_u(i+1)$, $D_u(i+1)$, and $I_l(i+1)$ can be easily done in $O(n \log n)$ time. For each point $p \in I_l(i+1)$, we can spend $O(n)$ time checking every point $p' \in S'_l(i)$ to determine whether p' should be added to $D_l(i+1)$. Hence, computing $D_l(i+1)$ can be done in $O(|I_l(i+1)| \cdot n)$ time.

Let $S = S'(i)$. Since we already know a maximum clique of S, to compute a maximum clique of $S'(i+1)$, we use the EE data structure to maintain the maximum clique of S, and perform the following sequence of updates to S: First delete the points of $D(i+1)$ from S and then insert the points of $I(i+1)$ into S. As above, $G(S)$ is always cobipartite after each update. Hence, the EE data structure can compute a maximum clique after each update in $O(n \log n)$ time. As such, the total time for computing a maximum clique $M(i+1)$ of $S'(i+1)$ is $O((|I(i+1)| + |D(i+1)| + 1) \cdot n \log n)$.

To summarize, in either case, computing a maximum clique for each $S'(i+1)$ takes $O((|I(i+1)| + |D(i+1)| + 1) \cdot n \log n)$ time. By Lemma 9, we have $\sum_i (|I(i+1)| + |D(i+1)|) = O(n)$. Hence, the total time of the algorithm is $O(n^2 \log n)$. We thus obtain the following.

Theorem 10 *Given a set P of n points in convex position in the plane, if a point in a maximum clique of $G(P)$ is provided, then a maximum clique of $G(P)$ can be computed in $O(n^2 \log n)$ time.*

Note that for any point $p^* \in P$, our algorithm guarantees to find a clique no smaller than any clique containing p^*. Indeed, suppose that M is a clique of $G(P)$ that contains p^*. Then, our algorithm guarantees $M \subseteq S'(i)$ for some $i \in [1, t]$. Since the algorithm finds a maximum clique for $S'(i)$ for each $i \in [1, t]$, it can always find a clique of size at least $|M|$. Therefore, we have the following more general result.

Theorem 11 *Let P be a set of n points in convex position in the plane and p^* is a point of P. Suppose that k is the size of a clique of $G(P)$ that contains p^*. Then, we can compute a clique whose size is at least k in $O(n^2 \log n)$ time.*

5 The convex position case without a given point

We now consider convex position case problem without assuming a point in a maximum clique is given. We present a randomized algorithm for the problem by combining the results in Theorems 3 and 10.

Let K denote the maximum clique size of $G(P)$. We first apply the algorithm of Lemma 2 with $k = n^{6/7}$ to decide if $K < n^{6/7}$ in $O(n^{15/7+o(1)})$ time. Depending on whether $K < n^{6/7}$, we run two different algorithms.

If $K < n^{6/7}$, we run the algorithm of Theorem 3 to find a maximum clique in $O(n \log n + nK^{4/3+o(1)})$ time, which is bounded by $O(n^{15/7+o(1)})$.

If $K \geq n^{6/7}$, then we randomly pick a point p^* from P and run the algorithm in Theorem 10. Let M^* be a maximum clique of $G(P)$. As $K \geq n^{6/7}$, the probability that $p^* \in M^*$ is at least $1/n^{1/7}$. We repeat the same process for $cn^{1/7} \log n$ times for a sufficiently large constant c, and return the largest clique we find. The probability that at least one selected point belongs to M^* is at least

$$1 - \left(1 - \frac{1}{n^{1/7}}\right)^{cn^{1/7} \log n} \geq 1 - e^{-c \log n}.$$

Hence, with high probability, one of the $O(n^{1/7} \log n)$ randomly selected points is in M^* and therefore the algorithm of Theorem 10 when applied to the point will compute a maximum clique of $G(P)$. As such, with high probability, our algorithm can find a maximum clique. The runtime of the algorithm is bounded by $O(n^2 \log n \cdot n^{1/7} \log n)$ in the worst case, which is $O(n^{15/7} \log^2 n)$.

Combining the above, the time complexity of the overall algorithm is $O(n^{15/7+o(1)})$ in the worst case. The following theorem summarizes the result.

Theorem 12 *Given a set P of n points in convex position in the plane, we can find a clique in the unit-disk graph $G(P)$ in $O(n^{15/7+o(1)})$ time, and with high probability the clique is a maximum clique of $G(P)$.*

References

[1] Alok Aggarwal, Leonidas J. Guibas, James Saxe, and Peter W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete and Computational Geometry*, 4:591–604, 1989. doi:10.1007/BF02187749.

[2] Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. *Journal of Algorithms*, 12(1):38–56, 1991. doi:10.1016/0196-6774(91)90022-Q.

[3] Christoph Ambühl and Uli Wagner. The clique problem in intersection graphs of ellipses and triangles. *Theory of Computing Systems*, 38:279–292, 2005. doi:10.1007/s00224-005-1141-6.

[4] Alberto Apostolico and Susanne E. Hambrusch. Finding maximum cliques on circular-arc graphs. *Information Processing Letters*, 26(4):209–215, 1987. doi:10.1016/0020-0190(87)90007-X.

[5] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011. doi:DOI:10.1287/opre.1100.0851.

[6] Paul Balister, Béla Bollobás, Amites Sarkar, and Mark Walters. Connectivity of random k-nearest-neighbour graphs. *Advances in Applied Probability*, 37(1):1–24, 2005. doi:doi:10.1239/aap/1113402397.

[7] Édouard Bonnet, Nicolas Grelier, and Tillmann Miltzow. Maximum clique in disk-like intersection graphs. In *Proceedings of the 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 17:1–17:18, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.17.

[8] Sergio Cabello, Jean Cardinal, and Stefan Langerman. The clique problem in ray intersection graphs. *Discrete Computational Geometry*, 50:771–783, 2013. doi:10.1007/s00454-013-9538-5.

[9] Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016. doi:10.4230/LIPIcs.ISAAC.2016.24.

[10] Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, Micha Sharir, and Emo Welzl. Improved bounds on weak ϵ-nets for convex sets. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, page 495–504, 1993. doi:10.1145/167088.167222.

[11] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *Proceedings of the 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022. doi:10.1109/FOCS54457.2022.00064.

[12] Jongmin Choi, Jaegun Lee, and Hee-Kap Ahn. Efficient k-center algorithms for planar points in convex position. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS)*, pages 262–274, 2023. doi:10.1007/978-3-031-38906-1_18.

[13] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86:165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

[14] David Eppstein and Jeff Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete and Computational Geometry*, 11(1):321–350, 2007. doi:10.1007/BF02574012.

[15] Jared Espenant, J. Mark Keil, and Debajyoti Mondal. Finding a maximum clique in a disk graph. In *Proceedings of the 39th International Symposium on Computational Geometry (SoCG)*, pages 30:1–30:17, 2023. doi:10.4230/LIPIcs.SoCG.2023.30.

[16] Tuvi Etzion and Patric R. J. Ostergard. Greedy and heuristic algorithms for codes and colorings. *IEEE Transactions on Information Theory*, 44(1):382–388, 1998. doi:10.1109/18.651069.

[17] Tomas Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51(2):261–272, 1995. doi:10.1006/jcss.1995.1065.

[18] John Hershberger and Subhash Suri. Finding tailored partitions. *Journal of Algorithms*, 3:431–463, 1991. doi:10.1145/73833.73862.

[19] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

[20] Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an

intersection graph of rectangles in the plane. *Journal of Algorithms*, 4:310–323, 1983. doi:10.1016/0196-6774(83)90012-3.

[21] Hiroshi Imai and Tetsuo Asano. Efficient algorithms for geometric graph search problems. *SIAM Journal on Computing*, 15(2):478–494, 1986. doi:10.1137/0215033.

[22] Matthew J. Katz and Micha Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997. doi:10.1137/S0097539794268649.

[23] J. Mark Keil and Debajyoti Mondal. The maximum clique problem in a disk graph made easy. In *Proceedings of the 41st International Symposium on Computational Geometry (SoCG)*, page to appear, 2025. URL: https://arxiv.org/abs/2404.03751.

[24] Andrzej Lingas. On approximation behavior and implementation of the greedy triangulation for convex planar point sets. In *Proceedings of the 2nd Annual Symposium on Computational Geometry (SoCG)*, page 72–79, 1986. doi:10.1145/10515.10523.

[25] Noël Malod-Dognin, Rumen Andonov, and Nicola Yanev. Maximum cliques in protein structure comparison. In *Proceedings of International Symposium on Experimental Algorithms*, pages 106–117, 2010. doi:10.1007/978-3-642-13193-6_10.

[26] Nicholas Nash and David Gregg. An output sensitive algorithm for computing a maximum independent set of a circle graph. *Information Processing Letters*, 110(16):630–634, 2010. doi:10.1016/j.ipl.2010.05.016.

[27] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, pages 234–244, 1994. doi:10.1145/190809.190336.

[28] Charles E. Perkins and Pravin Bhagwat. Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 90–100, 1999. doi:10.1109/MCSA.1999.749281.

[29] Dana S. Richards and Jeffrey S. Salowe. A rectilinear steiner minimal tree algorithm for convex point sets. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 201–212, 1990. doi:10.1007/3-540-52846-6_90.

[30] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. doi:10.1016/j.cosrev.2007.05.001.

[31] Vishwanath R. Singireddy, Manjanna Basappa, and Joseph S.B. Mitchell. Algorithms for k-dispersion for points in convex position in the plane. In *Proceedings of the 9th International Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*, pages 59–70, 2023. doi:10.1007/978-3-031-25211-2_5.

[32] Anastasiia Tkachenko and Haitao Wang. Dominating set, independent set, discrete k-center, dispersion, and related problems for planar points in convex position. In *Proceedings of the 42nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 73:1–73:20, 2025. doi:10.4230/LIPIcs.STACS.2025.73.

[33] Haibo Wang, Bahram Alidaee, Fred Glover, and Gary Kochenberger. Solving group technology problems via clique partitioning. *International Journal of Flexible Manufacturing Systems*, 18:77–97, 2007. doi:10.1007/s10696-006-9011-3.

[34] Haitao Wang. Unit-disk range searching and applications. *Journal of Computational Geometry*, 14(1):343–394, 2023. doi:10.20382/jocg.v14i1a13.

[35] Haitao Wang and Jie Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete and Computational Geometry*, 64:1141–1166, 2020. doi:10.1007/s00454-020-00219-7.

[36] Haitao Wang and Yiming Zhao. Improved algorithms for distance selection and related problems. In *Proceedings of the 31st Annual European Symposium on Algorithms (ESA)*, pages 101:1–101:14, 2023. doi:10.4230/LIPIcs.ESA.2023.101.

[37] Haitao Wang and Yiming Zhao. Reverse shortest path problem for unit-disk graphs. *Journal of Computational Geometry*, 14:14–47, 2023. doi:10.20382/jocg.v14i1a2.

[38] Haitao Wang and Yiming Zhao. Dynamic unit-disk range reporting. In *Proceedings of the 42nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 70:1–70:19, 2025. doi:10.4230/LIPIcs.STACS.2025.76.

[39] Oliver Weide, David Ryan, and Matthias Ehrgott. An iterative approach to robust and integrated aircraft routing and crew scheduling. *Computers & Operations Research*, 37(5):833–844, 2010. doi:10.1016/j.cor.2009.03.024.

[40] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015. doi:10.1016/j.ejor.2014.09.064.

[41] Qinghua Wu, Jin-Kao Hao, and Fred Glover. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196(1):611–634, 2012. doi:10.1007/s10479-012-1124-3.

[42] Onur Çağırıcı and Bodhayan Roy. Maximum clique of disks in convex position. In *Abstracts of Computational Geometry: Young Researchers Forum (CG:YRF)*, pages 21:1–21:3, 2018. URL: https://www.computational-geometry.org/old/YRF/cgyrf2018.pdf.

Approximation and Hardness of Polychromatic TSP

Thomas Schibler* Subhash Suri* Jie Xue†

Abstract

We introduce the *Polychromatic Traveling Salesman Problem* (PCTSP), where the input is an edge weighted graph whose vertices are partitioned into k equal-sized color classes, and the goal is to find a minimum-length Hamiltonian cycle that visits the classes in a fixed cyclic order. This generalizes the Bipartite TSP (when $k = 2$) and the classical TSP (when $k = n$). We give a polynomial-time $(3-2\cdot 10^{-36})$-approximation algorithm for metric PCTSP. Complementing this, we show that Euclidean PCTSP is APX-hard even in $\mathbb{R}^2$, ruling out the existence of a PTAS unless P = NP.

1 Introduction

The classic Traveling Salesman Problem (TSP) takes an edge-weighted graph G as input and aims to find a minimum-weight Hamiltonian cycle in G (which by definition is a simple cycle that visits all vertices of G). As one of the most fundamental algorithmic problems, TSP has stood as a testing ground for algorithmic developments in the theory community since its introduction as one of Karp's original NP-Complete problems. In 1976, Christofides showed a 1.5-approximation for any metric graph G, and this bound resisted progress for decades until Karlin et al. [6] improved the bound (slightly) to $1.5 - 10^{-36}$, which remains the best known. In the case that G contains only edge weights 1 and 2, Papadimitriou and Yannakakis obtained a better approximation ratio of 7/6, and showed that even this restricted case is APX-Hard, and thus does not admit a PTAS unless P=NP [8]. However, if the edge weights correspond to the Euclidean distances between points in some fixed $\mathbb{R}^d$, then a PTAS is known from the celebrated work of Arora [2] and Mitchell.

In this paper, we consider a colored version of TSP, called *polychromatic TSP* (PCTSP), in which the vertices of G are partitioned into color classes of the same size and the goal is to find a TSP tour that repeatedly visits the color classes in a consistent order. Formally, a *polychromatic graph* consists of a graph G together with a partition $\mathcal{P} = \{V_1, \cdots, V_k\}$ of $V(G)$ into subsets (called *color classes*) of equal size (i.e., $|V_1| = \cdots = |V_k|$) and a weight function $w : E(G) \to \mathbb{R}_{\geq 0}$. Let $(G, \mathcal{P}, w)$ be a polychromatic graph with $|V(G)| = n$ and $|\mathcal{P}| = k$. The *weight* of a path/cycle π in G, denoted by $w(\pi)$, is the sum of the weights of the edges in π under the weight function w. For a permutation $\sigma = (\sigma(0), \sigma(1), \ldots, \sigma(k-1))$ of $[k] = \{1, \cdots, k\}$, we say a cycle C in G is a *σ-cycle* if it can be written as $C = (v_0, v_1, \cdots, v_r)$ where $v_r = v_0$ such that $v_i \in V_{\sigma(i \bmod k)}$ for all $i \in \{0\} \cup [r]$; the weight of C is equal to $\sum_{i=1}^{n} w((v_{i-1}, v_i))$. Note that the length of a σ-cycle is always a multiple of k. Intuitively, a σ-cycle repeatedly visits the color classes of G in the order specified by σ. The goal of PCTSP is to find a minimum-weight Hamiltonian cycle in G that is a σ-cycle for some permutation σ of $[k]$. Clearly, when $k = n$, PCTSP is exactly the classic TSP. As such, PCTSP is a generalization of TSP and is thus NP-hard even when G is a metric graph. The main focus of this paper is to study the approximability of PCTSP in metric graphs, and more specifically, Euclidean graphs in a fixed dimension.

We remark that PCTSP is a natural extension of Bipartite TSP, where a tour must alternate between red and blue nodes. This problem has long been studied in the robotics literature as an appropriate model for "pick and place" route planning [1, 4, 10]. (In this setting, a robot with unit capacity must move objects from a set of sources to a set of destinations, necessitating a route that alternates between the two node types.) For metric graphs, Anily and Hassin gave a 2.5-approximation [1], and Chalasani et al. [4] later improved the ratio to 2.

Lastly, many other lines of research consider chromatic variants of the TSP. For example, Dross et al. [5] give a Gap-ETH tight approximation scheme for the bicolored noncrossing Euclidean TSP, where the goal is to find a separate tour for the points of each color, such that the two do not intersect in $\mathbb{R}^2$. Baligács et al. give a constant factor approximation for the 3-color version [3]. Other settings involve multiple salesman that can only visit given subsets of the colors, or seek the shortest cycle that visits each color once. In contrast, we aim for a single polychromatic tour that visits every node of G while cycling through a large number of colors.

1.1 Main results

We initiate the study of metric PCTSP and Euclidean PCTSP in a fixed dimension. Our first result is a polynomial-time constant-approximation algorithm for

*University of California, Santa Barbara, CA, USA, {tschibler,suri}@ucsb.edu

†New York University Shanghai, China, jiexue@nyu.edu

PCTSP in metric graphs.

Theorem 1 *There is a polynomial-time* $(3-2\cdot 10^{-36})$-*approximation algorithm for metric PCTSP.*

To complement the above algorithmic result, we rule out the existence of (Q)PTASes for metric PCTSP and even Euclidean PCTSP, by proving the following APX-hardness result.

Theorem 2 *Euclidean PCTSP in* $\mathbb{R}^2$ *does not admit a PTAS unless P=NP. In particular, Euclidean PCTSP is APX-Hard.*

Organization. The rest of the paper is organized as follows. In Section 2, we give the basic notions and preliminaries that are required for our results. Section 3 presents our approximation algorithm for metric PCTSP, and Section 4 presents our hardness result for Euclidean PCTSP.

2 Preliminaries

For a graph G, we use $V(G)$ and $E(G)$ to denote the vertex set and edge set of G, respectively. For $V \subseteq V(G)$, we denote by $G[V]$ the graph induced in G by V. Let $(G, \mathcal{P}, w)$ be a polychromatic graph where $\mathcal{P} = \{V_1, \dots, V_k\}$. We call a cycle in G a *polychromatic cycle* in $(G, \mathcal{P}, w)$ if it is a σ-cycle for some permutation of $[k]$. We denote by $\mathcal{H}_G$ the set of all Hamiltonian σ-cycles of G: $\mathcal{H}_G = \{\pi : \pi \text{ is a polychromatic cycle and } V(\pi) = V(G)\}$.

In metric PCTSP, we are given a polychromatic graph $(G, \mathcal{P}, w)$ such that G is complete and $w((v, v')) \leq w((v, v'')) + w((v'', v'))$ for all $v, v', v'' \in V(G)$. We are to compute $\arg\min_{\pi \in \mathcal{H}_G} w(\pi)$. In Euclidean PCTSP, the input is $(P, \mathcal{P})$ for a set of points $P \subset \mathbb{R}^d$, and partition of the points $\mathcal{P} = \{P_1, \cdots, P_k\}$. The task is to solve metric PCTSP on input $(G, \mathcal{P}', w)$, where G is the complete graph with vertices $V(G) = \{v_p : p \in P\}$, $\mathcal{P}' = \{P'_1, \cdots, P'_k\}$ s.t. $P'_i = \{v_p : p \in P_i\}$ for all $1 \leq i \leq k$, and $w((v_p, v_{p'})) = ||p - p'||_2$. $||\cdot||_2$ denotes the Euclidean norm. For convenience, we similarly refer to a cycle of the points $\pi = (p_0, \cdots, p_r)$ where $p_r = p_0$ as a σ-cycle if $p_i \in P_{\sigma(i \bmod k)}$ for all $i \in \{0\} \cup [r]$. The length of π is $||\pi|| = \sum_{i \in [r]} ||p_i - p_{i+1}||_2$. Finally, we use the term σ-tour to mean a Hamiltonian σ-cycle.

3 Constant approximation for metric PCTSP

In this section, we give our polynomial-time constant-approximation algorithm for metric PCTSP. Consider an instance $(G, \mathcal{P}, w)$ of metric PCTSP, where $|V(G)| = n$ and $\mathcal{P} = \{V_1, \dots, V_k\}$.

At a high level, we solve the problem via two steps. First, we give an algorithm that can compute in polynomial time, for any given permutation σ of $[k]$, an $O(1)$-approximation for the optimal σ-tour. When $k = O(1)$, this already solves metric PCTSP since we can simply run the algorithm on all $k!$ possibilities of σ. However, we do not have the assumption that k is a constant in PCTSP. Therefore, in the second step, we show how to approximate the optimal ordering σ in polynomial time.

3.1 Computing the tour for a fixed order

We first consider how to compute a good σ-tour in G for a given ordering σ of $[k]$. Without loss of generality, we can assume that $\sigma = (1, \dots, k)$. For convenience, define $V_{k+1} = V_1$. Our algorithm is presented in Algorithm 1. The sub-routine $\textsc{MinMatching}(G, w, V_i, V_{i+1})$ in line 2 computes a minimum-weight perfect matching M_i between V_i and V_{i+1} under the weight function w, which must exist because $|V_i| = |V_{i+1}|$ and G is a complete graph. Define H as the subgraph of G consisting of the edges in $\bigcup_{i=1}^{k} M_i$ (line 3), and let $\mathcal{C}$ be the set of connected components of H (line 4). Note that each $C \in \mathcal{C}$ is a σ-cycle in G. Next, we arbitrarily pick a vertex $v_C \in V(C) \cap V_1$ in each $C \in \mathcal{C}$ (line 6) and let $V'_1 \subseteq V_1$ be the set of these vertices (line 7). We then call the sub-routine $\textsc{ApproxTSP}(G[V'_1], w)$ in line 8, which computes a $(1.5 - 10^{-36})$-approximation TSP T of the subgraph $G[V'_1]$ under the weight function w, which can be done using the algorithm of Karlin et al. [6]. Finally, we "glue" the cycles in $\mathcal{C}$ along T as follows to obtain a Hamiltonian σ-cycle S (line 9). Write $\mathcal{C} = \{C_1, \dots, C_t\}$ such that $T = (v_{C_1}, \dots, v_{C_k}, v_{C_1})$. For $i \in [t]$, let u_{C_i} be the neighbor of v_{C_i} in C_t that belongs to V_k. Then we take the disjoint σ-cycles $C_1, \dots, C_t$, remove the edges $(u_{C_1}, v_{C_1}), \dots, (u_{C_k}, v_{C_k})$, and add the edges $(u_{C_1}, v_{C_2}), \dots, (u_{C_{k-1}}, v_{C_k}), (u_{C_k}, v_{C_1})$. This results in a Hamiltonian cycle S in G, and one can easily verify that S is a σ-cycle. We return S as the output of our algorithm (line 10).

Algorithm 1 $\textsc{FixedOrderTour}(G, \{V_1, \cdots, V_k\}, w)$

```
 1: for every $i \in [k]$ do
 2:     $M_i \leftarrow \textsc{MinMatching}(G, w, V_i, V_{i+1})$
 3: $H \leftarrow (V(G), \bigcup_{i=1}^{k} M_i)$
 4: $\mathcal{C} \leftarrow$ set of connected components of $H$
 5: for every $C \in \mathcal{C}$ do
 6:     pick an arbitrary vertex $v_C \in V(C) \cap V_1$
 7: $V'_1 \leftarrow \{v_C : C \in \mathcal{C}\}$
 8: $T \leftarrow \textsc{ApproxTSP}(G[V'_1], w)$
 9: $S \leftarrow \textsc{Glue}(T, \mathcal{C})$
10: return $S$
```

In the rest of this section, we show that S is a $(2.5 - 10^{-36})$-approximation for the optimal σ-tour of G.

Lemma 3 $\sum_{i=1}^{k}\sum_{e\in M_i} w(e) \le \sum_{e\in E(R)} w(e)$ *for any Hamiltonian σ-cycle R in G.*

Proof. Note that the edges of R between V_i and V_{i+1} form a perfect matching between V_i and V_{i+1}, for $i \in [k]$. Since M_i is a minimum-cost perfect matching between V_i and V_{i+1}, we have $\sum_{i=1}^{k}\sum_{e\in M_i} w(e) \le \sum_{e\in E(R)} w(e)$. □

Lemma 4 $\sum_{e\in E(S)} w(e) \le \sum_{i=1}^{k}\sum_{e\in M_i} w(e) + (1.5 - 10^{-36})\cdot \mathsf{tsp}(G)$*, where $\mathsf{tsp}(G)$ is the minimum weight of a Hamiltonian cycle of G.*

Proof. For convenience, let us write $C_{t+1} = C_1$. We observe that $\sum_{e\in E(S)} w(e) = \sum_{i=1}^{k}\sum_{e\in M_i} w(e) + \sum_{i=1}^{t}(w((u_{C_i}, v_{C_{i+1}})) - w((u_{C_i}, v_{C_i})))$. Since G is a metric graph, we have $w((u_{C_i}, v_{C_{i+1}})) - w((u_{C_i}, v_{C_i})) \le w((v_{C_i}, v_{C_{i+1}}))$. It follows that $\sum_{i=1}^{t}(w((u_{C_i}, v_{C_{i+1}})) - w((u_{C_i}, v_{C_i}))) \le \sum_{e\in E(T)} w(e)$. Recall that T is a $(1.5 - 10^{-36})$-approximation TSP T of $G[V_1']$. As $G[V_1']$ is an induced subgraph of G and G is a metric graph, we have $\sum_{e\in E(T)} w(e) \le (1.5 - 10^{-36})\mathsf{tsp}(G)$. □

Since $\mathsf{tsp}(G) \le \sum_{e\in E(R)} w(e)$ for any Hamiltonian σ-cycle R in G, the above lemmas imply that $\sum_{e\in E(S)} w(e) \le (2.5 - 10^{-36}) \sum_{e\in E(R)} w(e)$ for any Hamiltonian σ-cycle R in G.

3.2 Solving metric PCTSP

In this section, we show how to compute a good ordering σ. Together with the algorithm in the previous section, this shall give us the final algorithm for Theorem 1.

For $i, j \in [k]$ with $i \neq j$, we compute a minimum-cost perfect matching $M_{i,j}$ between V_i and V_j in G. Let Σ_k denote the set of all permutations of $[k]$. For $\sigma = (r_1, \ldots, r_k) \in \Sigma_k$, let H_σ denote the subgraph of G consisting of the edges in $\bigcup_{i=1}^{k} M_{r_i, r_{i+1}}$; here we set $r_{k+1} = r_1$ for convenience. Define $w_\sigma = \sum_{e\in E(H_\sigma)} w(e)$.

Suppose opt is the optimum of the PCTSP instance, i.e., the weight of an optimal solution. Note that $\mathsf{opt} \ge \mathsf{tsp}(G)$. By Lemma 3, we have $\mathsf{opt} \ge \min_{\sigma\in\Sigma_k} w_\sigma$. If $\hat{\sigma} \in \Sigma_k$ satisfies that $w_{\hat{\sigma}} \le c \cdot \min_{\sigma\in\Sigma_k} w_\sigma$, then Lemma 4 further implies that when running the algorithm in the previous section on $\hat{\sigma}$, the output tour S satisfies $\sum_{e\in E(S)} w(e) \le (c + 1.5 - 10^{-36}) \cdot \mathsf{opt}$. Based on this observation, it suffices to find $\hat{\sigma} \in \Sigma_k$ that (approximately) minimizes $w_{\hat{\sigma}}$.

We achieve this goal as follows. Build a complete graph G' with $V(G') = \{V_1, \ldots, V_k\}$, and define a weight function $w' : E(G') \to \mathbb{R}_{\ge 0}$ by setting $w'((V_i, V_j)) = \sum_{e\in E(M_{i,j})} w(e)$. One can easily check that G' is also a metric graph. Observe that for each $\sigma = (r_1, \ldots, r_k) \in \Sigma_k$, w_σ is just equal to the weight of the Hamiltonian cycle $(V_{r_1}, \ldots, V_{r_k}, V_{r_1})$ in G' under w'. As such, we simply compute a $(1.5 - 10^{-36})$-approximation TSP $(V_{\hat{r}_1}, \ldots, V_{\hat{r}_k}, V_{\hat{r}_1})$ in G'. Then $\hat{\sigma} = (\hat{r}_1, \ldots, \hat{r}_k)$ satisfies that $w_{\hat{\sigma}} \le (1.5 - 10^{-36}) \cdot \min_{\sigma\in\Sigma_k} w_\sigma$. Therefore, if we apply the algorithm in the previous section on $\hat{\sigma}$, the output tour S satisfies $\sum_{e\in E(S)} w(e) \le (3 - 10^{-36}) \cdot \mathsf{opt}$. This completes the proof of Theorem 1.

Theorem 1 *There is a polynomial-time $(3 - 2\cdot 10^{-36})$-approximation algorithm for metric PCTSP.*

Remark. As one can verify from our algorithm, the approximation factor and the running time of the algorithm in Theorem 1 in fact depend on the best matching algorithm and metric TSP algorithm. Specifically, let $M(n)$ be the time for computing a minimum-cost bipartite matching in a complete bipartite graph with n vertices and $T(n)$ be the time for computing an α-approximate TSP in a metric graph with n vertices. Then we get a 2α-approximation algorithm for PCTSP with running time $O(k^2 M(2n/k) + T(k) + T(n/k))$.

4 APX-Hardness for Euclidean PCTSP in $\mathbb{R}^2$

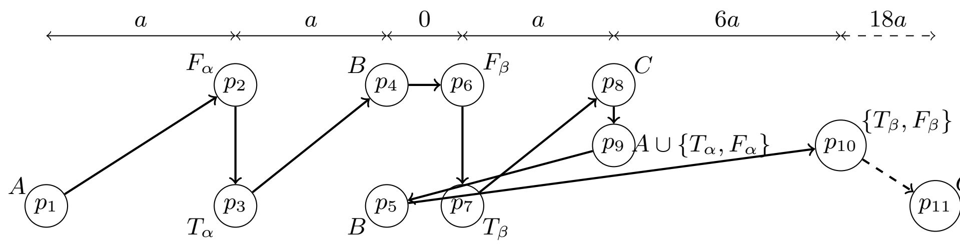
In this section, we prove Theorem 2 by a reduction from Max 2-SAT, which is known to be APX-hard [7]. The input to Max 2-SAT is a set of n variables $\{x_1, \cdots, x_n\}$ and m clauses $\{c_1, \cdots, c_m\}$. Each clause is a conjunction of two literals; a literal is a variable x_i or its negation $\bar{x}_i$. The task is to find a boolean assignment of the variables that satisfies the maximum number of clauses.

The general idea behind our reduction is to encode the truth assignment of SAT variables in the choice of permutation σ. We then test a corresponding tour with a series of clause gadgets; informally, if σ fails to "satisfy" a clause, then the clause gadget will penalize the tour by a fixed small amount. The structure of the proof is as follows. We first describe how to encode a truth assignment using σ. We then introduce our clause gadget and reason about its structure in Lemma 5. Finally, we build the full construction using a series of clause gadgets and an auxiliary set of points S. The points of S serve two purposes: (1) they rule out the possibility of σ that does not properly encode a truth assignment, and (2) they divert the return trip of the tour away from the clause gadgets so that we may reason about a single pass through our gadgets.

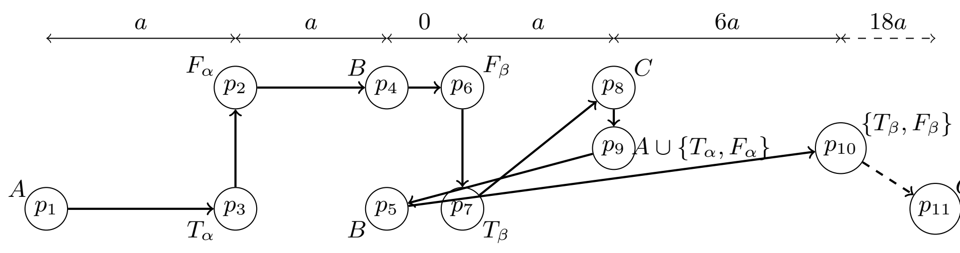
Our construction uses $k = 3n + 1$ color classes (point sets), which we will label R_α, T_α, F_α for $\alpha \in [1, n]$, and the additional R_{n+1}. For a permutation σ, let $\prec_\sigma$ be the total order of the classes under σ, i.e. $V_{\sigma(0)} \prec_\sigma V_{\sigma(1)} \cdots \prec_\sigma V_{\sigma(k-1)}$. We say that a permutation σ is *valid* if it satisfies:

1. $R_\alpha \prec_\sigma T_\alpha$, $R_\alpha \prec_\sigma F_\alpha$, for all $1 \le \alpha \le n$ and

2. $T_\alpha \prec_\sigma R_\beta$, $F_\alpha \prec_\sigma R_\beta$, for all $1 \le \alpha < \beta \le n + 1$.



(a) Optimal σ-path visiting C_i when σ does not satisfy the clause, i.e. $F_\alpha \prec_\sigma T_\alpha$, $F_\beta \prec_\sigma T_\beta$



(b) Optimal σ-path visiting C_i where σ satisfies x_α but not x_β, i.e. $T_\alpha \prec_\sigma F_\alpha$, $F_\beta \prec_\sigma T_\beta$

Figure 1: The clause gadget C_i for clause $c_i = x_\alpha \vee x_\beta$ showing two possible σ-paths. The gadget consists of $6m+2$ points (two of each color) placed at 9 distinct locations. (Notice that p_4, p_6 and p_5, p_7 coincide, but we distinguish them to make it clear when a path revisits a location.) At an intuitive level, the gadget tests whether σ satisfies x_α (i.e. that $T_\alpha \prec_\sigma F_\alpha$) in the triangle (p_1, p_2, p_3). Observe that path (a) does not satisfy x_α, and must take the longer leg (p_1, p_2), while path (b) takes the shorter leg (p_2, p_3) by satisfying x_α. Similarly, triangle (p_6, p_7, p_8) tests x_β. However, between these two tests, the tour has the choice of which set B to visit, either at p_4 or p_5, such that satisfying exactly one of x_α or x_β allows the tour to take an edge of rectangle (p_2, p_3, p_5, p_4) while satisfying both (or neither) of x_α, x_β forces the path to take a diagonal. This avoids over-rewarding σ for satisfying both x_α and x_β, and is the main idea behind the gadget.

Observe that there are exactly 2^n valid permutations that naturally correspond to truth assignments of the variables $\{x_1, \cdots, x_n\}$. For each $\alpha \in [1, n]$, we only have the freedom to choose whether $T_\alpha \prec_\sigma F_\alpha$ or $F_\alpha \prec_\sigma T_\alpha$. We say that σ *satisfies* the literal x_α if $T_\alpha \prec_\sigma F_\alpha$; likewise σ satisfies $\bar{x}_\alpha$ if $F_\alpha \prec_\sigma T_\alpha$. σ satisfies a clause c_i if it satisfies either of the literals of c_i.

4.1 Clause gadget

The crux of our reduction lies in the creation of a clause gadget that tests σ on whether it satisfies a given clause. This test can only depend on σ's encoding of the two variables in that clause, so the gadget will need to allow the tour to "jump through" all of the colors that precede $\{T_\alpha, F_\alpha\}$, that fall between $\{T_\alpha, F_\alpha\}$, $\{T_\beta, F_\beta\}$, and that come after $\{T_\beta, F_\beta\}$, respectively. Formally, we define these color sets for use in the gadget, for all α, β, s.t. $1 \leq \alpha \leq \beta \leq n$:

- $A_{\alpha,\beta} = \{R_{\alpha'}, T_{\alpha'}, F_{\alpha'} | 0 < \alpha' < \alpha\} \cup \{R_\alpha\}$
- $B_{\alpha,\beta} = \{R_{\alpha'}, T_{\alpha'}, F_{\alpha'} | \alpha < \alpha' < \beta\} \cup \{R_\beta\}$
- $C_{\alpha,\beta} = \{R_{\alpha'}, T_{\alpha'}, F_{\alpha'} | \beta < \alpha' < n+1\} \cup \{R_{n+1}\}$

When the indices α and β are clear, we abbreviate the sets as A, B, C. These sets are nonempty, and any valid σ satisfies (by definition):

$$A \prec_\sigma \{T_\alpha, F_\alpha\}, \prec_\sigma B \prec_\sigma \{T_\beta, F_\beta\} \prec_\sigma C \qquad (1)$$

Figure 1 shows the gadget, along with (optimal) example paths for when σ does, and does not satisfy the clause. We describe the position of the points of the gadget (and later the spacing between gadgets) in terms of constants a and b, with the intuition that $b \gg a \gg 1$. Let $x > 0$. A clause gadget C_i for the clause $c_i = x_\alpha \vee x_\beta$ anchored at x is the following collection of points.[1] (When we refer to a set of colors X at a location, we mean a set of $|X|$ coincident points at that location, one of each color in the set X.)

- Set A at $(x, -1)$
- Points of color F_α, T_α at $(x+a, 1), (x+a, -1)$, resp.

[1]For clauses that include negative literals (e.g. $\bar{x}_\alpha$ instead of x_α), we swap the corresponding colors (T_α and F_α) in the gadget construction. The key is that the satisfying color appears on the line $y = -1$, and the other on the line $y = 1$. Without loss of generality, we will assume these colors are labeled as T_α and T_β in our analysis.

- Two sets B at $(x+2a, \pm 1)$
- Points of F_β, T_β at $(x+2a, 1), (x+2a, -1)$, resp.
- Set C at $(x+3a, 1)$
- Set $A \cup \{T_\alpha, F_\alpha\}$ at $(x+3a, 0)$
- Set $\{T_\alpha, F_\alpha\}$ at $(x+9a, 0)$
- Set C at $(x+27a, -1)$

Let us first establish the length of the optimal path through the gadget. To do so, we need to clarify what it means for a path to be polychromatic. A path is a *σ-path* if it can be written as $(v_1, \cdots, v_{r-1})$ such that for some j, $v_i \in V_{\sigma(i+j \mod k)}$ for all $i \in [r]$. Note that a σ-path may start at a point of any color. We will also need the following definitions. Let a permutation σ be *α, β-valid* if equation 1 holds for σ, α, β. Two permutations σ, σ' *agree* on x_α if $T_\alpha \prec_\sigma F_\alpha$ iff $T_\alpha \prec_{\sigma'} F_\alpha$.

Lemma 5 *Let C_i be the clause gadget for $c_i = x_\alpha \vee x_\beta$. For any valid σ, let T_i^* be the shortest σ-path that visits all points of C_i. Then,*

$$\|T_i^*\| = \begin{cases} c + 2a & \text{if } \sigma \text{ satisfies } c_i \\ c + 2\sqrt{a^2+4} & \text{otherwise} \end{cases}$$

where $c = 5 + \sqrt{a^2+4} + \sqrt{a^2+1} + \sqrt{49a^2+1} + \sqrt{(18a)^2+1}$. Furthermore, there is no σ' and σ'-path shorter than T_i^* visiting C_i such that σ' is α, β-valid and agrees with σ on x_α, x_β.

Proof. Due to space constraints, the proof is included in the full version. [9] □

4.2 Full construction

We now describe the full construction, depicted in Figure 2. Given a Max 2-SAT instance $\Pi = (\{x_1, \cdots, x_n\}, \{c_1, \cdots, c_m\})$ we create a set of points $P_\Pi = S \cup C$. S will be a special set containing exactly one point from each color class. C is a collection of clause gadgets $C_1 \cup \cdots \cup C_m$. For each clause c_i, C_i is anchored at $(i-1)(9a+b)$. Fix W as the maximum x-coordinate of a point in C, i.e. $W = 27ma + (m-1)b$.

We place the points of S along the horizontal line $y = -(2W+1)$ from East to West following a valid permutation. Points of the colors T_i, F_i are coincident for each i, so that *any* valid tour may visit S via a straight line, while any invalid tour will have to backtrack. The spacing around the points is proportional to the number of clauses that x_i appears in. The idea behind this spacing is to only penalize invalid tours relative to the number of clause gadgets they could "cheat" in (precisely those for which they are not α, β-valid), since we cannot make the total spacing too large while preserving approximate solutions. Specifically, for each variable x_i, let n_i be the number of clauses that it or its negation appears in. Clearly $\sum_{i=1}^n n_i = 2m$, since each clause contains exactly two literals. Let $N_i = \sum_{j=1}^i n_j$. S contains the following points for $(1 \le i \le n)$, where $l = \frac{27}{4}a + (\frac{1}{4} - \frac{1}{m})b$:

- Point r_i of color R_i at $(W - 2lN_{i-1}, -2W-1)$
- Points t_i, f_i of colors T_i, F_i resp. both located at $(W - 2lN_{i-1} - ln_i, -2W-1)$

Finally, S contains a point r_{n+1} of color R_{n+1} at $(0, -2W-1) = (W - 2lN_n, -2W-1)$. We can now state the length of the optimal σ-tour for any valid σ.

Lemma 6 *Let σ be a valid permutation satisfying k clauses of Π. Then there is a σ-tour T^* of $P_\pi = S \cup C$ s.t.*

$$\|T^*\| = 5W + (m-1)b + k(c+2a) + (m-k)(c + 2\sqrt{a^2+4})$$

where $c = f(a)$ is the constant defined in lemma 5.

Proof. We construct a tour T^* that first visits the points of S via a straight line and then visits each of the clauses in ascending order $C_1, \cdots, C_m$, using the σ-path T_i^* of Lemma 5 to visit each C_i. First, write $S = \{s_1, \cdots, s_{3n+1}\}$ s.t. $s_1 \prec_\sigma \cdots \prec_\sigma s_{3n+1}$, and let $T_S = (s_1, \cdots, s_{3n+1})$ be the unique σ-path visiting S. By construction, $||T_S|| = W$ for any valid σ. Now, let u_i, v_i be the first and last points of T_i^*, respectively. Then T^* consists of the edges in T_S, the edges in $T_1^*, \cdots, T_m^*$, and the additional edges $(s_{3n+1}, u_1), (v_m, s_1)$, and (v_i, u_{i+1}) for all $1 \le i < m$. By Lemma 1, $\sum_{i \in [1,m]} T_i^* = k(c+2a) + (m-k)(c + 2\sqrt{a^2+4})$. Furthermore, $||s_{3n+1} - u_1|| = ||v_m, s_1|| = 2W$, and $\sum_{i=1}^m ||v_i - u_{i+1}|| = (m-1)b$. Summing yields the desired total length. □

We now argue that the existence of a σ-cycle with length close to that of T^* in Lemma 6 implies that σ satisfies close to k clauses of Π. To do so, we will need to break apart an arbitrary tour for a finer-grained analysis. With this goal in mind, let T^σ be a σ-tour (for possibly invalid σ), and let $T^\sigma[S]$ and $T^\sigma[C]$ be the portions of T^σ induced on S and C respectively. It is worth pointing out that, in general, $T^\sigma[S]$ and $T^\sigma[C]$ may not be connected. However, we have chosen a suitable separation of distance $2W$ between S and C so that any tour that alternates between S and C multiples times has length that well exceeds that of T^*. Thus, we can safely ignore these tours, and assume that both $T^\sigma[S]$ and $T^\sigma[C]$ are connected.

Now, let us further decompose $T^\sigma[C]$: we will use T_i^σ to denote the portion of $T^\sigma[C]$ spent inside the clause

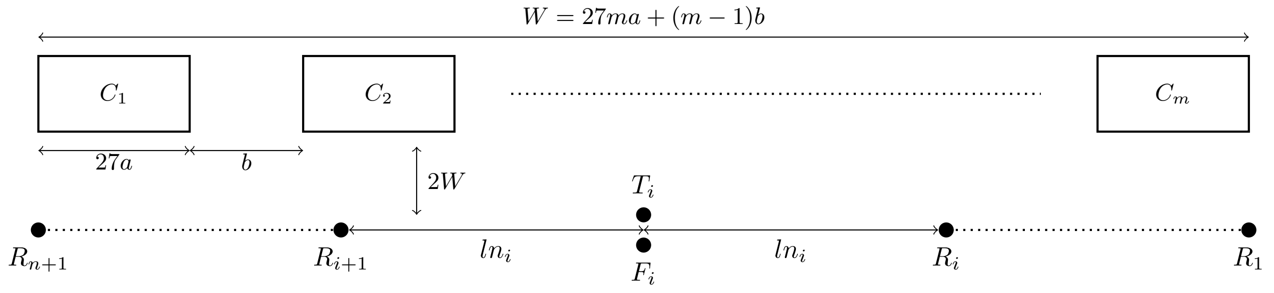


Figure 2: The instance $P_\Pi = S \cup C$. C (top) is the union of m clause gadgets $C_1 \cdots C_m$. S (bottom) contains exactly one point of each color placed along the line $y = -2W - 1$.

gadget C_i. More specifically, consider the set of vertical lines L given by the equations $x = (27a+b)(i-1)$ and $x = (27a+b)(i-1)+27a$ for $i \in [1,m]$, and let us subdivide each leg of $T^\sigma[C]$ at each crossing of a line in L. After subdivision, let T_i^σ be the set of line segments of $T^\sigma[C]$ contained (inclusively) between the vertical lines $x = (27a+b)(i-1)$ and $x = (27a+b)(i-1)+27a$. (T_i^σ may not be connected, and may include partial legs of $T^\sigma[C]$.) Collectively, the T_i^σ's do not account for the remaining portion of $T^\sigma[C]$ between gadgets, which has length at least $(m-1)b$. We get the following bound:

Observation 1 $\|T^\sigma\| > \sum_{i=1}^m \|T_i^\sigma\| + 5W + (m-1)b$.

If σ is valid, we can use Lemma 5 to bound the number of clauses σ satisfies based on the length of an approximate σ-tour. Lemma 8 will account for invalid σ.

Lemma 7 *Let T^σ be a σ-tour of P_Π for valid σ. Let $c = f(a)$ be the constant given in Lemma 5. If*

$$\|T^\sigma\| \le 5W+(m-1)b+k(c+2a)+(m-k)(c+2\sqrt{a^2+4}),$$

then σ satisfies at least k clauses of Π.

Proof. For the sake of contradiction, let us assume that σ satisfies k' clauses for some $k' < k$. If every T_i^σ is connected, then by Lemma 5,

$$\sum_{i=1}^m \|T_i^\sigma\| > k'(c+2a) + (m-k')(c+2\sqrt{a^2+4}).$$

For each disconnected T_i^σ, T^σ incurs an additional length b to enter and leave the gadget C_i. We charge this overhead to T_i^σ, such that the above bound always holds as long as $b >> a$, e.g. $b > 30a$. However, from Observation 1 and the assumption of the lemma, we have that

$$\sum_{i=1}^m \|T_i^\sigma\| < 2 + k(c+2a) + (m-k)(c+2\sqrt{a^2+4}),$$

which implies that $(k-k')(2\sqrt{a^2+4}-2a) < 0$, a contradiction since $k - k' > 0$. ☐

We lastly rule out the possibility of a σ-tour for invalid σ. Essentially, we are able to show that any gains made by "cheating" in the clause gadgets are offset by an equal increase in overhead to visit S. The proof is included in the full version. [9]

Lemma 8 *Let T^σ be a σ-tour of P_Π. There exists valid σ' and σ'-tour $T^{\sigma'}$ such that $\|T^{\sigma'}\| \le \|T^\sigma\|$. Furthermore, σ' can be computed in polynomial time given σ.*

We are ready to prove theorem 2, which we restate here.

Theorem 2 *Euclidean PCTSP in $\mathbb{R}^2$ does not admit a PTAS unless P=NP. In particular, Euclidean PCTSP is APX-Hard.*

Proof. Assuming a PTAS for Euclidean PCTSP, we can derive a PTAS for Max 2-SAT, which contradicts its hardness [7]. Given an instance to Max 2-SAT Π with optimal solution value k and target error ϵ, we first construct the instance P_Π of Figure 2 in polynomial time. Let $f(k)$ be the length of the tour in Lemma 6. For an appropriate ϵ', we apply the PTAS for EPCTSP to recover a σ-tour of P_Π that has length at most $(1+\epsilon')f(k)$, since OPT of P_Π is at most $f(k)$ by Lemma 6. We can assume σ is valid, else we can compute a valid σ' that admits a σ'-tour with equal or better cost in polynomial time by Lemma 8. By Lemma 7, σ satisfies at least $(1-\epsilon)k$ clauses of Π for a sufficiently small choice of constant $\epsilon' = f(\epsilon, a, b)$ (see full version for details [9]). ☐

5 Conclusion

In this paper, we introduced the Polychromatic TSP and studied its metric and Euclidean variants. In the metric case, we gave a constant factor approximation that remains polynomial time for any number of colors. We complemented this algorithmic result with the nonexistence of a PTAS, even for points in the plane as long as the number of colors is unbounded. An interesting open question remains: does Euclidean PCTSP admit a PTAS for a constant number of colors?

References

[1] Shoshana Anily and Refael Hassin. The swapping problem. *Networks*, 22(4):419–433, 1992.

[2] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, September 1998.

[3] J. Baligács, Y. Disser, A. E. Feldmann, and A. Zych-Pawlewicz. A $(5/3 + \epsilon)$-approximation for tricolored non-crossing euclidean tsp. In *32nd Annual European Symposium on Algorithms (ESA 2024)*, volume 308, pages 15:1–15:15, 2024.

[4] P. Chalasani, R. Motwani, and A. Rao. Algorithms for robot grasp and delivery. In *2nd International Workshop on Algorithmic Foundations of Robotics*, 1996.

[5] F. Dross, K. Fleszar, K. Węgrzycki, and A. Zych-Pawlewicz. Gap-eth-tight approximation schemes for red-green-blue separation and bicolored non-crossing euclidean travelling salesman tours. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1433–1463, 2023.

[6] A. R. Karlin, N. Klein, and S. O. Gharan. A (slightly) improved approximation algorithm for metric tsp. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 32–45. Association for Computing Machinery, 2021.

[7] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[8] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[9] T. Schibler, S. Suri, and J. Xue. Full version: Approximation and hardness of polychromatic tsp, 2025. https://arxiv.org/abs/2507.04974.

[10] A. Srivastav, H. Schroeter, and C. Michel. Approximation Algorithms for Pick-and-Place Robots. *Annals of Operations Research*, 107(1):321–338, October 2001.

Minimum Selective Subset on Some Graph Classes

Bubai Manna*

Abstract

In a connected simple graph $G = (V(G), E(G))$, each vertex is assigned a color from the set of colors $C = \{1, 2, \ldots, c\}$. The set of vertices is partitioned as $V(G) = \bigcup_{\ell=1}^{c} V_\ell$, where all vertices in V_ℓ share the same color ℓ. A subset $S \subseteq V(G)$ is called *Selective Subset* if, for every vertex $v \in V(G)$, if $v \in V_\ell$, at least one of its nearest neighbors in $S \cup (V(G) \setminus V_\ell)$ has the same color as v. The *Minimum Selective Subset* (MSS) problem seeks to find a selective subset of minimum size. The problem was first introduced by Wilfong in 1991 [18] for a set of points in the Euclidean plane, where two major problems, MCS (Minimum Consistent Subset) and MSS, were proposed.

In graph algorithms, the only known result is that the MSS problem is NP-complete, as shown in [2] in 2018. Beyond this, no further progress has been made to date. In contrast, the MCS problem has been widely studied in various graph classes over the years. Therefore, in this work, we also extend the algorithmic study of MSS on various graph classes. We first present a $\mathcal{O}(\log n)$-approximation algorithm for general graphs with n vertices and regardless of the number of colors. We also show that the problem remains NP-complete even for planar graphs when restricted to just two colors. Finally, we provide linear-time algorithms for computing optimal solutions in trees and unit interval graphs for any number of colors[1].

1 Introduction

Many supervised learning methods use a colored training data set T in a metric space (X, d), where each element $t \in T$ has a color from the set of colors $C = \{1, 2, \ldots, c\}$. The goal is to find a subset $S \subseteq T$ with minimum cardinality such that every element of T is either in S or has at least one nearest neighbor in S with the same color. This problem, known as the *Minimum Consistent Subset* (MCS), was first introduced by Hart [13], whose work has received more than 2,800 citations.

The problem is NP-complete for three or more colors [18] and remains NP-complete for two colors [14] in $\mathbb{R}^2$. It is also W[1]-hard when parameterized by output size [6]. Various algorithms for the problem MCS in $\mathbb{R}^2$ have been proposed [2,4,6,18], highlighting its significance in machine learning and computational geometry. However, the problem MCS is closely related to MSS which is discussed by Wilfond [18].

The problem MSS plays a crucial role in optimizing data selection by identifying the smallest subset that preserves essential information. This is particularly useful in applications such as fingerprint recognition, character recognition, and pattern recognition, where it helps reduce redundancy and improve decision-making in classification and feature selection tasks.

Wilfong [18] proved that MSS is also NP-complete even with two colors in $\mathbb{R}^2$. Recently, [2] established an PTAS with c-color points and showed that the problem is W[2]-hard when parameterized by the size of the solution, while MSS is contained in W[1] when the number of colors is two in $\mathbb{R}^2$.

1.1 Notations and Definitions

For any graph $G = (V(G), E(G))$, we denote the set of vertices by $V(G)$ and the set of edges by $E(G)$. Without loss of generality, we use $[n]$ to denote the set of integers $\{1, \ldots, n\}$. We use an arbitrary vertex color function $C : V(G) \to [c]$, such that each vertex is assigned exactly one color from the set $[c]$. For a subset of vertices $U \subseteq V(G)$, let $C(U)$ represent the set of colors of the vertices in U, formally defined as $C(U) = \{C(u) \mid u \in U\}$.

For any two vertices $u, v \in V(G)$, the shortest path distance between u and v in G is denoted by $d(u, v)$. $d(u, v)$ is called *hop-distance* between u and v. For a vertex $v \in V(G)$, the distance between v and the set $U \subseteq V(G)$ in G is given by $d(v, U) = \min_{u \in U} d(v, u)$.

The nearest neighbors of v in the set U is denoted as $\hat{\mathrm{N}}(v, U)$, formally defined as

$$\hat{\mathrm{N}}(v, U) = \{u \in U \mid d(v, u) = d(v, U)\}.$$

Therefore, if $v \in U$, then $\hat{\mathrm{N}}(v, U) = \{v\}$.

$G[U]$ denotes the subgraph of G induced by $U \subseteq V(G)$, and $|U|$ is the cardinality of U. We use standard graph-theoretic notation and symbols as presented in [12].

Suppose $G = (V(G), E(G))$ is a given connected and undirected graph, where the vertices are partitioned into

*Indian Institute of Technology, Kharagpur, bubaimanna11@gmail.com

[1]The full version of this work can be found in [17].

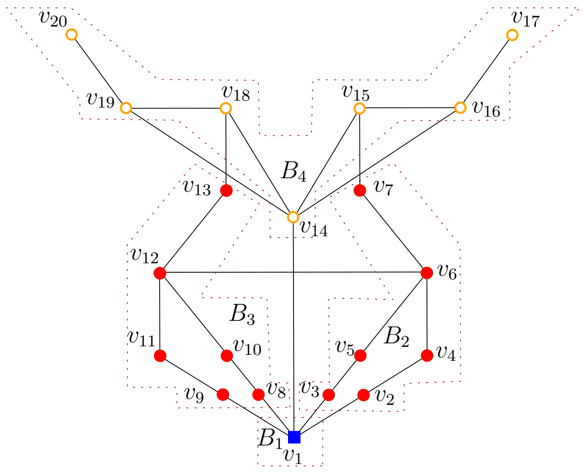


Figure 1: Colors: *blue= square*, *red= disk*, *orange=fdisk* and $V(G) = V_{blue} \cup V_{red} \cup V_{orange}$, where $V_{blue} = \{v_1\}$, $V_{red} = \{v_2, \ldots, v_{13}\}$, and $V_{orange} = \{v_{14}, \ldots, v_{20}\}$. The sets $\{v_1, v_4, v_5, v_{10}, v_{11}, v_{17}\}$ and $\{v_1, v_3, v_4, v_8, v_{11}, v_{17}\}$ both are MCS. Similarly, $\{v_1, v_2, v_3, v_7, v_8, v_9, v_{13}, v_{14}\}$ and $\{v_1, v_4, v_5, v_7, v_{10}, v_{11}, v_{13}, v_{14}\}$ are also MSS. Brown-dotted regions indicate the blocks. The complete list of blocks is $B_1 = \{v_1\}$, $B_2 = \{v_2, \ldots, v_7\}$, $B_3 = \{v_8, \ldots, v_{13}\}$, $B_4 = \{v_{14}, \ldots, v_{20}\}$.

c color classes, namely $V_1, V_2, \ldots, V_c$. This means that each vertex in $V(G)$ has a color from the set $[c]$, and each vertex in V_ℓ has color ℓ. A *Minimum Consistent Subset* (MCS) is a subset $S \subseteq V(G)$ of minimum cardinality such that for every vertex $v \in V(G)$, if $v \in V_\ell$, then

$$\hat{\mathrm{N}}(v, S) \cap V_\ell \neq \emptyset.$$

The definition of a selective subset is as follows:

Definition 1 *A subset $S \subseteq V(G)$ is called a* Minimum Selective Subset (MSS) *if, for each vertex $v \in V(G)$, if $v \in V_\ell$, the set of nearest neighbors of v in $S \cup (V(G) \setminus V_\ell)$, contains at least one vertex u such that $C(v) = C(u)$, and $|S|$ is minimum.*

In other words, we are looking for a vertex set $S \subseteq V(G)$ of minimum cardinality such that every vertex v has at least one nearest neighbor of the same color in the graph, excluding those vertices of the same color as v that are not in S.

Figure 1 illustrates that MCS and MSS are distinct and may not be unique for a given graph. The selective subset problem on graphs is defined as follows:

Selective Subset Problem on Graphs

Input: A graph $G = (V(G), E(G))$, a color function $C : V(G) \to [c]$, and an integer s.
Question: Does there exist a selective subset of size $\leq s$ for (G, C)?

Banerjee et al. [2] proved that MCS is W[2]-hard [7] when parameterized by the minimum consistent set size, even with two colors in general graphs. Dey et al. [9–11] provided polynomial-time algorithms for MCS on some simple graph classes including path, spider, caterpillar, comb and trees (for trees, $c = 2$). XP and NP-complete as well as the FPT algorithms (when c is a parameter) on trees, can be found in [1, 3]. MCS is also NP-complete in interval graphs [3] and APX-hard in circle graphs [15]. Variants such as the *Minimum Consistent Spanning Subset* (MCSS) and the *Minimum Strict Consistent Subset* (MSCS) of MCS have been studied in trees [5, 16]. However, the algorithmic results for MSS have not been extensively studied to date. Banerjee et al. [2] only showed that MSS is NP-complete in general graphs.

1.2 Results

Since only a hardness result for MSS in general graphs is known, our work provides new insights into the complexity and approximability of the problem, identifying cases where efficient approximations are achievable and where hardness persists. In Section 3, we present a $\mathcal{O}(\log n)$-approximation algorithm for MSS in general graphs, where the number of vertices is n and regardless of the number of colors.

Planar graphs are fundamental structures in graph theory and computational geometry, with many practical applications and rich structural properties. Therefore, in Section 4, we show that the problem MSS remains NP-complete even when restricted to planar graphs with just $c = 2$, highlighting the inherent difficulty of the problem even on well-behaved graph classes.

Trees are simple graph classes that often admit efficient algorithms, even when such algorithms are not possible for more general graph families. Unit interval graphs, which model intervals of equal length on the real line, are widely studied due to their applications in scheduling and biology and their tractable structure. Since both of these graph classes have been well explored in MCS [1, 3], it is natural and necessary to investigate MSS in the same classes. Therefore, in Sections 5 and 6, respectively, we present linear-time algorithms for finding optimal solutions to the MSS problem on trees and unit interval graphs with any number of colors. All proofs of (*)-marked results are in the full version [17].

2 Preliminaries

If all the vertices of a graph G are of the same color (that is, G is monochromatic), then any vertex of the graph is an MSS. Moreover, at least one vertex from each color class must be included in every selective subset;

otherwise, consider a vertex $v \in V_\ell$, and all vertices in $\hat{\mathrm{N}}(v, S \cup (V(G) \setminus V_\ell))$ must have a different color from that of v, violating the condition for a selective subset.

Definition 2 *A block is defined as a maximal connected subgraph whose vertices share the same color.*

Figure 1 illustrates an example of the blocks.

Lemma 1 *Any selective subset must contain at least one vertex from each block.*

Proof. Suppose, for contradiction, that M is a selective subset of a graph G, but there exists a block B_i such that $M \cap B_i = \emptyset$. Let the color assigned to the vertices of B_i be ℓ, and let $v \in B_i$ be any vertex.

Therefore, there must exist a nearest neighbor (say u) of v in $M \cup (V(G) \setminus V_\ell)$ such that $C(u) = C(v)$; otherwise, M would not be a selective subset. Since $M \cap B_i = \emptyset$, we have $u \notin B_i$. Thus, $u \in B_j$ for some $j \neq i$.

Let P denote the shortest path between v and u. As B_i and B_j are distinct blocks and $C(v) = C(u)$, there must exist at least one vertex $w \in P$ such that $C(w) \neq C(u)$. This implies that w lies closer to v than u does, i.e., $d(v, w) < d(v, u)$, and $w \in M \cup (V(G) \setminus V_\ell)$ with $C(w) \neq C(u)$.

Hence, instead of u, w is the nearest neighbor of v in $M \cup (V(G) \setminus V_\ell)$ with $C(w) \neq C(v)$, violating the definition of a selective subset. This contradiction completes the proof. □

The above proof indicates that, for any vertex $v \in B_i$, either v itself must belong to M, or there must exist a vertex $u \in B_i$ such that $u \in M$ and u is the nearest neighbor of v in $M \cup (V(G) \setminus V_\ell)$. In other words, each vertex $v \in B_i$ must have its nearest neighbor in $M \cup (V(G) \setminus V_\ell)$ that also lies within B_i. Therefore, we have the following observation:

Observation 1 *The blocks are independent of each other in the solution of a selective subset.*

3 $\mathcal{O}(\log n)$-Approximation Algorithm of MSS in General Graphs

Let $G = (V(G), E(G))$ be a graph with c colors and B_1, B_2, …, B_k be the blocks in G. The *Set Cover* problem is defined as follows: Given a universe I and a collection of m subsets S, the goal is to select a minimum number of subsets from S whose union covers all elements in I. The reduction of MSS in G to the *set cover* problem is as follows (see Figure 2).

Reduction. Let $B_{i,1} \subseteq B_i$ be the set of vertices adjacent to at least one vertex of a different color, for each $i = 1, \dots, k$. For each vertex $v \in B_i \setminus B_{i,1}$, if v in adjacent to at least one vertex of $B_{i,1}$, include v in $B_{i,2}$. Let $B^{all}_{i,1} = \bigcup_{i=1}^{k} B_{i,1} = \{a_1, \dots, a_{n_1}\}$ and $B^{all} = \bigcup_{i=1}^{k}(B_{i,1} \cup B_{i,2}) = \{b_1, \dots, b_{n_2}\}$ where $n_1 \leq n$ and $n_2 \leq n$. Therefore, each b_j is a vertex of either $B_{i,1}$ or $B_{i,2}$ for $j = 1, \dots, n_2$ and for some $i \in \{1, \dots, k\}$.

Construct a universe set $I = \{e_1, \dots, e_{n_1}\}$, where each e_i corresponds to vertex $a_i \in B^{all}_{i,1}$. For each $b_i \in B^{all}$, define a set $S_i \subseteq I$ such that $e_j \in S_i$ if the corresponding vertex a_j and the vertex b_i are either the same vertex or adjacent and in the same block. Let $S = \{S_1, \dots, S_{n_2}\}$.

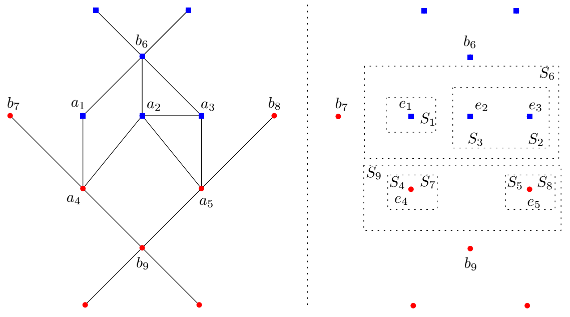


Figure 2: Reduction to set cover. Colors: *blue= square*, *red= disk*. Sets are $B_{1,1} = \{a_1, a_2, a_3\}$, $B_{2,1} = \{a_4, a_5\}$, $B_{1,2} = \{b_6\}$, $B_{2,2} = \{b_7,\ b_8, b_9\}$. Sets are $B^{all}_{i,1} = \{a_1, \dots, a_5\}$, $B^{all} = \{b_1,\ \dots, b_9\}$, where $b_i = a_i$ for $i = 1, \dots, 5$. The set elements for the set cover problem is $I = \{e_1, \dots, e_5\}$ where $e_i = b_i = a_i$ for $i = 1, \dots, 5$. The subsets are given $S_1 = \{e_1\}$, $S_2 = S_3 = \{e_2, e_3\}$, $S_4 = S_7 = \{e_4\}$, $S_5 = S_8 = \{e_5\}$, $S_6 = \{e_1, e_2, e_3\}$, $S_9 = \{e_4, e_5\}$. The elements inside the dotted rectangles represent the sets. The selective subset is $M = \{b_6, b_9\}$, and the corresponding set cover is $\{S_6, S_9\}$.

Lemma 2 * *If a set cover of I uses some sets of S, then the vertices represented by those sets form a selective subset, and vice versa.*

Theorem 3 *The Minimum Selective Subset problem admits an $\mathcal{O}(\log n)$-approximation in general graphs.*

Proof. By Lemma 2, the MSS problem reduces to a Set Cover instance where the universe size is $n_1 \leq n$, with $n = |V(G)|$.

The greedy algorithm for Set Cover achieves an $\mathcal{O}(\log n_1)$-approximation. Since $n_1 \leq n$, it follows that $\log n_1 \leq \log n$, and therefore the approximation factor becomes $\mathcal{O}(\log n)$ in terms of the original graph size.

As the reduction preserves the approximation guarantee, this yields an $\mathcal{O}(\log n)$-approximation algorithm for the MSS problem in general graphs. □

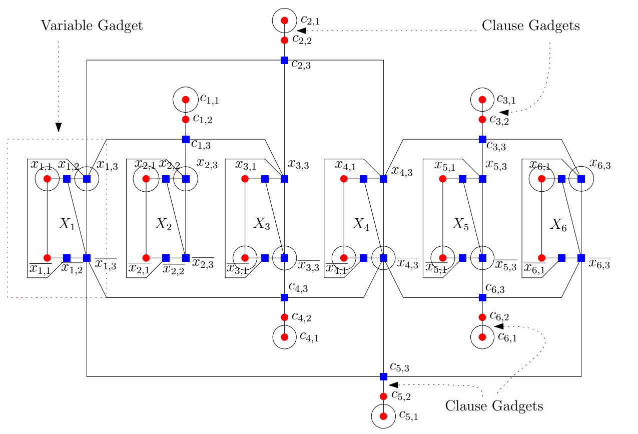


Figure 3: Reduction from PRM-3SAT to MSS in planar graph. Colors: *blue= square*, *red= disk*. The SAT expression is $\theta = c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge c_5 \wedge c_6$. The clauses are $c_1 = (x_1 \vee x_2 \vee x_3), c_2 = (x_1 \vee x_3 \vee x_4), c_3 = (x_4 \vee x_5 \vee x_6), c_4 = (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4}), c_5 = (\overline{x_1} \vee \overline{x_4} \vee \overline{x_6}), c_6 = (\overline{x_4} \vee \overline{x_5} \vee \overline{x_6})$ with variables $x_1 = x_2 = x_6 = 1$ and $x_3 = x_4 = x_5 = 0$. The vertices inside small circles are in the selective subset.

4 NP-Hardness of MSS in Planar Graphs

We reduce an instance of Planar Rectilinear Monotone 3-SAT (PRM-3SAT) θ to a bichromatic planar graph. The definition of PRM-3SAT is provided in [17]. In [8], it is shown that PRM-3SAT is NP-complete. It is also shown that, given a PRM-3SAT formula θ, its embedding described in [8] can be obtained in polynomial time.

Reduction. We embed an instance of PRM-3SAT formula θ with n variables $x_1, x_2, \ldots, x_n$ and m clauses $c_1, c_2, \ldots, c_m$ into a bichromatic planar graph $G = (V(G), E(G))$ (see Figure 3).

Variable gadget: For each variable x_i $(1 \leq i \leq n)$, we construct a variable gadget X_i shown as a brown dotted rectangle in Figure 3, as follows: A path $(x_{i,1}, x_{i,2}, x_{i,3})$ of length two, where $C(x_{i,1}) =$ *red* and $C(x_{i,2}) = C(x_{i,3}) =$ *blue*, is called the *positive literal path* of x_i. Similarly, a path $(\overline{x_{i,1}}, \overline{x_{i,2}}, \overline{x_{i,3}})$ of length two, where $C(\overline{x_{i,1}}) =$ *red* and $C(\overline{x_{i,2}}) = C(\overline{x_{i,3}}) =$ *blue*, is called the *negative literal path* of x_i.

The vertex $x_{i,1}$ is adjacent to $\overline{x_{i,1}}$, and $x_{i,2}$, $x_{i,3}$ are adjacent to $\overline{x_{i,3}}$, $\overline{x_{i,2}}$, respectively. Additionally, $x_{i,3}$ is adjacent to $\overline{x_{i,3}}$.

Clause gadget: For each clause c_j $(1 \leq j \leq m)$, the clause gadget C_j is as follows: A path $(c_{j,1}, c_{j,2}, c_{j,3})$ of length two, where $C(c_{j,1}) = C(c_{j,2}) =$ *red* and $C(c_{j,3}) =$ *blue*, is called *clause path* for the clause c_j.

If a clause c_j consists of three positive literals x_i, x_l, x_t (i.e., $c_j = (x_i \vee x_l \vee x_t)$), then the vertex $c_{j,3}$ adjacent to $x_{i,3}$, $x_{l,3}$, and $x_{t,3}$ from their respective positive literal paths. If c_j has three negative literals $\overline{x_i}, \overline{x_l}, \overline{x_t}$, then $c_{j,3}$ is adjacent to $\overline{x_{i,3}}, \overline{x_{l,3}}, \overline{x_{t,3}}$ from the corresponding negative paths as shown in the Figure 3.

The construction of the graph G is now complete. Notably, G remains planar because its embedding (Figure 3) closely resembles the PRM-3SAT embedding described in [8]. Thus, for n variables and m clauses, the bichromatic planar graph G contains $6n + 3m$ vertices and $8n + 5m$ edges. We set $V(G) = V_r \cup V_b$, where V_r is the set of *red* vertices and V_b is the set of *blue* vertices.

Lemma 4 * *θ is satisfied if and only if G has a selective subset of size $2n + m$.*

Theorem 5 *Finding a minimum selective subset is NP-complete for planar graphs with two colors.*

Proof. It is easy to see that the problem is in NP. As for NP-complete, Lemmas 4 establishes a relationship between θ and the size of the selective subset of G in polynomial time. Therefore, MSS is NP-complete in planar graphs. □

Remark. The above reduction remains valid even if the pair of vertices $\{c_{j,1}, c_{j,2}\}$ in each clause gadget is assigned a distinct (except blue color), unique color not shared across gadgets. That is, we may assign a different color to each such pair for $1 \leq j \leq m$, and the reduction still preserves the equivalence between satisfying the PRM-3SAT formula θ and the existence of a selective subset of size $2n + m$.

5 Linear-time Algorithm of MSS in Trees

We now describe a linear-time algorithm for finding a MSS in tree. The key idea is based on Observation 1, which tells us that we can solve the problem independently on each block. So, to simplify notation and ideas, we focus on just one block.

Let $T = (V(T), E(T))$ be a tree rooted at a vertex r with $|V(T)| = n$ and a total of c colors. For each block B, we compute a minimum selective subset M_B using Lemma 1. The algorithm has two phases: initialization (Algorithm 1 in [17]) and selection (Algorithm 2 in [17]). The final solution is the union of the subsets computed for each block.

The algorithm follows these steps (see Figure 4):

Initialization (Algorithm 1).

- Start with the empty sets: $M_B := \emptyset$, $B^1 := \emptyset$, and $B^2 := \emptyset$.
- For each vertex $v \in B$, if v is adjacent to a vertex of a different color, include v in B^1.

- For each vertex $v \in B \backslash B^1$, if v is adjacent to at least one vertex in B^1, then include v in B^2.
- Define $B^{all} = B^1 \cup B^2$ (note that $B^1 \cap B^2 = \emptyset$).

Subtree Formation (refer to lines 1-4 in Algorithm 2).

- Since $B^{all} \subseteq B$, B^{all} induces one or more connected induced subtrees in T, and each subtree consists of vertices of the same color because $B^{all} \subseteq B$. We consider such induced connected maximal subtrees.
- Let T^{v_1}, T^{v_2}, ..., T^{v_t} be such connected maximal subtrees in T formed by the vertices of B^{all} and call their roots $v_1, v_2, \ldots, v_t$, respectively. Note that, since the whole tree is rooted, a root is naturally defined for each subtree.

Selection Process (refer to lines 5-31 in Algorithm 2).

- For each subtree T^{v_j} (where $1 \leq j \leq t$), start from the *lowest-level vertex* u. A lowest-level vertex of a tree is a vertex that is farthest from the root.
- u must belong to either B^1 or B^2.
- If $u \in B^2$:
  - Remove u from B^2, B^{all}, and T^{v_j}, as it must be adjacent to a vertex in B^1.
- If $u \in B^1$:
  - If u has a parent (say v) in T^{v_j}, add v to M_B and remove v along its children from B^1, B^2, B^{all}, and T^{v_j}.
  - If u has no parent in T^{v_j}, add u to M_B and remove it from B^1, B^{all}, and T^{v_j}.
  - If the grandparent of u (if it exists in T^{v_j}) belongs to B^1, move it from B^1 to B^2, and update B^{all} accordingly.
- Repeat the **Selection Process** until T^{v_j} becomes empty. Once T^{v_j} has no vertices, increase $j \leftarrow j+1$ and repeat the **Selection Process** until $j = t$.

We apply the algorithm to each block B and obtain subsets M_B. The final solution is $M = \bigcup_{B \in T} M_B$. We now prove that M is not just selective, but also minimum.

Lemma 6 *For any block B and any vertex $u \in B^1$, either $u \in M$ or at least one adjacent vertex of u in B^{all} must belong to M.*

Proof. Let X be the set that contains u and all its adjacent vertices in B^{all}. Suppose T^{v_j} is the subtree that contains the vertices of X.

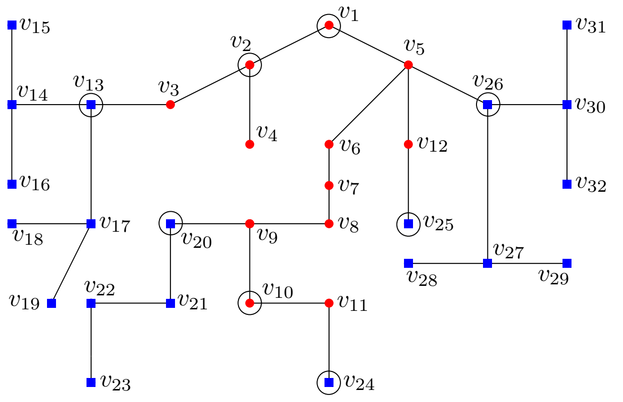


Figure 4: Colors: *blue= square*, *red= disk*. $r = v_1$ is the root of the tree T. The blocks are $B_1 = \{v_1, \ldots, v_{12}\}$, $B_2 = \{v_{13}, \ldots, v_{19}\}$, $B_3 = \{v_{20}, \ldots, v_{23}\}$, $B_4 = \{v_{24}\}$, $B_5 = \{v_{25}\}$, $B_6 = \{v_{26}, \ldots, v_{32}\}$. $B_1^1 = \{v_3, v_5, v_9, v_{11}, v_{12}\}$, $B_1^2 = \{v_1, v_2, v_6, v_8, v_{10}\}$, $B_2^1 = \{v_{13}\}$, $B_2^2 = \{v_{14}, v_{17}\}$, $B_3^1 = \{v_{20}\}$, $B_3^2 = \{v_{21}\}$, $B_4^1 = \{v_{24}\}$, $B_4^2 = \emptyset$, $B_5^1 = \{v_{25}\}$, $B_5^2 = \emptyset$, $B_6^1 = \{v_{26}\}$, $B_6^2 = \{v_{27}, v_{30}\}$. $T_1^{v_1} = \{v_1, v_2, v_3, v_5, v_6, v_{12}\}$, $T_1^{v_8} = \{v_8, v_9, v_{10}, v_{11}\}$, $T_2^{v_{13}} = \{v_{13}, v_{14}, v_{17}\}$, $T_3^{v_{20}} = \{v_{20}, v_{21}\}$, $T_4^{v_{24}} = \{v_{24}\}$, $T_5^{v_{25}} = \{v_{25}\}$, $T_6^{v_{26}} = \{v_{26}, v_{27}, v_{30}\}$. MSS $= \{v_2, v_1, v_{10}, v_{13}, v_{20}, v_{24}, v_{25}, v_{26}\}$. The vertices inside small circles are in the minimum selective subset.

By the process described in Algorithm 2, after a finite number of iterations, a vertex from X will become a lowest-level vertex of T^{v_j} (since we delete vertices in each iteration until the subtree becomes empty).

We analyze two cases:

- If u becomes a lowest-level vertex, then its parent (if it exists in T^{v_j}) must be included in M_B because $u \in B^1$, meaning that an adjacent vertex of u from B^{all} is included in M_B. If the parent does not exist, then u itself must be included in M_B according to Algorithm 2 as $u \in B^1$.
- If a vertex $v \in X \setminus \{u\}$ becomes a lowest-level vertex, we consider two subcases:
  - If $v \in B^1$, then its parent must be u, as u is the only adjacent vertex of v (as v is lowest-level vertex), and u must be included in M_B.
  - If $v \in B^2$, then v is removed from the tree. In subsequent iterations, either u eventually becomes a lowest-level vertex (in which case we proceed as above), or one of its remaining neighbors becomes a lowest-level vertex in B^1, again forcing $u \in M_B$. If all neighbors of u are in B^2 and get deleted, u itself becomes lowest-level, completing the process.

Combining all these cases, we conclude that $X \cap M_B \neq$

$\emptyset$. Thus, either $u \in M$ or at least one adjacent vertex of u in B^{all} belongs to M. □

Lemma 7 * *M is a minimum selective subset of the tree T.*

Lemma 8 * *Algorithm 2 runs in $O(n)$ time.*

Remark. Since our algorithm is described for a single block and applies uniformly to all blocks, the core idea becomes clear: we aim to dominate all vertices in a block that are adjacent to vertices in other blocks. This naturally aligns with an MSOL-expressible formulation. Therefore, the algorithm is not only applicable to trees but also extends to graphs of constant treewidth.

6 Linear-time Algorithm of MSS in Unit Interval Graphs

Let $I = (V(I), E(I))$ be a unit interval graph with $|V(I)| = n$. Each unit interval in $V(I)$ is on the x-axis, which is treated as a vertex in I. Two vertices are adjacent if their corresponding unit intervals intersect, forming an *edge* between them. The set of all such edges is denoted as $E(I)$. Each interval has a *left end* and a *right end*.

An interval v is called a *left adjacent* of an interval u if the x-coordinate of the left endpoint of v is less than that of u, and v intersects u. Similarly, an interval v is called a *right adjacent* of an interval u if the x-coordinate of the left endpoint of v is greater than that of u, and v intersects u. The *leftmost* interval is the one whose left endpoint has the smallest x-coordinate among all adjacent intervals, while the *rightmost* interval has the largest x-coordinate at its right endpoint.

Each interval is assigned a color from a set of c colors. The **Initialization** step is the same as discussed in Section 5. The other steps are also similar but with a little change, which is as follows:

Unit Interval Subgraph Formation.

- Since $B^{all} \subseteq B$, each B^{all} induces one or more connected induced unit interval subgraphs in I, and each such graph consists of vertices of the same color because $B^{all} \subseteq B$. We focus on these induced connected maximal unit interval subgraphs.
- Let $I_1, I_2, \ldots, I_t$ be induced connected maximal unit interval subgraphs in I formed by the vertices of B^{all}.

Selection Process.

- For each unit interval subgraph I_j (where $1 \leq j \leq t$), start from the leftmost interval u.
- u must belong to either B^1 or B^2.
- If $u \in B^2$:
  - Remove u from B^2, B^{all}, and I_j, as it must be adjacent to a interval in B^1.
- If $u \in B^1$:
  - If the rightmost adjacent interval of u exists (say v) in I_j, add v to M_B and remove w along with all of its left adjacent vertices from B^1, B^2, B^{all}, and I_j. Also move all the right adjacent intervals (if exists) of v from B^1 to B^2 and update the sets B^{all} accordingly.
  - If u has no right adjacent interval in I_j, add u to M_B and remove it from B^2, B^{all} and I_j.
- Repeat the **Selection Process** until I_j becomes empty. Once I_j has no vertices, increment $j \leftarrow j+1$ and repeat the **Selection Process** until $j = t$.

Assume $M = \bigcup_{B \in I} M_B$. Since the above algorithm is very similar to the algorithm for trees, Lemma 6 must also hold for unit interval graphs.

Lemma 9 * *M is a minimum selective subset of the interval graph I.*

The proof of this lemma is quite similar to the proof of the Lemma 7. The runtime is also the same as explained in Lemma 8.

7 Remarks

As the MSS problem is NP-complete in planar graphs, developing approximation algorithms and studying its parameterized complexity (specifically, designing FPT algorithms when the number of colors c is the parameter) are important open problems. Additionally, designing approximation algorithms and establishing hardness results for other graph classes, such as circular-arc graphs, present further promising directions for future research.

Acknowledgments

I would like to sincerely thank Bodhayan Roy for his constant guidance and support throughout this work. I am also grateful to Aritra Banik, Anil Maheshwari, Subhas C Nandy and Sasanka Roy for their valuable suggestions and feedback that helped improve the initial version of this paper. Their insights were instrumental during the early stages of this research. I also thank the anonymous reviewers for their thoughtful comments and suggestions, which helped enhance the quality and presentation of the paper.

References

[1] H. Arimura, T. Gima, Y. Kobayashi, H. Nochide, and Y. Otachi. Minimum consistent subset for trees revisited. *CoRR*, abs/2305.07259, 2023.

[2] S. Banerjee, S. Bhore, and R. Chitnis. Algorithms and hardness results for nearest neighbor problems in bicolored point sets. In M. A. Bender, M. Farach-Colton, and M. A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics*, pages 80–93, 2018.

[3] A. Banik, S. Das, A. Maheshwari, B. Manna, S. C. Nandy, K. Priya K. M., B. Roy, S. Roy, and A. Sahu. Minimum Consistent Subset in Trees and Interval Graphs. In S. Barman and S. Lasota, editors, *44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024)*, volume 323 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[4] A. Biniaz, S. Cabello, P. Carmi, J. D. Carufel, A. Maheshwari, S. Mehrabi, and M. Smid. On the minimum consistent subset problem. *Algorithmica*, 83(7):2273–2302, 2021.

[5] A. Biniaz and P. Khamsepour. The minimum consistent spanning subset problem on trees. *Journal of Graph Algorithms and Applications*, 28(1):81–93, 2024.

[6] R. Chitnis. Refined lower bounds for nearest neighbor condensation. In S. Dasgupta and N. Haghtalab, editors, *International Conference on Algorithmic Learning Theory, 29 March - 1 April 2022, Paris, France*, volume 167 of *Proceedings of Machine Learning Research*, pages 262–281. PMLR, 2022.

[7] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[8] M. de Berg and A. Khosravi. Optimal binary space partitions in the plane. In M. T. Thai and S. Sahni, editors, *Computing and Combinatorics*, pages 216–225, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[9] S. Dey, A. Maheshwari, and S. C. Nandy. Minimum consistent subset of simple graph classes. In A. Mudgal and C. R. Subramanian, editors, *Algorithms and Discrete Applied Mathematics*, pages 471–484, Cham, 2021. Springer International Publishing.

[10] S. Dey, A. Maheshwari, and S. C. Nandy. Minimum consistent subset problem for trees. In E. Bampis and A. Pagourtzis, editors, *Fundamentals of Computation Theory - 23rd International Symposium, FCT 2021, Athens, Greece, September 12-15, 2021, Proceedings*, volume 12867 of *Lecture Notes in Computer Science*, pages 204–216. Springer, 2021.

[11] S. Dey, A. Maheshwari, and S. C. Nandy. Minimum consistent subset of simple graph classes. *Discret. Appl. Math.*, 338:255–277, 2023.

[12] R. Diestel. Graph theory, volume 173 of. *Graduate texts in mathematics*, page 7, 2012.

[13] P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

[14] K. Khodamoradi, R. Krishnamurti, and B. Roy. Consistent subset problem with two labels. In B. Panda and P. P. Goswami, editors, *Algorithms and Discrete Applied Mathematics*, pages 131–142, 2018.

[15] B. Manna. Minimum Consistent Subset in Interval Graphs and Circle Graphs. *arXiv e-prints*, page arXiv:2405.14493, May 2024.

[16] B. Manna. Minimum Strict Consistent Subset in Paths, Spiders, Combs and Trees. *arXiv e-prints*, page arXiv:2405.18569, May 2024.

[17] B. Manna. Minimum Selective Subset on Some Graph Classes. *arXiv e-prints*, page arXiv:2507.00235, June 2025.

[18] G. Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, pages 224—-233, 1991.

Combinatorial Reconfiguration: Flipping Non-Crossing Spanning Trees

Ferran Hurtado Memorial Lecture

Birgit Vogtenhuber[*]

Flipping edges in non-crossing geometric graphs has been one of the core research topics of Ferran Hurtado, which is witnessed by the fact that he chose this topic for his invited talk at CCCG 2003. In this talk, I will report on flips in non-crossing spanning trees.

For a set P of n points in the plane, a non-crossing (geometric) spanning tree is a spanning tree of the points in which every edge is a straight-line segment between a pair of points and no two edges intersect except at a common endpoint. In its most general form, an *edge flip* in a non-crossing spanning tree T of P is the operation of removing one edge from T and adding another edge such that the resulting structure is again a non-crossing spanning tree of P. Besides this edge flip, several more restricted flip operations have been considered for spanning trees. Most notably these include *compatible edge flips* (where the exchanged edges are non-crossing), *rotations* (where the exchanged edges share an endpoint), and *edge slides* (where the exchanged edges together with some third edge form an uncrossed triangle).

The problem of transforming one non-crossing spanning tree into another one via a sequence of flip operations of some type has been widely studied. As already reported by Ferran in 2003, such a transformation is always possible with a finite number of flips even for the case of edge slides, which is the most restricted of the flip operations. We will review recent results concerning bounds on the number of flips that are sometimes required and always sufficient. We will close with possible properties of shortest flip sequences, the algorithmic question of finding such sequences, and some open problems.

[*]Graz University of Technology, birgit.vogtenhuber@tugraz.at

Motion Planning of Disk and Rectangular Robots

Eduard Eiben[*] Iyad Kanj[†] Salman Parsa[‡]

Abstract

We study the parameterized complexity of the motion planning of geometric-shaped robots in the plane, where the task is to move $k \in \mathbb{N}$ robots from their starting to their destination points without collisions.

We focus on disk and axis-aligned rectangular robots and on rectilinear motion. We consider both types of motions: coordinated (i.e., robots move in parallel) and serial. We also consider two objective functions: the total rectilinear length traveled and the total number of moves.

We prove that, in the presence of (rectangular) obstacles, the coordinated rectilinear motion planning of rectangular robots with the target of minimizing the total number of moves, is W[1]-hard parameterized by the number k of robots. This gives strong evidence that the problem is unlikely to be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$, for some computable function f, where n is the number of bits that encode the coordinates of the robots and obstacles, contrasting the recent result for the setting of obstacle-free motion in the plane, which showed the problem to be fixed-parameter tractable (FPT).

We then focus on the case of free motion in the plane, and consider the rectilinear motion of axis-aligned rectangular robots, with the goal of minimizing their total rectilinear travel length. We show that the restriction in which we require each robot to travel along a shortest rectilinear path, is NP-hard.

Finally, we consider the free rectilinear motion of congruent disk robots in the plane.

We show that the problem is FPT parameterized by k, for both target functions, if we restrict the motion of the centers of the disks to a unit grid, a restriction that has applications in real-world robotic problems.

1 Introduction

Motivation and Related Work. We investigate the parameterized complexity of several motion planning problems of disk robots and of axis-aligned rectangular robots in the plane, where the parameter under consideration is the number of robots. In this setting, we are given a set of k robots, each with a starting and ending position, and the goal is to compute a schedule in which each robot reaches its destination without colliding with other robots, and while minimizing a certain objective function. The two objective functions under consideration are the total rectilinear length traveled by all the robots in the schedule, and the total number of moves in the schedule. We consider only rectilinear translation motion, that is, each move is a translation along either a horizontal or vertical direction. We consider two settings: the setting of free motion in the plane (i.e., motion in the plane with no obstacles), and the setting where rectangular obstacles may be present. We also consider two types of motion: coordinated, where robots may move in parallel, and serial, where robots move one at a time. Most of the problems under consideration are either NP-hard or PSPACE-hard.

Our work assumes a Turing machine model and that the input length is the number of bits needed to represent the coordinates of the points defining the geometric objects (i.e., robots/obstacles) in the problem instance. Note that this number can be much larger than the parameter k. We believe that this model is more realistic to the problems under consideration than the real RAM model [35], which assumes that arithmetic operations over the reals can be performed in constant time.

There has been an enormous body of work on geometric motion planning problems, mainly focusing on the feasibility (i.e., whether any schedule exists), dating back to the works of Schwartz and Sharir in the 1980s [32–34]. They showed that deciding the feasibility of a problem instance for two disks in a region bounded by n "walls" can be done in time $\mathcal{O}(n^3)$ [32]; their result can be generalized to any number, k, of disks to yield an $\mathcal{O}(n^{h(k)})$-time algorithm, for some function h of k. Ramanathan and Alagar [30] improved this to $\mathcal{O}(n^k)$, conjecturing that this running time is asymptotically optimal. The feasibility of the coordinated motion planning of rectangular robots confined to a bounding box was shown to be PSPACE-hard [22, 23], even for congruent square robots [38]. The problem of moving disks among polygonal obstacles in the plane was shown to be strongly NP-hard [28].

More recently, there has been quite some work on the continuous collision-free motion of a constant number of rectangles in the plane, to optimize or approximate the total Euclidean traveled length; we refer to [2, 15, 27] for

[*]Department of Computer Science, Royal Holloway, University of London, Egham, UK, `eduard.eiben@rhul.ac.uk`

[†]School of Computing, DePaul University, Chicago, USA, `ikanj@cdm.depaul.edu`

[‡]School of Computing, DePaul University, Chicago, USA, `s.parsa@depaul.edu`

some of the most recent works on this topic.

Dumitrescu and Jiang [12] studied the problem of moving unit disks in an obstacle-free environment. They consider two types of motion: translation and sliding (a move along a continuous curve). In a single step, a unit disk may move any distance either along a line (translation) or a curve (sliding) provided that it does not collide with another disk. They showed that deciding whether the disks can reach their destinations within $\ell \in \mathbb{N}$ moves is NP-hard, for either of the two motion types. For more information, we refer to the survey [11].

Recently, there has been a surge of works on the parametrized complexity of combinatorial variants of coordinated motion planning problems (i.e., on grids and graphs) [7, 8, 13, 14, 17, 18]. Notably, minimizing the total number of moves and the total travel length (referred to as the energy) for the classical coordinated motion planning problem on grids featured as the Symposium on Computational Geometry (SoCG 2021) Challenge problem [16], due to its applications in artificial intelligence [4, 36, 39] and robotics [3, 21, 37].

Very recently, the parameterized complexity of translating axis-aligned rectangles in the free plane, with the goal of minimizing the number of moves was studied by Kanj and Parsa [24]. It was shown in [24] that the problem is FPT parameterized by the number of robots. The parameterized complexity for other shapes – in particular congruent disks – and for environments with obstacles were posed as open problems in [24]. Moreover, the work in [24] focused on the total number of moves, and did not consider the variants of the problems in which the goal is to minimize the travel length/energy.

In this paper, we study the parameterized complexity of these fundamental geometric motion planning problems, answering some of the open questions in [24], and extending their work to the objective function of minimizing the total travel length.

Contributions. We present both hardness and algorithmic results for the motion planning problems of disk robots and of axis-aligned rectangular robots in the plane. Our main contributions are:

(i) We prove that, in the presence of rectangular obstacles, the coordinated rectilinear motion planning of rectangular axis-aligned robots with the target of minimizing the total number of moves, is W[1]-hard parameterized by the number k of robots, giving strong evidence that it is unlikely to be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$, for some computable function f. This answers an open question in [24], and contrasts the case of motion in the free plane, where the problem is FPT [24].

We then consider free motion and show:

(ii) The restriction of the rectilinear motion of axis-aligned rectangular robots, with the goal of minimizing the rectilinear total travel length, to instances in which each robot is required to travel along a shortest rectilinear path (i.e., whose length is equal to the rectilinear distance between the starting and ending positions of the robot), is NP-hard. A byproduct of this hardness result – among others that we show in this paper – is the NP-hardness of a restriction of the classical coordinated motion planning problem on full rectangular grids, to instances where we require each robot to travel along a shortest grid path between its starting and ending position (i.e., whose length is equal to the Manhattan distance between the starting and ending position of the robot on the grid). We also describe an FPT algorithm for the rectilinear motion of axis-aligned rectangular robots, with the goal of minimizing the rectilinear total travel length.

Finally, we consider the rectilinear free motion of congruent disk robots in the plane.

(iii) We show that the problem is FPT parameterized by k, for both target functions, the total number of moves and the total rectilinear traveled length, if we restrict the motion of the centers of the disks to a unit grid. This restriction models problems that arise in real-world applications. For instance, the movement of Amazon Kiva robots [29] is restricted to a floor grid.

2 Preliminaries and Problem Definition

The $\mathcal{O}^*()$ notation hides a polynomial function in the input size n, which is the length of the binary encoding of the instance. We write $[k]$ for the set $\{1, \ldots, k\}$.

A *parameterized problem* Q is a subset of $\Omega^* \times \mathbb{N}$, where Ω is a fixed alphabet. Each instance of Q is a pair (I, κ), where $\kappa \in \mathbb{N}$ is called the *parameter*. A parameterized problem Q is *fixed-parameter tractable* (FPT) [6, 10], if there is an algorithm, called an FPT-*algorithm*, that decides whether an input (I, κ) is a member of Q in time $f(\kappa) \cdot |I|^{\mathcal{O}(1)}$, where f is a computable function and $|I|$ is the input instance size. The class FPT denotes the class of all fixed-parameter tractable parameterized problems.

A parameterized problem Q is FPT-*reducible* to a parameterized problem Q' if there is an algorithm, called an FPT-*reduction*, that transforms each instance (I, κ) of Q into an instance (I', κ') of Q' in time $f(\kappa) \cdot |I|^{\mathcal{O}(1)}$, such that $\kappa' \leqslant g(\kappa)$ and $(I, \kappa) \in Q$ if and only if $(I', \kappa') \in Q'$, where f and g are computable functions. Based on the notion of FPT-reducibility, a hierarchy of parameterized complexity, *the W-hierarchy* $= \bigcup_{t \geqslant 0} \mathsf{W}[t]$, where $\mathsf{W}[t] \subseteq \mathsf{W}[t+1]$ for all $t \geqslant 0$, has been introduced, in which the 0-th level W[0] is the class FPT. The notion of hardness has been defined for each level W[t] of the W-hierarchy for $t \geqslant 1$ [6, 10]. It is commonly believed that W[1] $\neq$ FPT (see [6, 10]).

Let $\mathcal{R} = \{R_i \mid i \in [k]\}$ be a set of robots. We assume, both at the starting and ending positions, that the robots are pairwise non-overlapping (in their interiors). Depending on the problem under consideration, the robots in $\mathcal{R}$ can either be all axis-aligned rectangular robots (not necessarily congruent), or all congruent disk robots of fixed radius, but not a combination of both. We will refer to a robot by its identifying name (e.g., R_i), which determines the location of its center (either the center of the corresponding disk, or the center of the corresponding rectangle – the intersection of its two diagonals) in the schedule at any time step.

For two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$, the *rectilinear* distance, or the *Manhattan* distance, between p and q, is defined as $d_1(p, q) = |x_p - x_q| + |y_p - y_q|$.

A *translation move*, or a *move*, for $R_i \in \mathcal{R}$ w.r.t. a direction $\overrightarrow{v}$, is a translation of R_i by a vector $\alpha \cdot \overrightarrow{v}$ where $\alpha > 0$; we denote the move as $(R_i, \alpha\overrightarrow{v})$. For a vector $\overrightarrow{u}$, $\mathsf{translate}(R_i, \overrightarrow{u})$ denotes the axis-aligned rectangle resulting from translating R_i by vector $\overrightarrow{u}$.

In this paper, we consider only axis-aligned translations, that is translations using the vectors in $\mathcal{V} = \{\overrightarrow{H}^-, \overrightarrow{H}^+, \overrightarrow{V}^-, \overrightarrow{V}^+\}$, which are the negative and positive unit vectors of the x- and y-axis, respectively.

We consider two types of moves: *serial* and *coordinated*, where the former type corresponds to the robots moving one at a time (i.e., a robot must finish its move before the next starts), and the latter type corresponds to (possibly) multiple robots moving simultaneously. Formally, a coordinated move is a move in which a subset X of robots move simultaneously and at the same speed. The move ends when all the robots in X reach their desired locations during that move.

We now define collision for the two types of motion.

For a robot R_i that is translated by a vector $\overrightarrow{v}$, we say that R_i *collides* with a stationary robot $R_j \neq R_i$, if there exists $0 \leqslant x \leqslant 1$ such that R_j and $\mathsf{translate}(R_i, x \cdot \overrightarrow{v})$ intersect in their interior. For two distinct robots R_i and R_j that are simultaneously translated by vectors $\overrightarrow{v_i}$ and $\overrightarrow{v_j}$, respectively, we say that R_i and R_j *collide* if there exists $0 \leqslant x \leqslant 1$ such that $\mathsf{translate}(R_i, x \cdot \overrightarrow{v_i})$ and $\mathsf{translate}(R_j, x \cdot \overrightarrow{v_j})$ intersect in their interior.

A *serial schedule* (resp. *coordinated schedule*) $\mathcal{S}$ for $\mathcal{R}$ is a sequence of axis-aligned collision-free serial (resp. coordinated) moves such that after all the moves in $\mathcal{S}$, each R_i ends at its final destination, for $i \in [k]$. The *length* $|\mathcal{S}|$ of the schedule is the number of moves in it, and the cost of $\mathcal{S}$, denoted $cost(\mathcal{S})$, is the total rectilinear distance traveled by all the (centers of the) robots in $\mathcal{S}$. More specifically, suppose that $\mathcal{S} = \langle (R_{i_1}, \overrightarrow{v_1}), \ldots, (R_{i_\ell}, \overrightarrow{v_\ell}) \rangle$, for some $\ell \in \mathbb{N}$, then $|S| = \ell$, and $cost(\mathcal{S}) = \sum_{j=1}^{\ell} ||\overrightarrow{v_j}||$. (Note that each vector in $\mathcal{S}$ either has horizontal or vertical orientation.) In this paper, we study the following problems:

MOVES RECTANGLES MOTION PLANNING (Moves-Rect-MP)
Given: A set of pairwise non-overlapping axis-aligned rectangular robots $\mathcal{R} = \{R_i \mid i \in [k]\}$ each given with its starting and final positions; $k, \lambda \in \mathbb{N}$.
Question: Is there a schedule for $\mathcal{R}$ of length $\leqslant \lambda$?

Let $\mu \geqslant 0$ be a constant.

GRID-MOVES μ-DISKS MOTION PLANNING (Grid-Moves-μ-Disk-MP)
Given: A set of pairwise non-overlapping congruent disk robots $\mathcal{R} = \{R_i \mid i \in [k]\}$ of radius μ, each given with its starting and final positions; an $N \times M$ unit grid Ω, where $N, M \in \mathbb{N}$; $k, \lambda \in \mathbb{N}$.
Question: Is there a schedule for $\mathcal{R}$ of length at most λ in which the centers of the robots in $\mathcal{R}$ are confined to (moving on) points of Ω?

COST RECTANGLES MOTION PLANNING (Cost-Rect-MP)
Given: A set of pairwise non-overlapping axis-aligned rectangular robots $\mathcal{R} = \{R_i \mid i \in [k]\}$ each given with its starting and final positions; $k \in \mathbb{N}$.
Question: Compute a schedule for $\mathcal{R}$ of minimum cost (if one exists).

The GRID-COST μ-DISKS MOTION PLANNING (Grid-Cost-μ-Disk-MP) is defined analogously.

We note that the time complexity for solving the decision problems Moves-Rect-MP and Grid-Moves-μ-Disk-MP will be essentially the same (up to a polynomial factor) as that for solving its optimization version (where we seek to minimize ℓ), as we can binary-search for the length of an optimal schedule.

We also study the RECTANGLES COORDINATED MOTION PLANNING problem (Moves-Rect-CMP) and DISKS COORDINATED MOTION PLANNING problems (Grid-Moves-μ-Disks-CMP) and their cost counterparts the COST RECTANGLES COORDINATED MOTION PLANNING (Cost-Rect-CMP) and COST DISKS COORDINATED MOTION PLANNING problems (Grid-Cost-μ-Disks-CMP), which are defined analogously with the only difference being that the moves are coordinated. More specifically, the schedule of the robots consists of a sequence of coordinated collision-free moves.

We will use the following GRID-TILING problem and its varian for our hardness results. GRID-TILING [6]: Given $k, n \in \mathbb{N}$ and k^2 nonempty sets S_{ij}, $i, j \in [k]$, where each $S_{ij} \subseteq [n] \times [n]$, decide if we can choose a $p_{ij} = (x_{ij}, y_{ij}) \in S_{ij}$, for each $(i, j) \in [k] \times [k]$, such that

- for each $i \in [k]$, $y_{i1} = y_{i2} = \cdots = y_{ik}$, and
- for each $j \in [k]$, $x_{1j} = x_{2j} = \cdots = x_{kj}$.

That is, the y-coordinates of the p_{ij}'s are the same in any row and the x-coordinates are the same in any column. The INCREASING GRID-TILING problem is defined similarly with the exception that we require the sequence of the x-coordinates of $p_{1j}, \dots, p_{kj}$ and the sequence of y-coordinates of $p_{i1}, \dots, p_{ik}$ be non-decreasing as opposed to being equal. Both versions of the problem are NP-hard and W[1]-hard parameterized by k [6].

We conclude the preliminaries section by giving upper bounds in k on the length (i.e., number of moves) of an optimal schedule for feasible instances of the problems under consideration. These upper bounds will be used to obtain FPT algorithms for these problems.

Proposition 1 *For any instance of* Cost-Rect-MP*,* Grid-Cost-μ-Disk-MP*,* Cost-Rect-CMP*, and* Grid-Cost-μ-Disks-CMP*, there is an optimal schedule for the instance of length at most* $2k \cdot 5^{k(k-1)}$*.*

3 W[1]-hardness of Moves-Rect-CMP With Obstacles

In this section, we consider the setting where, besides the robots, the plane may contain stationary obstacles. The obstacles are axis-aligned rectangles, and the collision between a robot and an obstacle is defined in the natural way, treating the obstacle as a stationary robot. We show that Moves-Rect-CMP with obstacles is W[1]-hard, contrasting the FPT result for the same problem in the free plane [25].

Theorem 2 Moves-Rect-CMP *with rectangular obstacles is* W[1]*-hard parameterized by the number* k *of robots.*

Proof. We reduce from an instance $(k, n, (S_{ij})_{i,j \in [k]})$ of Grid-Tiling. The produced instance of Moves-Rect-CMP has $2k$ rectangular robots. There are k robots $V_1, \dots, V_k$ that we refer to as "vertical", which will be responsible for selecting the x-coordinate in each of k columns of the $k \times k$ grid from the Grid-Tiling instance. Each vertical robot has width 1 and height 2. Similarly, there are k "horizontal" robots $H_1, \dots, H_k$ responsible for selecting the y-coordinate in each of the k rows of the $k \times k$ grid. Each horizontal robot has width 2 and height 1. The goal is now to create an environment, using rectangular obstacles, that encodes the Grid-Tiling instance (see Figure 1). The upper bound

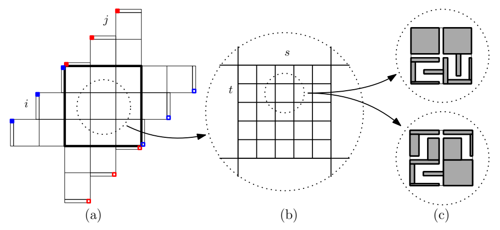


Figure 1: A high-level overview of the constructed instance of Moves-Rect-CMP in Theorem 2 starting from the Grid-Tiling instance with $k = 3$ and $n = 5$.

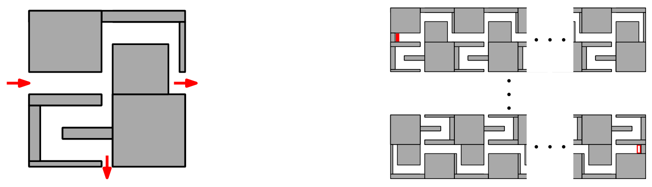


Figure 2: An illustration of a vertical selection gadget.

on the total number of moves is equal to the minimum number of moves each robot needs to reach its destination. The core of the reduction is a $k \times k$ grid (see the bold-boundaried square in the center of Figure 1-(a)). Each of the k^2 cells of this grid is further divided into an $n \times n$ grid, whose cells represent the pairs in the S_{ij}. We assume that the rows are numbered from top to bottom and the columns from left to right; so the t-th row from the top and the s-th column from the left represent the pair (s, t) and depending on whether $(s, t) \in S_{ij}$ or not, the cell contains an "allow-intersection gadget" (Figure 1-(c) bottom) or a "forbid-intersection gadget" (Figure 1-(c) top). To ensure that the horizontal robot H_i and the vertical robot V_j enter the cell at the same time, the starting position of H_i (resp. V_j) is shifted left by $i - 1$ (resp. up by $j - 1$) auxiliary $n \times n$ grids, which are outside of the central $k \times k$ big grid, and hence robots do not cross in these auxiliary grids (see Figure 1-(a)). Each robot then needs to fully pass through $2k - 1$ many $n \times n$ grids. Finally, robots start and end in a selection gadget; see Figure 2 for the vertical selection gadget (the horizontal selection gadget is analogous). The vertical (horizontal) selection gadgets are the only places that allow vertical (horizontal) robots to change their horizontal (vertical) position. That is, a vertical robot with height 2 and width 1 can enter each cell of the top part of the vertical selection gadget from the left and either leave it from the right after 4 moves, or from the bottom after 5 moves. The bottom (top) of the i-th cell in the top (bottom) part of the selection gadget connects directly to an intersection gadget (one of the two possible gadgets in Figure 1-(c)) in the first row and the i-th column of an $n \times n$ grid. Each intersection gadget can be passed by a vertical robot only from top to bottom in 4 moves; a vertical robot does not fit through the left/right entrance of the gadget. Similarly, a horizontal robot can pass through an intersection gadget in 4 moves from left to right, but cannot enter/exit from the top/bottom entrances. The above finishes the description of the gadgets. The total number of moves is then set to $4(n + 1) + 4(2k - 1)n + 3$. Each robot needs $4(2k - 1)n$ moves to pass through $(2k - 1)n$ many intersection gadgets in $2k - 1$ many $n \times n$ grids, and $4(n + 1) + 3$ moves to pass through the selection gadget. For the selection gadget, if the robot leaves from the s-th cell, then it is easy to see from Figure 2 that it makes

$4s+1$ moves in the top part (its $(4s+2)$-nd move counts towards the cell it enters) and $4(n+1-s)+2$ moves in the bottom part of the selection gadget. Hence, robots have to move towards their destinations in every step.

We now show that the original GRID-TILING instance is equivalent to the produced instance of Moves-Rect-CMP with obstacles and with $\lambda = 4(n+1)+4(2k-1)n+3$. We already argued that, in a schedule with an upper bound of λ on the total number of moves, each vertical robot V_j selects a single column in the selection gadgets and then passes through all the intersection gadgets in that column; this selection clearly corresponds to selecting the same x-coordinate in each cell of the j-th column of the $k \times k$ grid (i.e., $x_{1j} = x_{2j} = \cdots = x_{kj}$). Similarly, the horizontal robot H_i picks a single row, effectively ensuring that $y_{i1} = y_{i2} = \cdots = y_{ik}$. To prove the correctness of the reduction, it suffices to show that H_i can pick row t and V_j column s if and only if $(s,t) \in S_{ij}$. For that, let us assume that they pick t and s, respectively, and let us compute the move in which they enter the intersection gadget representing the position (s,t) in the $n \times n$ grid for S_{ij}, where their two paths intersect. First, for V_j to select s, it does $4s+1$ moves in the selection gadget. Afterwards, it does $4(j-1)n$ moves through the auxiliary grids to enter the central $k \times k$ grid. There it does $4 \cdot (i-1) \cdot n$ moves to pass first the $i-1$ many $n \times n$ grids and finally, it does $4 \cdot (t-1)$ moves to reach the top entrance of the corresponding intersection gadget after $4(i+j-2) \cdot n + 4 \cdot (s+t) - 3$ moves. Symmetrically, it is easy to see that H_i is at the left entrance of the intersection gadget after $4(i+j-2) \cdot n + 4 \cdot (s+t) - 3$ many moves as well. It is easy to see that only the allow-intersection gadget (Figure 1-(c) bottom) allows both robots to enter at the same time. This is because vertical robots can pass the crossing point in the same move in which it enters the gadget, while for horizontal robots it is their third move. On the other hand, in the forbid-intersection gadget (Figure 1-(c) top) the crossing is in the first move for both robots and at the same distance from the entrance, and hence there will be a collision. It follows that the instance of Moves-Rect-CMP with obstacles admits a schedule with a total number of moves at most λ if and only if the instance of GRID-TILING is a Yes-instance. □

4 On the Cost-Rect-MP Problem

It is not difficult to prove the NP-hardness of Cost-Rect-MP via a reduction from the NP-hard (n^2-1)-puzzle problem [31]. In this section, we present the NP-hardness of Cost-Rect-MP restricted to instances in which each robot must travel along a shortest rectilinear path between its starting and ending positions, denoted Shortest Cost-Rect-MP, which is an important restriction in its own right, and implies the NP-hardness of an important variant of the classical COORDINATED MOTION PLANNING ON GRIDS as well as other problems in this paper:

Theorem 3 SHORTEST COST-RECT-MP *is* NP*-hard.*

We also give FPT algorithms for Cost-Rect-MP and Cost-Rect-CMP, which follow exactly the algorithms given in [24] for Moves-Rect-MP and Moves-Rect-CMP, with a minor tweak of changing the linear program to include an objective function for minimizing the cost, which is a linear function. The running time of the algorithms for Cost-Rect-MP and Cost-Rect-CMP is $\mathcal{O}^*(5^{k^2} \cdot 8^{2k^2 \cdot 5^{k^2}} \cdot 5^{k^3 \cdot 5^{k^2}})$ (see [25]).

5 The FPT Algorithms for Grid-Moves-μ-Disk-MP and Grid-Cost-μ-Disk-MP

The NP-hardness of Grid-Moves-μ-Disk-MP follows from the NP-hardness of the restriction of the problem to instances in which the disks are points (i.e., the radius μ of the disks is zero), proved in [5].

We describe the FPT algorithms for Grid-Moves-μ-Disk-MP and Grid-Cost-μ-Disk-MP. The algorithms are similar to those in the previous section, with the exception that we no longer can use linear programming since the motion must be confined to the grid points (as opposed to the whole plane). Instead, we use Integer Linear Programming (ILP), and that poses quite some complications. To be able to use ILP and obtain fixed-parameter tractability, we resort to Lenstra's result [19, 20, 26], which requires that the number of variables in the ILP instance be upper bounded by a function of the parameter k. We describe the algorithm for Grid-Moves-μ-Disk-MP; the algorithm for Grid-Cost-μ-Disk-MP is exactly the same except for adding a minimization objective linear function. Since the upper bound on the running time in Lenstra's result [19, 20, 26] is not practical, we omit discussing the running time of our algorithm and concern ourselves with showing that it is FPT.

We start by guessing the number of moves, λ, in an optimal schedule, which was shown in Section 2 to be upper bounded by a function of k. Next, the algorithm guesses in each step which robot moves and its direction. We create ILP variables to encode the position of each robot in each step, and the amplitude of the translation vector in each move. Since λ and the number of possible directions for each move are upper bounded by a function of k and a constant, respectively, the number of ILP variables is upper bounded by a function of k. What is left is showing that we can add linear constraints to stipulate that the moves are collision-free.

Let R be a disk robot whose center is at (x_0, y_0), and suppose that in a certain move R is translated horizontally by a vector $\overrightarrow{v}$ of amplitude α in the direction $\overrightarrow{H}^+$;

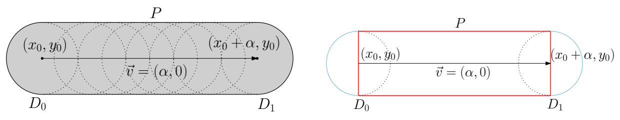


Figure 3: Left: The trace of disk D_0 w.r.t. a translation by a vector $\overrightarrow{v} = (\alpha, 0)$. Right: The decomposition of the trace into the rectangle P (red) and the two disks D_0 and D_1.

other directions are analogous. At the translation, the center of R is at the point $(x_0 + \alpha, y_0)$. During the motion of R, it covers a region of the plane, denoted *trace*$(R, \overrightarrow{v})$, depicted in Figure 3 (left). For any robot $R' \neq R$, located at point (a, b), R does not collide with R' during this translation iff the disk for R' centered at (a, b) does not intersect the interior of $\mathsf{trace}(R, \overrightarrow{v})$. Note that x_0, y_0, α, a, b are all variables in the ILP instance. To stipulate this non-collision condition, we decompose $\mathsf{trace}(R, \overrightarrow{v})$ into three (non-disjoint) regions (see the right side of Figure 3): the two disks D_0 centered at (x_0, y_0) and D_1 centered at $(x_0 + \alpha, y_0)$ of radius μ, and the interior of rectangle P whose two diametrically-opposite corners/vertices are $(x_0, y_0 + \mu)$ and $(x_0 + \alpha, y_0 - \mu)$. Clearly, R' does not intersect $\mathsf{trace}(R, \overrightarrow{v})$ if and only if it does not intersect any of these three regions. Consider first rectangle P. To stipulate that R' does not intersect the interior of P, we first observe that if R' intersects a vertical line segment of P then it must intersect D_0 or D_1. Since we will later add constraints to stipulate that R' does not intersect D_0 or D_1, it suffices at this point to add constraints to enforce that the interior of R' does not intersect any of the two horizontal line segments of P. We first make a guess for if R' intersects a vertical slab B defined by P such that R' intersects P in B iff it intersects a horizontal segment. If not, it can make intersection with P only around D_0 and D_1, in the other case we do as follows. Observe that if R' does not intersect a horizontal segment of P then (a, b) must be either above the top horizontal segment of P, or below the bottom horizontal segment of P, and in either case, at a distance of at least μ from the corresponding line. We guess which of the two cases holds and add a linear constraint accordingly. For example, if the guess is that (a, b) is above the top horizontal line of P, we add the linear constraint: $b - \mu \geqslant y_0 + \mu$, or equivalently, $b - y_0 \geqslant 2\mu$. The case is similar for the other guess. Since we make two guesses per rectangle R', the number of guesses is FPT for this part. Next, we encode non-collision between R' and D_0; the treatment is analogous for D_1. Since R' and D_0 are congruent disks of radius μ, these disks collide if and only if the distance between their centers is smaller than 2μ. Since μ is a constant and the grid is a unit grid, the number of grid points whose distance from the

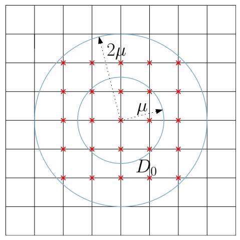


Figure 4: The grid points corresponding to the centers of disks that collide with D_0.

center (x_0, y_0) of D_0 is smaller than 2μ is a function of μ, this is a constant, and this will be true even after adding some points outside of the vertical slab near the D_i, and those points have coordinates that are linear in x_0 and y_0, and can be enumerated in constant time; see Figure 4. Therefore, we add constraints to enforce that the center (a, b) is not equal to any of these points. To do so, note that two points are not equal if one of the two coordinates of one of the two points is less than the corresponding coordinate of the other point. For any of the candidate points (c, d), we guess which of the four cases holds (i.e., which of the coordinates of (a, b) and (c, d) is less than the corresponding one) and add linear constraints stipulating that. The number of guesses and variables introduced in this part are FPT.

We constructed an ILP instance that can be solved in FPT-time using the results in [19, 20, 26]. Since the number of guesses is FPT, the above nondeterministic algorithm can be simulated in deterministic FPT-time:

Theorem 4 *The* GRID-MOVES-μ-DISK-MP *and the* GRID-COST-μ-DISK-MP *problems are* FPT *parameterized by* k.

We can extend the above FPT algorithms to Grid-Moves-μ-Disks-CMP and Grid-Cost-μ-Disks-CMP.

Theorem 5 *The* GRID-MOVES-μ-DISKS-CMP *and the* GRID-COST-μ-DISKS-CMP *problems are* FPT *parameterized by* k.

6 Concluding Remarks

A couple of natural open questions ensue from our work:

- Can we extend the $\mathsf{W}[1]$-hardness result of Moves-Rect-CMP where obstacles are present to Moves-Rect-MP (i.e., where the motion is serial)? We note that the coordinated motion was a very essential ingredient in the $\mathsf{W}[1]$-hardness proof for Moves-Rect-CMP.

- Can we extend the FPT algorithms for disk robots confined to a grid, given in this paper, to the case where the disk robots move in the free plane?

References

[1] Manuel Abellanas, Sergey Bereg, Ferran Hurtado, Alfredo García Olaverri, David Rappaport, and Javier Tejel. Moving coins. *Computational Geometry*, 34(1):35–48, 2006.

[2] Pankaj K. Agarwal, Dan Halperin, Micha Sharir, and Alex Steiger. Near-optimal min-sum motion planning for two square robots in a polygonal environment. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4942–4962, 2024.

[3] Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. Intractability of time-optimal multirobot path planning on 2D grid graphs with holes. *IEEE Robotics and Automation Letters*, 2(4):1941–1947, 2017.

[4] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, pages 740–746, 2015.

[5] Gruia Călinescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. *SIAM Journal on Discrete Mathematics*, 22(1):124–138, 2008.

[6] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[7] Argyrios Deligkas, Eduard Eiben, Robert Ganian, Iyad Kanj, and M. S. Ramanujan. Parameterized algorithms for coordinated motion planning: Minimizing energy. In *51st International Colloquium on Automata, Languages, and Programming, ICALP*, volume 297 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[8] Argyrios Deligkas, Eduard Eiben, Robert Ganian, Iyad Kanj, and M. S. Ramanujan. Parameterized algorithms for multiagent pathfinding on trees. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025)*. ACM, 2025.

[9] Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019.

[10] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[11] Adrian Dumitrescu. Mover problems. In János Pach, editor, *Thirty Essays on Geometric Graph Theory*, pages 185–211, New York, NY, 2013. Springer New York.

[12] Adrian Dumitrescu and Minghui Jiang. On reconfiguration of disks in the plane and related problems. *Computational Geometry*, 46(3):191–202, 2013.

[13] Eduard Eiben, Robert Ganian, and Iyad Kanj. The parameterized complexity of coordinated motion planning. In *Symposium on Computational Geometry*, volume 258, pages 28:1–28:16, 2023.

[14] Eduard Eiben, Robert Ganian, Iyad Kanj, and M. S. Ramanujan. A minor-testing approach for coordinated motion planning with sliding robots. In *Symposium on Computational Geometry*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[15] Guillermo Esteban, Dan Halperin, Víctor Ruíz, Vera Sacristán, and Rodrigo I. Silveira. Shortest coordinated motion for square robots. In *Proceedings of the 18th International Symposium on Algorithms and Data Structures (WADS)*, volume 14079 of *Lecture Notes in Computer Science*, pages 430–443. Springer, 2023.

[16] Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing coordinated motion plans for robot swarms: The CG: SHOP challenge 2021. *ACM J. Exp. Algorithmics*, 27:3.1:1–3.1:12, 2022.

[17] Foivos Fioravantes, Dusan Knop, Jan Matyás Kristan, Nikolaos Melissinos, and Michal Opler. Exact algorithms and lowerbounds for multiagent path finding: Power of treelike topology. In *Thirty-Eighth AAAI Conference on Artificial Intelligence*, pages 17380–17388, 2024.

[18] Foivos Fioravantes, Dusan Knop, Jan Matyás Kristan, Nikolaos Melissinos, and Michal Opler. Exact algorithms for multiagent path finding with communication constraints on tree-like structures. *To appear in AAAI 2025*, abs/2412.08556, 2024. URL: `https://doi.org/10.48550/arXiv.2412.08556`, `arXiv:2412.08556`, `doi:10.48550/ARXIV.2412.08556`.

[19] András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[20] Jr. H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[21] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[22] John E. Hopcroft, Jacob T. Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "Warehouseman's Problem. *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[23] John E. Hopcroft and Gordon T. Wilfong. Reducing multiple object motion planning to graph searching. *SIAM Journal on Computing*, 15(3):768–785, 1986.

[24] Iyad Kanj and Salman Parsa. On the parameterized complexity of motion planning for rectangular robots. In *40th International Symposium on Computational Geometry, SoCG 2024, June 11-14, 2024, Athens, Greece*, volume 293 of *LIPIcs*, pages 65:1–65:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[25] Iyad Kanj and Salman Parsa. On the parameterized complexity of motion planning for rectangular robots, 2024. URL: https://arxiv.org/abs/2402.17846, arXiv:2402.17846.

[26] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.

[27] David G. Kirkpatrick and Paul Liu. Characterizing minimum-length coordinated motions for two discs. In *Proceedings of the 28th Canadian Conference on Computational Geometry, CCCG*, pages 252–259, 2016.

[28] Daniel Kornhauser, Gary Miller, and Paul Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pages 241–250, 1984.

[29] Hang Ma and Sven Koenig. AI buzzwords explained: Multi-agent path finding (mapf). 3(3):15–19, 2017.

[30] Geetha Ramanathan and Vangalur S. Alagar. Algorithmic motion planning in robotics: Coordinated motion of several disks amidst polygonal obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 514–522, 1985.

[31] Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.

[32] Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: III. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2:46 – 75, 1983.

[33] Jacob T. Schwartz and Micha Sharir. On the "piano movers'" problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on pure and applied mathematics*, 36(3):345–398, 1983.

[34] Jacob T. Schwartz and Micha Sharir. On the "piano movers" problem. II. General techniques for computing topological properties of real algebraic manifolds. *Advances in applied Mathematics*, 4(3):298–351, 1983.

[35] Michael Ian Shamos. *Computational geometry.* PhD thesis, USA, 1978.

[36] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[37] Irving Solis, James Motes, Read Sandström, and Nancy M. Amato. Representation-optimal multi-robot motion planning using conflict-based search. *IEEE Robotics Autom. Lett.*, 6(3):4608–4615, 2021.

[38] Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. *The International Journal of Robotics Research*, 35(14):1750–1759, 2016.

[39] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

Optimal Delivery with a Faulty Drone

Jared Coleman[*] Evangelos Kranakis[†] Danny Krizanc[‡] Oscar Morales-Ponce[§]

Abstract

We introduce and study a new cooperative delivery problem inspired by drone-assisted package delivery. We consider a scenario where a drone moving in a straight line towards its destination loses communication (at time $t = 0$) with its central command station. The command station cannot know whether the drone's system has wholly malfunctioned or merely experienced a communications failure and the drone is continuing towards its destination. Consequently, a second helper drone with the same speed is deployed from the command station to retrieve the package and ensure successful delivery should the first drone fail before reaching its destination. The central question of this study is to find an algorithm for this second drone with optimal competitive ratio with respect to an omniscient drone that knows exactly if and when the first drone will fail. We show that the optimal algorithm depends on a surprisingly intricate relationship between the relative initial positions of the two drones and the destination.

1 Introduction

In recent years, drone-based package delivery has emerged as a promising application of unmanned aerial vehicle (UAV) technology. As these systems are increasingly integrated into supply chain infrastructures, it becomes imperative to design algorithms for their robust operation amidst unexpected complications. In this paper, we consider a complication arising from faulty communication and propose a solution for a cooperative delivery problem inspired by this scenario.

Consider a situation where a drone, en route to deliver a package to a given destination, unexpectedly loses communication with its central command station. This unexpected loss of contact leaves the command station uncertain of whether the drone has suffered a communications breakdown or complete system failure. Furthermore, even if the issue *is* only with the communications, the command station no longer has any way of knowing if/where the drone will fail on the rest of its way to the destination. In order to guarantee the package gets delivered, the command station must dispatch a second helper drone to retrieve the package and complete the delivery. Our goal is to design an online algorithm (one that cannot anticipate the true fail location of the drone) that, given the drone's last known location, determines the best trajectory for the second drone to find the package and complete the delivery in minimal time.

Formally, let us denote the last known location of the drone as the origin $S = (0, 0)$, the destination as the point $T = (1, 0)$, and the location of the command station as $P = (x, y)$, where $y \geq 0$ (all without loss of generality). The task is to identify an optimal trajectory for the second drone that minimizes the competitive ratio when compared to an optimal offline algorithm that knows the exact failure location $(t, 0)$ of the first drone in advance. We assume the first drone will fail at some time $0 \leq t \leq 1$ (if it does not fail the delivery time is optimal and the problem is uninteresting). Also, to simplify notation, we only consider the package to be delivered once the *second* drone and the package are co-located at the destination (i.e., if $t = 1$ and the first drone fails *at* the destination, the package is only delivered once the second drone reaches T). For an algorithm $\mathcal{A}(x, y)$, which defines the trajectory of the second drone, let $A(x, y, t)$ denote the delivery time of algorithm $\mathcal{A}(x, y)$ for failure time t. Our goal is to find an online algorithm (where t is unknown) with minimum competitive ratio with respect to the delivery time of an optimal offline algorithm, $\mathrm{Opt}(x, y, t)$ (where t is known ahead of time). The competitive ratio for algorithm $\mathcal{A}(x, y)$ for a given fail time t can be written as

$$\mathrm{CR}_{\mathcal{A}(x,y)}(t) = \frac{A(x, y, t)}{\mathrm{Opt}(x, y, t)}.$$

Then the competitive ratio of $\mathcal{A}(x, y)$ is

$$\mathrm{CR}_{\mathcal{A}(x,y)} = \sup_t \mathrm{CR}_{\mathcal{A}(x,y)}(t).$$

To simplify notation we sometimes eliminate x and y when they are clear from context and write algorithm $\mathcal{A}(x, y)$ as $\mathcal{A}$ and its delivery time $A(x, y, t)$ as $A(t)$.

1.1 Model and Notation

In this section, we describe the model and notation used throughout the paper. We call the first drone, which is

[*]Loyola Marymount University, jared.coleman@lmu.edu
[†]Carleton University, kranakis@scs.carleton.ca
[‡]Wesleyan University, dkrizanc@wesleyan.edu
[§]California State University, Long Beach, Oscar.MoralesPonce@csulb.edu

initially carrying the package towards the destination, the *starter* and the second drone, which completes the delivery, the *finisher*. Without loss of generality, let $S = (0, 0)$ be the initial location of the starter (the point where it loses communication with the central command station) and $T = (1, 0)$ be the destination point on the plane. The starter drone begins at point S carrying the package and moves following a straight line directly towards point T. At some unknown time $t \le 1$, the starter will fail at position $(t, 0)$.

The finisher drone starts at a point $P = (x, y)$ on the plane (i.e., at the command station). Without loss of generality, we assume that $y \ge 0$ (all results follow trivially by symmetry). We assume that both drones have a speed of 1 and always move at this speed. The finisher can start, stop, and change direction instantaneously. The drones can communicate with each other and exchange the package only when they are co-located (face-to-face communication). The package is considered to be delivered as soon as the finisher and the package are co-located at T.

Let $D(c, r)$ (resp., $\overline{D}(c, r)$) denote the open (resp., closed) disk with radius r and center $(c, 0)$. We use capital letters to denote points, $|PQ|$ to denote the Euclidean distance between points P and Q, and $\overline{PQ}$ to denote the line segment with endpoints P and Q. To simplify notation we sometimes eliminate x and y when they are clear from context, including when writing derived quantities such as d, t_1, and others that are functions of (x, y).

1.2 Related Work

An important aspect of our problem is that the starter agent experiences a failure. Many problems with cooperative mobile agent experiencing failures have been studied for a variety of basic problems in distributed computing and in various domains. In [11], the authors study the search problem on the line by n mobile agents where f of the agents are faulty. They present algorithms and their competitive ratios for different values of f. In [13] the authors study the evacuation problem on the disk by n agents, f of which may be faulty. Competitive algorithms for gathering have been proposed for n agents in a synchronous system with less than $(n-1)/3$ failures [1]. Optimal algorithms and hardness results for the multi-agent patrolling problem with faulty agents were presented in [9]. Algorithms for flocking [16] and evacuation [10] have also been studied in the context of faulty agents.

The problem studied in this paper has both search and delivery components. Many search problems have been studied for different domains and under different models and assumptions (cf. the book [2]). The authors of [6] consider the delivery problem for messages on the line segment by multiple agents with different speeds and propose competitively optimal algorithms. The results of the previous study were also extended to the plane [5]. Joint search and delivery problems, however, have received much less attention in the literature. The problem has also been studied for the single- and two-agent case on the line [4]. On the systems research side, many solutions for drone-assisted package delivery have been proposed for different environments and under different assumptions (we refer the reader to the survey [14]). There are also numerous studies on drone reliability. E.g., in [15] the authors consider the reliability of drones in a delivery network so as to minimize expected loss of demand assuming drone failures follow an exponential distribution and study the impact of including reliability in drone scheduling.

Most related but different to our delivery problem are the papers [3] and [12] on package delivery. In the former, the authors investigate delivery of one or two packages of many autonomous mobile agents initially located on distinct nodes of a weighted graph. In the latter paper, they are concerned with delivering a package from a source node to a destination node in a graph using a set of drones and study the setting where the movements of each drone are restricted to a certain subgraph of the given graph. Note that both papers above address the delivery problem in a graph setting. To the best of our knowledge our paper is the first to address delivery of a package in the plane in the presence of a faulty drone.

1.3 Results

The main result we present is an optimal online algorithm that depends only on the starting position of the finisher (the central command station). Essentially, the algorithm executes one of three candidate algorithms depending on the finisher's starting position. This is depicted in Figure 1. If the finisher starts in the diagonally striped region, then the optimal algorithm is for the finisher to go to S (the origin) and then towards the destination until finding the failed starter with the object. If the finisher starts in the vertically striped region, the optimal algorithm is for the finisher to go first to T (the destination) and then toward S until finding the failed starter with the object. Finally, if the finisher starts in the horizontally striped region, then the optimal algorithm is for the starter to go to the point $M = \left(\left(x^2 + y^2\right)/(2x), 0\right)$ and then toward S until finding the failed starter with the object. We will show that the finisher is guaranteed to find the starter on the interval $[0, M]$. While the behavior of the finisher is simple, the separating curves that mark the boundaries between these regions are non-linear and quite surprising. Intricate details of these curves are depicted in Figures 4, 5, and 6 (Section 4).

The rest of the paper is organized as follows.

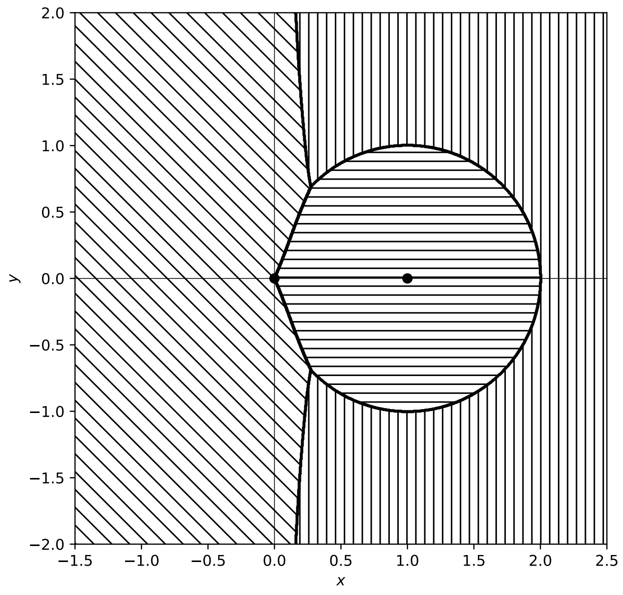


Figure 1: The optimal algorithm with $S = (0,0)$ and $T = (1,0)$ depends on the starting position of the finisher. The striped regions depict the finisher starting positions for which each of the three candidate algorithms is optimal.

Section 2 introduces three candidate algorithms and derives their competitive ratios as functions of the command station's starting position. Section 3 presents a hybrid algorithm that selects the best candidate based on the starting position (x, y); we prove this hybrid is optimal. Section 4 examines how the starting position influences the chosen candidate within the hybrid. Finally, Section 5 summarizes the results and outlines future directions. All proofs omitted due to space constraints can be found in the full version of the paper [8]. Some proofs rely on computations performed in Mathematica, which are publicly available on GitHub [7].

2 Candidate Algorithms

In this section, we present three algorithms and derive their competitive ratios. In order to do so, we must first consider the optimal offline algorithm, where the fail location $(t, 0)$ (at time t) is known ahead of time and can be used to compute the optimal trajectory for the finisher. Clearly in this case the finisher should go directly from its starting location to the starter's fail location $(t, 0)$ and then complete the delivery. The delivery time then can be written:

$$\text{Opt}(t) = \max\left\{1, \sqrt{(x-t)^2 + y^2} + 1 - t\right\}.$$

Indeed, although the finisher moves directly to $(t, 0)$, it may have to wait for the starter to arrive (if necessary) prior to completing the delivery task.

For the online algorithms, we start by reasoning about what an optimal algorithm looks like. First, since the starter must fail at some point on the line segment $\overline{ST}$, the finisher must *eventually* move from its initial location to some point $(m, 0)$ on the segment (otherwise it will never find the starter with the package). We use this to prove the following intuitively obvious lemma:

Lemma 1 *There exists an online algorithm with optimal competitive ratio that involves the finisher moving from its initial position $P = (x, y)$ directly to a point $M = (m, 0) \in \overline{ST}$, past which, it remains within the line segment $\overline{ST}$.*

Now, we present three candidate online algorithms:

1. $\mathcal{A}_0$ (**Go To Last Point of Contact**): The finisher moves to the origin $S = (0,0)$ and then towards the destination until it finds the failed starter (trajectory $P \to S \to T$).

2. $\mathcal{A}_1$ (**Go To Destination**): The finisher goes first to the destination $T = (1,0)$ and then towards the origin until it finds the failed starter (trajectory $P \to T \to S \to T$).

3. $\mathcal{A}_d$ (**Meet in the Middle**): The finisher goes first to the point $(d, 0)$, where $d = (x^2 + y^2)/(2x)$, and then towards the origin until it finds the failed starter (trajectory $P \to (d,0) \to S \to T$). The point $(d, 0)$ is the unique point on the line segment where the drones, both moving continuously, would meet simultaneously if the starter does not fail before time d.

Lemma 2 $CR_{\mathcal{A}_d(x,y)} \leq CR_{\mathcal{A}_1(x,y)}$ *if and only if the finisher starts within distance* 1 *of the destination (i.e., $(x, y) \in \overline{D}(1,1)$).*

Lemma 2 essentially tells us that we need only consider (among the candidate algorithms) $\mathcal{A}_0$ and $\mathcal{A}_d$ when the finisher starts *inside* the disk and algorithms $\mathcal{A}_0$ and $\mathcal{A}_1$ when the finisher starts *outside* the disk. Note that, when (x, y) is on the edge of the disk $\overline{D}(1,1)$ (i.e., $(x-1)^2 + y^2 = 1$), then $d = 1$ and so $\mathcal{A}_d$ and $\mathcal{A}_1$ are the same algorithm.

2.1 Go to Last Point of Contact

In this section, we derive the competitive ratio of Algorithm $\mathcal{A}_0$. Observe that, no matter the failure time of the starter, the algorithm takes time $1 + \sqrt{x^2 + y^2}$ to deliver the message. This fact makes deriving the algorithm's competitive ratio rather simple.

Theorem 3 *The competitive ratio of $\mathcal{A}_0$ is*

$$CR_{\mathcal{A}_0} = \frac{1+\sqrt{x^2+y^2}}{\max\left\{1, \sqrt{(x-1)^2+y^2}\right\}}.$$

2.2 Go to Destination

We will now derive the competitive ratio of algorithm $\mathcal{A}_1$. Unlike the delivery time of algorithm $\mathcal{A}_0$, the delivery time $A_1(t) = \sqrt{(x-1)^2+y^2} + 2(1-t)$ (the time for the agent to go to $(1,0)$ and then backtrack until it finds the package at $(t,0)$ before returning to $(1,0)$ to complete the delivery) depends greatly on the fail time of the starter. Thus, to find the competitive ratio of algorithm $\mathcal{A}_1$, we must find the worst-case fail time for a given finisher starting position (x,y).

Theorem 4 *Assume $(x,y) \notin D(1,1)$ (i.e., $\sqrt{(x-1)^2+y^2} \geq 1$). The competitive ratio of $\mathcal{A}_1$ is $CR_{\mathcal{A}_1}(\max\{t_1, 0\})$ where*

$$t_1 = \begin{cases} 1 - 3y/4 & x = 1 \\ \frac{x^2+y^2+z_1(1-x)-1-z_1\sqrt{x(x+z_1-2)+y^2-z_1+1}}{2(x-1)} & o.w. \end{cases}$$

and $z_1 = \sqrt{(x-1)^2+y^2}$.

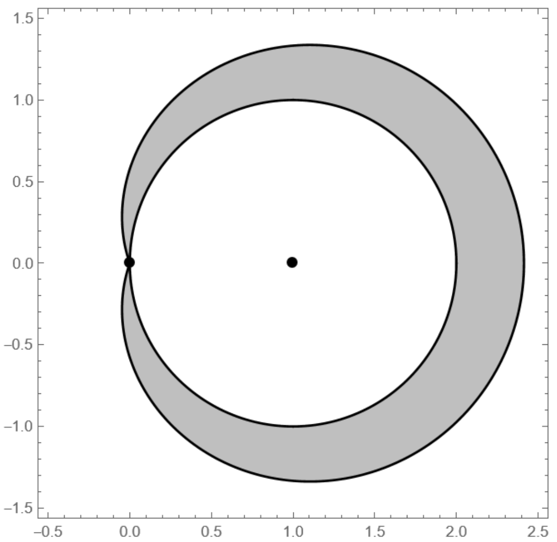


Figure 2: The worst-case fail time for the starter is 0 *except* in the gray shaded region (defined in the proof of Theorem 4, which can be found in the full version of the paper [8]). Recall that we only consider $\mathcal{A}_1$ when the finisher starts outside of the Disk $D(1,1)$.

There are essentially two cases that drive the competitive ratio of Algorithm $\mathcal{A}_1$: when t_1 (from the statement of Theorem 4) is less than or equal to 0 and when it is greater than 0. Observe the value of t_1 is determined by the finisher's starting position (x,y). When $t_1 \leq 0$ (i.e., when the finisher starts outside of the gray region shown in Figure 2), the competitive ratio is driven by the case where the starter fails at the origin (as soon as it loses contact with the command station). When $t_1 > 0$, though, the worst-case scenario for $\mathcal{A}_1$ is when the starter fails at time t_1.

2.3 Meet in the Middle

In this section, we derive the competitive ratio of the last candidate algorithm, $\mathcal{A}_d$, where the finisher moves from its starting position to the point $d = (x^2+y^2)/(2x)$, then towards the origin until it finds the starter with the package before completing the delivery. Recall from Lemma 2 that we need only consider algorithm $\mathcal{A}_d$ inside the closed disk centered at $(1,0)$ with radius 1, namely $\overline{D}(1,1)$. Furthermore, observe that on the edge of this disk $d=1$ and so algorithms $\mathcal{A}_d$ and $\mathcal{A}_1$ are equal. By construction, then, the finisher *must* find the starter between the origin and the point $(d,0)$ since $(d,0)$ is the unique point on $\overline{ST}$ such that the distance between the origin and $(d,0)$ is equal to the distance from (x,y) to 0 (i.e., it is the point where the two drones would meet simultaneously if the starter does not fail). It is easy to derive that $d = (x^2+y^2)/(2x)$ by solving $\sqrt{(x-d)^2+y^2} = d$ for d using simple algebra. Thus, at time d, the finisher reaches point $(d,0)$ and the starter cannot have reached any point further than $(d,0)$.

Theorem 5 *Assume $(x,y) \in \overline{D}(1,1)$ (i.e., $\sqrt{(x-1)^2+y^2} \leq 1$). The competitive ratio of $\mathcal{A}_d$ is*

$$CR_{\mathcal{A}_d} = \begin{cases} \frac{x^2+y^2+x}{x\left(1+\sqrt{x^2+y^2}\right)} & \text{if } (x,y) \in \overline{D}(1/2,1/2) \\ 1 + \frac{y^2}{x\left(\sqrt{x}+1\right)^2} & \text{otherwise.} \end{cases}$$

Similar to Algorithm $\mathcal{A}_1$, two cases drive Algorithm $\mathcal{A}_d$'s competitive ratio. The first is when t' (from the proof of Theorem 5) is less than 0, whenever the starting position of the finisher is inside the disk $\overline{D}(1/2,1/2)$. In this case, the worst-case failure time for the starter is $t=0$ (i.e., at the origin). When the finisher starts outside of the disk $\overline{D}(1/2,1/2)$ (but inside the disk $\overline{D}(1,1)$, of course, since we only consider Algorithm $\mathcal{A}_d$ in this region), however, the worst-case failure time for the starter is $t=t'$ (i.e., at location $(t',0)$, see Figure 3).

3 A Hybrid Algorithm

For convenience, we summarize our main result by introducing Algorithm 1, which simply executes the best of $\mathcal{A}_0$, $\mathcal{A}_1$, and $\mathcal{A}_d$ given a finisher starting position (x,y), and prove it to be optimal.

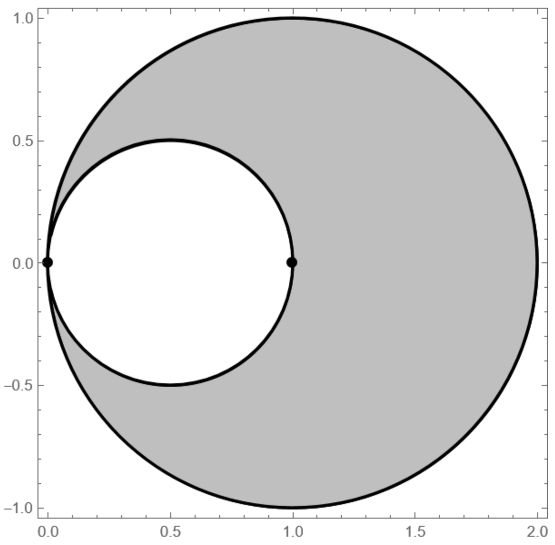


Figure 3: The worst-case fail time for the starter is 0 *except* in the gray shaded region, where it is t' (from the proof of Theorem 5). Recall that we only consider $\mathcal{A}_d$ when the finisher starts inside the Disk $\overline{D}(1,1)$.

Algorithm 1 A hybrid algorithm

```
1: input: Finisher starting position $(x, y)$
2: if $(x-1)^2 + y^2 > 1$ then
3:     if $\mathrm{CR}_{\mathcal{A}_0(x,y)} \leq \mathrm{CR}_{\mathcal{A}_1(x,y)}$ then
4:         Execute Algorithm $\mathcal{A}_0$
5:     else Execute Algorithm $\mathcal{A}_1$
6: else
7:     if $\mathrm{CR}_{\mathcal{A}_0(x,y)} \leq \mathrm{CR}_{\mathcal{A}_d(x,y)}$ then
8:         Execute Algorithm $\mathcal{A}_0$
9:     else Execute Algorithm $\mathcal{A}_d$
```

In fact, we will show that Algorithm 1 is optimal by proving that any other algorithm A_a that moves first to a position $(a, 0)$ such that $0 < a < 1$ and $a \neq d$ is always worse than at least one of the candidate algorithms $\mathcal{A}_0$, $\mathcal{A}_1$, or $\mathcal{A}_d$. Recall from Lemma 1 that we do not need to consider any other algorithms. Leveraging Lemma 2, we know that we only need consider $\mathcal{A}_0$ and $\mathcal{A}_d$ when the finisher's starting position (x, y) is inside the disk $\overline{D}(1,1)$ and algorithms $\mathcal{A}_0$ and $\mathcal{A}_1$ when it is outside the disk. The following lemmas cover each of these cases.

Lemma 6 *For any* $(x, y) \in \overline{D}(1,1)$, $\min\{CR_{\mathcal{A}_d}, CR_{\mathcal{A}_0}\} \leq CR_{\mathcal{A}_a}$ *for all* $a \in \overline{ST}$.

Again, by Lemma 2, when the finisher starts outside of the disk $D(1,1)$, we need only consider $\mathcal{A}_0$ and $\mathcal{A}_1$.

Lemma 7 *For any* $(x, y) \notin D(1,1)$, *either* $CR_{\mathcal{A}_1} \leq CR_{\mathcal{A}_a}$ *or* $CR_{\mathcal{A}_0} \leq CR_{\mathcal{A}_a}$ *for all* $a \in \overline{ST}$.

Theorem 8 *Algorithm 1 is optimal.*

Proof. Follows directly from Lemmas 6 and 7. □

4 Discussion

We now examine in more detail the regions where each candidate algorithm is optimal, as shown in Figures 4, 5, and 6. While the plots were generated numerically, we are able to obtain a closed-form equation for one of the separating curves.

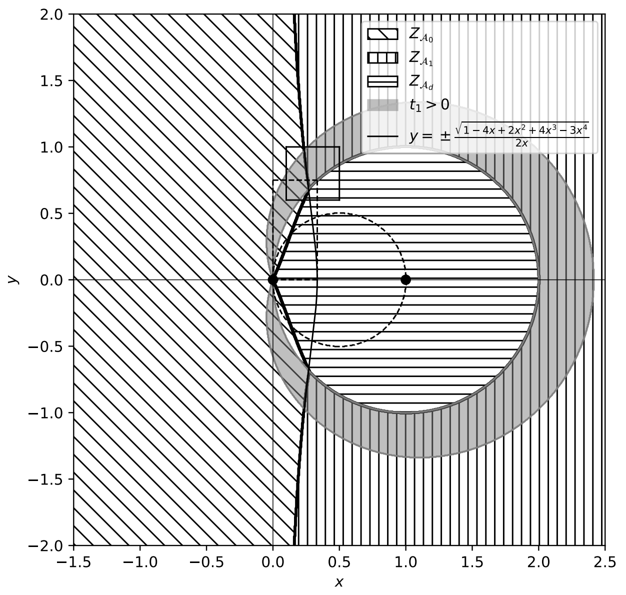


Figure 4: Plot showing which algorithm has the least competitive ratio given finisher starting position (x, y).

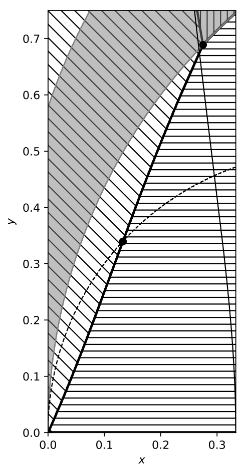


Figure 5: Region in Figure 4 bounded by dashed rectangular box.

The competitive ratio of each of the candidate algorithms, and therefore the hybrid algorithm, depends on the starting position of the finisher. Let $Z_{\mathcal{A}_0}$ denote the set of points (x, y) such that Algorithm 1 executes

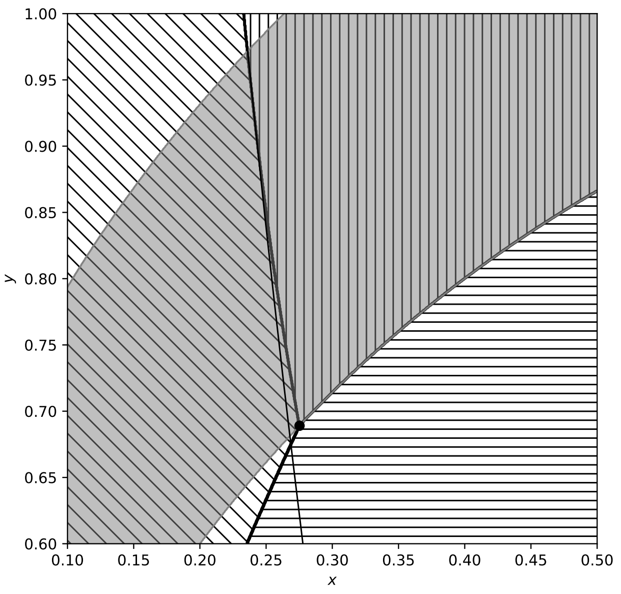


Figure 6: Region in Figure 4 bounded by solid-bordered square box.

$\mathcal{A}_0$ (i.e., $(x-1)^2+y^2>1$ and $\mathrm{CR}_{\mathcal{A}_0(x,y)} \le \mathrm{CR}_{\mathcal{A}_1(x,y)}$, or else $(x-1)^2+y^2 \le 1$ and $\mathrm{CR}_{\mathcal{A}_0(x,y)} \le \mathrm{CR}_{\mathcal{A}_d(x,y)}$). Similarly, let $Z_{\mathcal{A}_1}$ denote the set of points (x,y) such that Algorithm 1 executes $\mathcal{A}_1$ (i.e., $(x-1)^2+y^2>1$ and $\mathrm{CR}_{\mathcal{A}_0(x,y)} > \mathrm{CR}_{\mathcal{A}_1(x,y)}$) and $Z_{\mathcal{A}_1}$ the set of points (x,y) such that Algorithm 1 executes $\mathcal{A}_d$ (i.e., $(x-1)^2+y^2 \le 1$ and $\mathrm{CR}_{\mathcal{A}_0(x,y)} > \mathrm{CR}_{\mathcal{A}_d(x,y)}$). In other words, for a finisher starting position (x,y), the hybrid algorithm executes algorithm $\mathcal{A}_0$ iff $(x,y) \in Z_{\mathcal{A}_0}$, $\mathcal{A}_1$ iff $(x,y) \in Z_{\mathcal{A}_1}$, and $\mathcal{A}_d$ iff $(x,y) \in Z_{\mathcal{A}_d}$. Observe the regions $Z_{\mathcal{A}_0}$, $Z_{\mathcal{A}_1}$, and $Z_{\mathcal{A}_d}$ are disjoint and their union comprises the entire plane. Figure 4 depicts these regions.

Observe if (x,y) is inside the disk $D(1,1)$, either $\mathcal{A}_0(x,y)$ or $\mathcal{A}_d(x,y)$ has the least competitive ratio. Otherwise, if $(x,y) \notin \overline{D}(1,1)$ either $\mathcal{A}_0(x,y)$ or $\mathcal{A}_1(x,y)$ has the least competitive ratio. This is consistent, of course, with Lemma 2. Inside each of these regions, there are even more interesting things going on. Examine the bang-bang curve (i.e., the separating curve) between the regions $Z_{\mathcal{A}_0}$ and $Z_{\mathcal{A}_d}$ inside the disk $D(1/2,1/2)$. Recall that the worst-case fail time for $\mathcal{A}_d(x,y)$ is $t_d = \frac{x(x-1)+y^2}{2(x+\sqrt{x})}$ if $(x,y) \in D(1/2,1/2)$ and 0 otherwise. This is what causes the change of inflection when the bang-bang curve leaves the disk $D(1/2,1/2)$.

Now, look at the bang-bang curve between $Z_{\mathcal{A}_0}$ and $Z_{\mathcal{A}_1}$ outside of the disk $D(1,1)$. Recall the worst-case fail time for $\mathcal{A}_1(x,y)$ is

$$t_1 = \begin{cases} 1-3y/4 & \text{if } x=1 \\ \frac{x^2+y^2+z_1(1-x)-1-z_1\sqrt{b}}{2(x-1)} & \text{otherwise} \end{cases}$$

(where $z_1 = \sqrt{(x-1)^2+y^2}$ abd $b = x(x+z_1-2)+y^2-z_1+1$) if $t_1>0$ and 0 otherwise. Outside of this region, we can find a closed-form equation for the curve separating $\mathcal{A}_0$ and $\mathcal{A}_1$ since the competitive ratios for $\mathrm{CR}_{\mathcal{A}_0}$ and $\mathrm{CR}_{\mathcal{A}_1}$ are relatively simple. We can find where the competitive ratios are equal for this case:

$$\mathrm{CR}_{\mathcal{A}_1(x,y)}(0) = \mathrm{CR}_{\mathcal{A}_0(x,y)}$$
$$\frac{\sqrt{(x-1)^2+y^2}+2}{\sqrt{x^2+y^2}+1} = \frac{\sqrt{x^2+y^2}+1}{\sqrt{(x-1)^2+y^2}}.$$

By cross-multiplying the right-hand side and simplifying, we obtain:

$$-2x+1 = 2x\sqrt{x^2+y^2}+x^2$$

From this, it is easy to arrive at:

$$y = \pm\sqrt{\left(\frac{1-2x-x^2}{2x}\right)^2 - x^2}$$
$$= \pm\frac{\sqrt{1-4x+2x^2+4x^3-3x^4}}{2x} \qquad (4.1)$$

Unfortunately, this is the only bang-bang curve for which we were able to find a closed form equation. Observe in Figure 6 how the bang-bang curve diverges from the curve given by Equation (4.1) when $t_1 > 0$ (inside the shaded region).

5 Conclusion

In this paper, we present a competitively optimal algorithm for two-agent delivery in the plane with a single faulty agent. We show that the competitive ratio of the algorithm depends on the relative positioning of the two agents and the destination. An upper bound of 3 on the competitive ratio over all points (x,y) is straightforward to prove (both $\mathrm{CR}_{\mathcal{A}_0(x,y)}$ and $\mathrm{CR}_{\mathcal{A}_1(x,y)}$ have a maximum of 3). Numerical calculations indicate the maximum competitive ratio is given by the case where $\mathrm{CR}_{\mathcal{A}_0(x,y)} = \mathrm{CR}_{\mathcal{A}_1(x,y)} = \mathrm{CR}_{\mathcal{A}_d(x,y)}$, which is approximately 1.74197 at $x \approx 0.275257, y \approx 0.689019$. The results presented in this paper introduce a number of interesting questions and avenues for future work. First, we assume the starter can only move directly toward the destination. It would be interesting to see how the competitive ratio changes if the starter can move in any direction (e.g., toward the finisher). Second, we could remove the assumption that the starter/finisher move at the same speed. In this case, the finisher might be able to deliver the package faster by participating even if the starter does not fail. We could also extend the problem to consider multiple (potentially faulty) agents (i.e., what happens if the finisher also fails?). Finally, another interesting open problem arises when the starter's trajectory from the source to the target is on a curve other than a line segment.

References

[1] N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM J. Comput.*, 36(1):56–82, 2006.

[2] S. Alpern and S. Gal. *The theory of search games and rendezvous*, volume 55 of *International series in operations research and management science.* Kluwer, 2003.

[3] I. A. Carvalho, T. Erlebach, and K. Papadopoulos. On the fast delivery problem with one or two packages. *J. Comput. Syst. Sci.*, 115:246–263, 2021.

[4] J. Coleman, L. Cheng, and B. Krishnamachari. Search and rescue on the line. In *Structural Information and Communication Complexity*, pages 297–316, Cham, 2023. Springer Nature Switzerland.

[5] J. Coleman, E. Kranakis, D. Krizanc, and O. Morales-Ponce. Message delivery in the plane by robots with different speeds. In *Stabilization, Safety, and Security of Distributed Systems - 23rd International Symposium, SSS 2021, Virtual Event, November 17-20, 2021, Proceedings*, volume 13046 of *Lecture Notes in Computer Science*, pages 305–319. Springer, 2021.

[6] J. Coleman, E. Kranakis, D. Krizanc, and O. Morales-Ponce. The pony express communication problem. In *Combinatorial Algorithms - 32nd International Workshop, IWOCA 2021, Ottawa, ON, Canada, July 5-7, 2021, Proceedings*, volume 12757 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2021.

[7] J. Coleman, D. Krizanc, E. Kranakis, and O. Morales-Ponce. Mathematica code for faulty delivery problem. https://github.com/jaredraycoleman/faulty_delivery, 2025. Accessed: 2025-06-20.

[8] J. R. Coleman, D. Krizanc, E. Kranakis, and O. Morales-Ponce. Optimal delivery with a faulty drone. *CoRR*, abs/2404.17711, 2024.

[9] J. Czyzowicz, L. Gasieniec, A. Kosowski, E. Kranakis, D. Krizanc, and N. Taleb. When patrolmen become corrupted: Monitoring a graph using faulty mobile robots. *Algorithmica*, 79(3):925–940, 2017.

[10] J. Czyzowicz, K. Georgiou, M. Godon, E. Kranakis, D. Krizanc, W. Rytter, and M. Wlodarczyk. Evacuation from a disc in the presence of a faulty robot. In *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017, Porquerolles, France, June 19-22, 2017, Revised Selected Papers*, volume 10641 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 2017.

[11] J. Czyzowicz, E. Kranakis, D. Krizanc, L. Narayanan, and J. Opatrny. Search on a line with faulty robots. *Distributed Comput.*, 32(6):493–504, 2019.

[12] T. Erlebach, K. Luo, and F. C. R. Spieksma. Package delivery using drones with restricted movement areas. In *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 49:1–49:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[13] K. Georgiou, E. Kranakis, N. Leonardos, A. Pagourtzis, and I. Papaioannou. Optimal circle search despite the presence of faulty robots. In *Algosensors 2019, Munich, Germany, September 12-13, 2019*, volume 11931 of *LNCS*, pages 192–205. Springer, 2019.

[14] A. M. Raivi, S. M. A. Huda, M. M. Alam, and S. Moh. Drone routing for drone-based delivery systems: A review of trajectory planning, charging, and security. *Sensors*, 23(3):1463, 2023.

[15] M. Torabbeigi, G. J. Lim, and S. J. Kim. Drone delivery schedule optimization considering the reliability of drones. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1048–1053. IEEE, 2018.

[16] Y. Yang, S. Souissi, X. Défago, and M. Takizawa. Fault-tolerant flocking for a group of autonomous mobile robots. *J. Syst. Softw.*, 84(1):29–36, 2011.

Efficient Reconfiguration of Tile Arrangements by a Single Active Robot

Aaron T. Becker* Sándor P. Fekete† Jonas Friemel‡ Ramin Kosfeld† Peter Kramer† Harm Kube§ Christian Rieck¶ Christian Scheffer‡ Arne Schmidt†

Abstract

We consider the problem of reconfiguring a two-dimensional connected grid arrangement of passive building blocks from a start configuration to a goal configuration, using a single active robot that can move on the tiles, remove individual tiles from a given location and physically move them to a new position by walking on the remaining configuration. The objective is to determine a schedule that minimizes the overall makespan, while keeping the tile configuration connected. We provide both negative and positive results. (1) We generalize the problem by introducing weighted movement costs, which can vary depending on whether tiles are carried or not, and prove that this variant is NP-hard. (2) We give a polynomial-time constant-factor approximation algorithm for the case of disjoint start and target bounding boxes, which additionally yields optimal carry distance for 2-scaled instances.

1 Introduction

Building and modifying structures consisting of many basic components is an important objective, both in fundamental theory and in a spectrum of practical settings. Transforming such structures with the help of autonomous robots is particularly relevant in very large [17] and very small dimensions [41] that are hard to access for direct human manipulation, e.g., in extraterrestrial space [10,34] or in microscopic environments [7]. This gives rise to the natural algorithmic problem of rearranging a given start configuration of many *passive* objects by a small set of *active* agents to a desired target configuration. Performing such reconfiguration at scale faces a number of critical challenges, including (i) the cost and weight of materials, (ii) the potentially disastrous accumulation of errors, (iii) the development of simple yet resilient agents to carry out the active role, and (iv) achieving overall feasibility and efficiency.

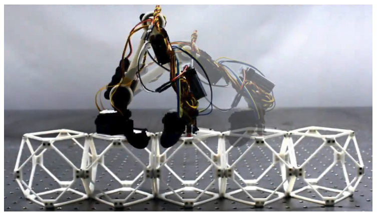


Figure 1: A Bill-E robot moves on a configuration of light-weight material and relocate individual voxels for overall reconfiguration. Photo adapted from [32].

In recent years, significant advances have been made to deal with these difficulties. Macroscopically, ultra-light and scalable composite lattice materials [13,28,29] tackle the first problem, making it possible to construct modular, reconfigurable structures with platforms such as NASA's Bill-E and Armadas [28,31,33]; the underlying self-adjusting lattice also resolves the issues of accuracy and error correction (ii), allowing it to focus on discrete, combinatorial structures, consisting of regular tiles (in two dimensions) or voxels (in three dimensions). A further step has been the development of simple autonomous robots [11,32] that can be used to carry out complex tasks (iii), as shown in Figure 1: The robot can move on the tile arrangement, remove individual tiles and physically relocate them to a new position by walking on the remaining configuration, which needs to remain connected at all times. At microscopic scales, advances in micro- and nanobots [39,40] offer novel ways to (re)configure objects and mechanisms, e.g., assembling specific structures or gathering in designated locations for tasks like targeted drug delivery [9,35].

In this paper, we deal with challenge (iv): How can we use such a robot to transform a given start configuration into a desired goal arrangement, as quickly as possible?

1.1 Our contributions

We investigate the problem of finding minimum cost reconfiguration schedules for a single active robot operating on a (potentially large) number of tiles, and give the following results. Details for statements marked by ($\star$) can be found in the full version of our paper [8].

*Department of Electrical Engineering, University of Houston, USA, atbecker@uh.edu

†Department of Computer Science, TU Braunschweig, Germany, {kosfeld, kramer, aschmidt}@ibr.cs.tu-bs.de, s.fekete@tu-bs.de

‡Department of Electrical Engineering and Computer Science, Bochum University of Applied Sciences, Germany, {jonas.friemel, christian.scheffer}@hs-bochum.de

§Department of Computer Science, TU Berlin, Germany, h.kube@campus.tu-berlin.de

¶Department of Discrete Mathematics, University of Kassel, Germany, christian.rieck@mathematik.uni-kassel.de

(1) We present a generalized version of the problem, parameterized by movement costs, which can vary depending on whether tiles are carried or not, and show that this is NP-hard.

(2) We give a polynomial-time constant-factor approximation algorithm for the case of disjoint start and target bounding boxes. Our approach further yields optimal carry distance for 2-scaled instances.

1.2 Related work

Recently, the authors of [24,25] showed that computing optimal schedules for BILL-E bots, see [31] and Figure 1, is NP-hard in unweighted models. They designed a heuristic approach that exploits rapidly exploring random trees (RRT) and a time-dependent variant of the A* algorithm, as well as target swapping heuristics to reduce the overall distance traveled for multiple robots. The authors of [12] present an algorithm for computing feasible build orders that adhere to three-dimensional tile placement constraints in the ARMADAS project.

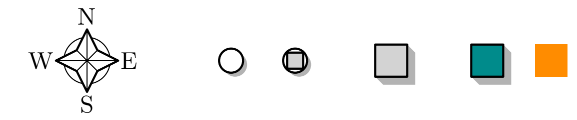
A different context for reconfiguration arises from programmable matter [15]. Here, finite automata are capable of building bounding boxes of tiles around polyominoes, as well as scale and rotate them while maintaining connectivity at all times [20,38]. On hexagonal grids, finite automata can build and recognize simple shapes such as triangles or parallelograms [26, 27, 30] as well as more complicated shapes if they are able to recognize nodes that belong to the target shape [23].

When considering active matter, arrangements composed of self-moving objects (or agents), numerous models exist [5,6,14,18,37]. For example, in the *sliding cube model* (or the *sliding square model* in two dimensions) first introduced in 2003 [21, 22], agents are allowed to slide along other, temporarily static, agents for reconfiguration, but must maintain connectivity throughout. The authors of [3] show that sequential reconfiguration in two dimensions is always possible, but deciding the minimal makespan is NP-complete. Recent work presents similar results for the three-dimensional setting [1,36], as well as parallel movement [4].

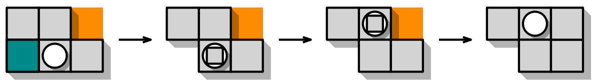
In a relaxed model, the authors of [18, 19] show that parallel connected reconfiguration of swarms of (labeled) agents is NP-complete, even for makespan 2, and present algorithms for schedules with constant *stretch*; the ratio of a schedule's makespan to the individual maximum minimum distance between start and target.

1.3 Preliminaries

For the following, we refer to Figure 2. We are given a fixed set of n indistinguishable square *tile* modules located at discrete, unique positions (x, y) in the infinite integer grid $\mathbb{Z}^2$. If their positions induce a connected subgraph of the grid, where two positions are connected if either their x- or y-coordinate differs by 1, we say that the tiles form a connected *configuration* or *polyomino*. Let $\mathcal{C}(n)$ refer to the set of all polyominoes of n tiles.



(a) Left to right: The cardinal directions, a robot without and with payload, a tile, and tiles that only exist in either the start or the target configuration, respectively.



(b) A schedule for the instance on the left. The robot moves on tiles, picking up and dropping off adjacent tiles.

Figure 2: A brief overview of legal operations.

Consider a robot that occupies a single tile at any given time. In discrete time steps, the robot can either move to an adjacent tile, pick up a tile from an adjacent position (if it is not already carrying one), or place a carried tile at an adjacent unoccupied position. Tiles can be picked up only if connectivity is preserved.

We use cardinal directions; the unit vectors $(1, 0)$ and $(0, 1)$ correspond to *east* and *north*, respectively. Naturally, their opposing vectors extend *west* and *south*.

A *schedule* S is a finite, connectivity-preserving sequence of operations to be performed by the robot. As the robot's motion is restricted to movements on the polyomino, we refer to distances it travels as *geodesic* distances. Let $d_C(S)$ denote the *carry distance*, which is the number of robot moves while carrying a tile (i.e., the sum of geodesic distances between consecutive pickups and drop-offs), plus the number of all pickups and drop-offs in S. Accounting for the remaining moves without carrying a tile, the *empty distance* $d_E(S)$ is the sum of geodesic distances between drop-offs and pickups in S. For $C_s, C_t \in \mathcal{C}(n)$, we say that S is a schedule for $C_s \rightrightarrows C_t$ exactly if it transforms C_s into C_t.

Problem statement. We consider SINGLE ROBOT RECONFIGURATION: Given configurations $C_s, C_t \in \mathcal{C}(n)$ and a rational weight factor $\lambda \in [0, 1]$, determine a schedule S for $C_s \rightrightarrows C_t$ that minimizes the *weighted makespan* $|S| := \lambda \cdot d_E(S) + d_C(S)$. We refer to the minimum weighted makespan for a given instance as OPT.

2 Computational complexity of the problem

We investigate the computational complexity of the decision variant of the reconfiguration problem. In particular, we prove that the problem is NP-hard for any rational $\lambda \in [0,1]$. This generalizes the previous result from [25], which handles the case of $\lambda = 1$.

Theorem 1 Single Robot Reconfiguration *is* NP*-hard for any rational* λ*.*

We distinguish between the two cases of (1) $\lambda \in (0,1]$ and (2) $\lambda = 0$. For (1), we modify the construction from [25] to yield a slightly stronger statement, reducing from the Hamiltonian path problem in grid graphs. The primary result of our reduction is the following.

Lemma 2 (⋆) Single Robot Reconfiguration *is* NP*-hard for* $\lambda \in (0,1]$*.*

Our modified construction further implies that it is NP-hard to decide the minimal number of pickup/drop-off operations in an optimal schedule with $\lambda \in (0,1]$.

Corollary 3 *Given two configurations* $C_s, C_t \in \mathcal{C}(n)$ *and an integer* $k \in \mathbb{N}$*, it is* NP*-hard to decide whether there exists an optimal schedule* $C_s \rightrightarrows C_t$ *with at most* k *pickup (at most* k *drop-off) operations, if* $\lambda \in (0,1]$*.*

The reduction for (2) is significantly more involved: As $\lambda = 0$, the robot is effectively allowed to "teleport" across the configuration, albeit only without cargo.

Lemma 4 (⋆) Single Robot Reconfiguration *is* NP*-hard for* $\lambda = 0$*.*

We reduce from Planar Monotone 3Sat [16], following ideas by the authors of [3] for the sequential sliding square problem. This variant of the SAT problem asks whether a given Boolean formula φ in conjunctive normal form is satisfiable, given the following properties: First, each clause consists of at most 3 literals, all either positive or negative. Second, the clause-variable incidence graph G_φ admits a plane drawing in which variables are placed on the x-axis, and all positive (resp., negative) clauses are located in the same half-plane, such that edges do not cross the x-axis, see Figure 3. We assume, without loss of generality, that each clause

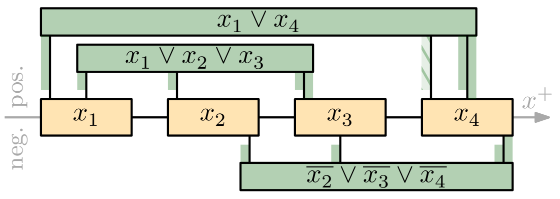


Figure 3: The embedded clause-variable incidence graph of $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$.

contains exactly three literals; otherwise, we extend a shorter clause with a redundant copy of one of the existing literals, e.g., $(x_1 \vee x_4)$ becomes $(x_1 \vee x_4 \vee x_4)$.

Our reduction maps from an instance φ of Planar Monotone 3Sat to an instance $\mathcal{I}_\varphi$ of Single Robot Reconfiguration such that the minimal feasible makespan $\mathcal{I}_\varphi$ is determined by whether φ is satisfiable. Recall that, due to $\lambda = 0$, we only account for carry distance. Consider an embedding of the clause-variable incidence graph as above, where C and V refer to the m clauses and n variables of φ, respectively.

We construct $\mathcal{I}_\varphi$ as follows. A *variable gadget* is placed on the x-axis for each $x_i \in V$, and connected along the axis in a straight line. Intuitively, the variable gadget asks the robot to move a specific tile west along one of two feasible paths, which encode a value assignment. These paths are highlighted in red and blue in Figure 4, each of length exactly $9(\delta(x_i)+1)$, where $\delta(x_i)$ refers to the degree of x_i in the clause-variable incidence graph. Both paths require the robot to place

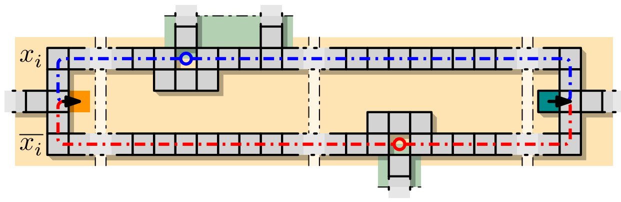


Figure 4: The variable gadget; the middle segments are repeated to match the incident clauses.

its payload into the highlighted gaps to minimize the makespan, stepping over it before picking up again. We further add one *clause gadget* per clause, connected to its incident variables. These are effectively combs with three or four prongs, connected to a variable gadget by the rightmost one, see Figure 5. The remaining prongs

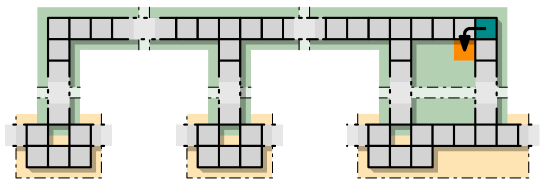


Figure 5: A clause gadget and its variable attachment.

each terminate at a position diagonally adjacent to the variable gadget of a corresponding literal. To solve the gadget, the rightmost prong must be temporarily disconnected from the comb, meaning that another prong must have been connected to a variable gadget first. Such a link can be established without extra cost if one of the adjacent variable gadgets has a matching Boolean value, allowing us to use its payload to temporarily close one of the gaps marked by circles in Figure 4.

We can show that a weighted makespan of $29m + 9n$ can be achieved exactly if φ is satisfiable; recall that m and n denote the number of clauses and variables in φ, respectively. Theorem 1 is then simply the union of Lemmas 2 and 4.

3 Constant-factor approximation for 2-scaled instances

We now turn to the case in which the configurations have *disjoint bounding boxes*, i.e., there exists an axis-parallel bisector that separates the configurations. Without loss of generality, let this bisector be horizontal such that the target configuration lies south. We present a constant-factor approximation algorithm. For the remainder of this section, we additionally impose the constraint that both the start and target configurations are 2-*scaled*, i.e., they consist of 2×2-squares of tiles aligned with a corresponding grid. In Section 4, we extend our result to non-scaled configurations.

Theorem 5 *For any $\lambda \in [0,1]$, there exists a constant c such that for any pair of 2-scaled configurations $C_s, C_t \in \mathcal{C}(n)$ with disjoint bounding boxes, we can efficiently compute a schedule for $C_s \rightrightarrows C_t$ with weighted makespan at most $c \cdot \mathsf{OPT}$.*

Our algorithm utilizes *histograms* as intermediate configurations. A histogram consists of a *base* strip of unit height (a single tile, also when 2-scaled) and unit width *columns* attached orthogonally to its base. The direction of its columns determines the orientation of a histogram, e.g., H_s in Figure 6 is *north-facing*.

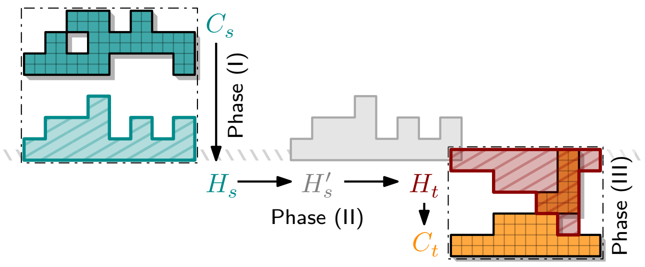


Figure 6: An example for a start and target configuration C_s, C_t, the intermediate histograms H_s, H_t with a shared baseline, and the horizontally translated H'_s that shares a tile with H_t. If H_s and H_t overlap, $H'_s := H_s$.

As illustrated in Figure 6, we proceed in three phases.

Phase (I). Iteratively move components of C_s south until it forms a north-facing histogram H_s such that its base shares its y-coordinate with a module of C_t.

Phase (II). Translate H_s to overlap with the target bounding box and transform it into a specific south-facing histogram H_t within the bounding box of C_t.

Phase (III). Finally, apply Phase (I) in reverse to obtain C_t from H_t.

We reduce this to two subroutines: Transforming any 2-scaled configuration into a histogram and reconfiguring any two histograms into one another. Note that while the respectively obtained histograms are 2-scaled when the initial configurations are, the reconfiguration between two opposite-facing histograms actually does not depend on this condition. In fact, the same reconfiguration routines remain valid even when the histograms involved are not 2-scaled, highlighting the broader applicability of our method.

3.1 Preliminaries for the algorithm

For our algorithm, we use the following terms. Given two configurations $C_s, C_t \in \mathcal{C}(n)$, the weighted bipartite graph $G_{C_s,C_t} = (C_s \cup C_t, C_s \times C_t, L_1)$ assigns each edge a weight equal to the L_1-distance between its end points.

A *perfect matching* M in G_{C_s,C_t} is a subset of edges such that every vertex is incident to exactly one of them; its weight $w(M)$ is defined as the sum of its edge weights. By definition, there exists at least one such matching in G_{C_s,C_t}. Furthermore, a *minimum weight perfect matching* (MWPM) is a perfect matching M of minimum weight $\sigma(C_s, C_t) = w(M)$, which is a natural lower bound on the necessary carry distance, i.e., OPT.

Let S be any schedule for $C_s \rightrightarrows C_t$. Then S induces a perfect matching in G_{C_s,C_t}, as it moves every tile of C_s to a distinct position of C_t. We say that S has *optimal carry distance* exactly if $d_C(S) = \sigma(C_s, C_t)$.

3.2 Phase (I): Transforming into a histogram

We proceed by constructing a histogram from an arbitrary 2-scaled configuration by moving tiles strictly in one cardinal direction.

Lemma 6 ($\star$) *Let $C_s \in \mathcal{C}(n)$ be a 2-scaled polyomino and let H_s be a histogram that can be created from C_s by moving tiles in only one cardinal direction. We can efficiently compute a schedule with optimal carry distance and total makespan $\mathcal{O}(n + \sigma(C_s, H_s))$ for $C_s \rightrightarrows H_s$.*

To achieve this, we iteratively move sets of tiles in the target direction by two units, until the histogram is constructed. We give a high-level explanation of our approach by example of a north-facing histogram, as depicted in Figure 7.

Let P be any intermediate 2-scaled polyomino obtained by moving tiles south while realizing $C_s \rightrightarrows H_s$. Let H be the set of maximal vertical strips of tiles that contain a base tile in H_s, i.e., all tiles that do not need to be moved further south. We define the *free components* of P as the set of connected components in $P \setminus H$. By definition, these exist exactly if P is not equivalent to H_s, and once a tile in P becomes part of H, it is not moved again until H_s is obtained.

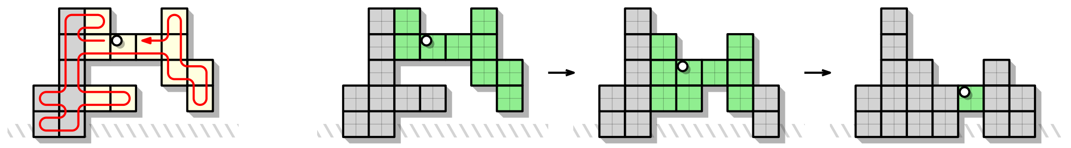


Figure 7: The figure visualizes Lemma 6. Left: A walk across all tiles (red), the set H (gray) and two free components (yellow). Right: Based on the walk, free components are iteratively moved south to reach a histogram shape. The free component that is going to be translated next is highlighted in green.

We compute a *walk* that covers the polyomino, i.e., a path that is allowed to traverse edges multiple times and visits each vertex at least once, by depth-first traversal on an arbitrary spanning tree of P. The robot then continuously follows this walk, iteratively moving free components south and updating its path accordingly: Whenever it enters a free component F of P, it performs a subroutine with makespan $\mathcal{O}(|F|)$ that moves F south by two units. Adjusting the walk afterward may increase its length by $\mathcal{O}(|F|)$ units per free component.

Upon completion of this algorithm, we can bound both the total time spent performing the subroutine and the additional cost incurred by extending the walk by $\mathcal{O}(\sigma(C_s, H_s))$. Taking into account the initial length of the walk, the resulting makespan is $\mathcal{O}(n + \sigma(C_s, H_s))$. The subroutine for the translation of free components can be stated as follows.

Lemma 7 *We can efficiently compute a schedule of makespan $\mathcal{O}(|F|)$ to translate any free component F of a 2-scaled polyomino in the target direction by two units.*

Proof. Without loss of generality, let the target direction be south. We show how to translate F south by one unit without losing connectivity, which we do twice.

We follow a bounded-length walk across F that visits exclusively tiles with a tile neighbor in northern direction. Such a walk can be computed by depth-first traversal of F. Whenever the robot enters a maximal vertical strip of F for the first time, it picks up the northernmost tile, places it at the first unoccupied position to its south, and continues its traversal.

As each strip is handled exactly once, the total movement cost on vertical strips for carrying tiles and returning to the pre-pickup position is bounded by $\mathcal{O}(|F|)$. This bound also holds for the length of the walk. □

Applying Lemma 7 to the entire polyomino, rather than restricting it to a free component, allows the translation of the polyomino in any direction with asymptotically optimal makespan.

Corollary 8 *Any 2-scaled polyomino can be translated by k units in any cardinal direction by a schedule of weighted makespan $\mathcal{O}(n \cdot k)$.*

3.3 Phase (II): Reconfiguring histograms

It remains to show how to transform one histogram into the other. By the assumption of the existence of a horizontal bisector between the bounding boxes of C_s and C_t, the histogram H_s is north-facing, whereas H_t is south-facing. The bounding box of C_s is vertically extended to share exactly one y-coordinate with the bounding box of C_t, and this is where both histogram bases are placed; see Figure 6 for an illustration. Note that the histograms may not yet overlap. However, by Corollary 8, the tiles in H_s can be moved toward H_t with asymptotically optimal makespan until both histogram bases share a tile.

Lemma 9 (⋆) *Let H_s and H_t be opposite-facing histograms that share a base tile. We can efficiently compute a schedule of makespan $\mathcal{O}(n + \sigma(H_s, H_t))$ for $H_s \rightrightarrows H_t$ with optimal carry distance.*

As illustrated on the left side of Figure 8, we order all tiles to be moved, as well as all positions to be populated, from west to east and north to south. By that, iteratively moving the first remaining tile from $H_s \setminus H_t$ to $H_t \setminus H_s$ yields a schedule with the desired properties. The right side of Figure 8 depicts a special case in which the westernmost positions in $H_t \setminus H_s$ cannot be reached initially. This case requires a preprocessing in which the baseline is first extended to the west.

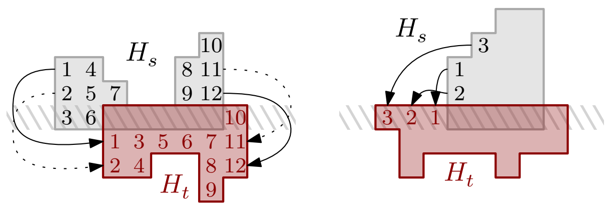


Figure 8: Left: Ordering of tile moves for $H_s \rightrightarrows H_t$. Right: If the westernmost unoccupied position in H_t is unreachable, the base may need to be extended first.

3.4 Correctness of the algorithm

In the previous sections, we presented schedules for each phase of the overall algorithm. We will now leverage these insights to prove the main result of this section, restated here.

Theorem 5 *For any $\lambda \in [0,1]$, there exists a constant c such that for any pair of 2-scaled configurations $C_s, C_t \in \mathcal{C}(n)$ with disjoint bounding boxes, we can efficiently compute a schedule for $C_s \rightrightarrows C_t$ with weighted makespan at most $c \cdot \mathit{OPT}$.*

Proof. By Lemmas 6 and 9, the makespan of the three phases is bounded by, respectively,

$$\mathcal{O}(n+\sigma(C_s,H_s)),\ \mathcal{O}(n+\sigma(H_s,H_t)),\ \mathcal{O}(n+\sigma(H_t,C_t)),$$

which we now bound by $\mathcal{O}(\sigma(C_s,C_t))$, proving asymptotic optimality for $C_s \rightrightarrows C_t$.

Clearly, $n \in \mathcal{O}(\sigma(C_s,C_t))$, as each of the n tiles has to be moved due to $C_s \cap C_t = \varnothing$. We prove a tight bound on the remaining terms, i.e.,

$$\sigma(C_s,H_s)+\sigma(H_s,H_t)+\sigma(H_t,C_t)=\sigma(C_s,C_t). \qquad (\blacklozenge)$$

In Phase (I), tiles are moved exclusively toward the bounding box of C_t along shortest paths to obtain H_s; therefore, $\sigma(C_s,C_t)=\sigma(C_s,H_s)+\sigma(H_s,C_t)$. The same applies to the reverse of Phase (III), i.e., $\sigma(C_t,H_s)=\sigma(C_t,H_t)+\sigma(H_t,H_s)$. As the lower bound is symmetric, Equation ($\blacklozenge$) immediately follows. The total makespan is thus in $\mathcal{O}(\sigma(C_s,C_t))=\mathcal{O}(\mathsf{OPT})$. □

Lemmas 6 and 9 move tiles on shortest paths and establish schedules that minimize the carry distance. Equation ($\blacklozenge$) ensures that the combined paths remain shortest possible with regard to $C_s \rightrightarrows C_t$. Thus, the provided schedule has optimal carry distance. In particular, we obtain an optimal schedule when $\lambda = 0$, which corresponds to the case where the robot incurs no cost for movement when not carrying a tile.

Corollary 10 *An optimal schedule for $C_s \rightrightarrows C_t$ can be computed efficiently for any two 2-scaled configurations $C_s, C_t \in \mathcal{C}(n)$ with disjoint bounding boxes and $\lambda = 0$.*

4 Constant-factor approximation for general instances

The key advantage of 2-scaled instances is the absence of cut vertices, which simplifies the maintenance of connectivity during reconfiguration. Therefore, the challenge with general instances lies in managing cut vertices.

Most parts of our previous method already work independent of the configuration scale. The only modification required concerns Lemma 7, as the polyomino may become disconnected while moving free components that are not 2-scaled. To preserve local connectivity during the reconfiguration, we utilize two auxiliary tiles as a patching mechanism; this technique is also employed in other models [2,37].

Lemma 11 ($\star$) *Let the robot hold two auxiliary tiles. Given a free component F on a polyomino P, we can efficiently compute a schedule of makespan $\mathcal{O}(|F|)$ to translate F in the target direction by one unit.*

A key idea in the proof of Lemma 11 is partitioning the configuration into horizontal and vertical strips.

By using auxiliary tiles, we are able to preserve connectivity while translating horizontal strips; see Figure 9. To guarantee connectivity, the strips must be moved in a specific order; we resolve this constraint using a recursive strategy that systematically moves dependent strips in the correct sequence.

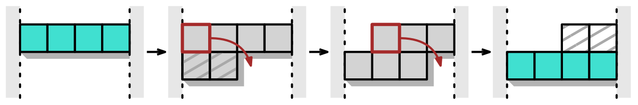


Figure 9: Using two auxiliary tiles to retain connectivity while moving a unit-height strip south.

Theorem 12 ($\star$) *For any $\lambda \in [0,1]$, there exists a constant c such that for any two configurations $C_s, C_t \in \mathcal{C}(n)$ with disjoint bounding boxes, we can efficiently compute a schedule for $C_s \rightrightarrows C_t$ with weighted makespan at most $c \cdot \mathit{OPT}$.*

To prove Theorem 12, it suffices to demonstrate how the auxiliary tiles can be obtained (any two tiles that do not break connectivity can serve this purpose) and that carrying two tiles can be emulated by sequentially carrying one tile after the other.

5 Discussion

We presented progress on the reconfiguration problem for tile-based structures (i.e., polyominoes) within abstract material-robot systems. In particular, we showed that the problem is NP-hard for any weighting between moving with or without carrying a tile.

Complementary to this negative result, we developed an algorithm to reconfigure two polyominoes into one another in the case that both configurations are contained in disjoint bounding boxes. The computed schedules are within a constant factor of the optimal reconfiguration schedule. It is easy to see that our approach can also be used to construct a polyomino, rather than reconfigure one into another. Instead of deconstructing a start configuration to generate building material, we can assume that tiles are located within a "depot" from which they can be picked up. Performing the second half of the algorithm works as before and builds the target configuration out of tiles from the depot.

Several open questions remain. A natural open problem is to adapt the approach to instances in which the bounding boxes of the configurations intersect, i.e., they overlap, or are nested. This hinges on proving a good lower bound on the makespan of any such schedule. Note that a minimum-weight perfect matching does not provide a reliable lower bound in this setting, as the cost of an optimal solution can be arbitrarily larger; particularly in cases involving small matchings, as visualized in Figure 10.

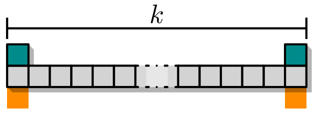


Figure 10: Although the MWPM has weight 4, the robot is required to perform at least $k - 1 \gg 4$ empty moves. This discrepancy shows that the MWPM significantly underestimates the true cost, making it unsuitable as a lower bound for an approximation in the setting where bounding boxes intersect.

However, an alternative would be to provide an algorithm that achieves worst-case optimality. Furthermore, the more general question about three dimensional settings remain open. Our methods could likely be generalized for parallel execution by multiple robots. More intricate is the question on whether a fully distributed approach is possible.

Acknowledgements

This work was partially supported by multiple sources: German Research Foundation (DFG) project “Space Ants” (FE 407/22-1 and SCHE 1931/4-1) and grant 522790373, as well as by grants NSF IIS-2130793, ARL W911NF-23-2-0014, and DAF AFX23D-TCSO1.

References

[1] Zachary Abel, Hugo A. Akitaya, Scott Duke Kominers, Matias Korman, and Frederick Stock. A universal in-place reconfiguration algorithm for sliding cube-shaped robots in a quadratic number of moves. In *Symposium on Computational Geometry (SoCG)*, pages 1:1–1:14, 2024. doi:10.4230/LIPICS.SOCG.2024.1.

[2] Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmovic, Robin Y. Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $O(1)$ musketeers. *Algorithmica*, 83:1316–1351, 2021. doi:10.1007/s00453-020-00784-6.

[3] Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. In *Scandinavian Symposium on Algorithm Theory (SWAT)*, pages 4:1–4:19, 2022. doi:10.4230/LIPICS.SWAT.2022.4.

[4] Hugo A. Akitaya, Sándor P. Fekete, Peter Kramer, Saba Molaei, Christian Rieck, Frederick Stock, and Tobias Wallner. Sliding squares in parallel. In *European Symposium on Algorithms (ESA)*, 2025. doi:10.48550/arXiv.2412.05523.

[5] Abdullah Almethen, Othon Michail, and Igor Potapov. Pushing lines helps: Efficient universal centralised transformations for programmable matter. *Theoretical Computer Science*, 830:43–59, 2020. doi:10.1016/j.tcs.2020.04.026.

[6] Abdullah Almethen, Othon Michail, and Igor Potapov. On efficient connectivity-preserving transformations in a grid. *Theoretical Computer Science*, 898:132–148, 2022. doi:10.1016/j.tcs.2022.09.016.

[7] Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, Christian Rieck, Christian Scheffer, and Arne Schmidt. Tilt assembly: Algorithms for micro-factories that build objects with uniform external forces. *Algorithmica*, 82(2):165–187, 2020. doi:10.1007/S00453-018-0483-9.

[8] Aaron T. Becker, Sándor P. Fekete, Jonas Friemel, Ramin Kosfeld, Peter Kramer, Harm Kube, Christian Rieck, Christian Scheffer, and Arne Schmidt. Efficient reconfiguration of tile arrangements by a single active robot, 2025. arXiv:2502.09299.

[9] Aaron T. Becker, Sándor P. Fekete, Li Huang, Phillip Keldenich, Linda Kleist, Dominik Krupke, Christian Rieck, and Arne Schmidt. Targeted drug delivery: Algorithmic methods for collecting a swarm of particles with uniform, external forces. In *International Conference on Robotics and Automation (ICRA)*, pages 2508–2514, 2020. doi:10.1109/ICRA40945.2020.9196551.

[10] Mohamed Khalil Ben-Larbi, Kattia Flores Pozo, Tom Haylok, Mirue Choi, Benjamin Grzesik, Andreas Haas, Dominik Krupke, Harald Konstanski, Volker Schaus, Sándor P. Fekete, Christian Schurig, and Enrico Stoll. Towards the automated operations of large distributed satellite systems. Part 1: Review and paradigm shifts. *Advances in Space Research*, 67(11):3598–3619, 2021. doi:10.1016/j.asr.2020.08.009.

[11] Christopher G. Cameron, Zach Fredin, and Neil Gershenfeld. Discrete assembly of unmanned aerial systems. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 339–344, 2022. doi:10.1109/ICUAS54217.2022.9836082.

[12] Kenneth Cheung, Irina Kostitsyna, and Tom Peters. Assembly order planning for modular structures by autonomous multi-robot systems. In *International Conference on Robotics and Automation (ICRA)*, 2025.

[13] Kenneth C. Cheung and Neil Gershenfeld. Reversibly assembled cellular composite materials. *Science*, 341(6151):1219–1221, 2013. doi:10.1126/science.1240889.

[14] Matthew Connor and Othon Michail. Transformation of modular robots by rotation: $3+1$ musketeers for all orthogonally convex shapes. *Journal of Computer and System Sciences*, 150:103618, 2025. doi:10.1016/j.jcss.2024.103618.

[15] Joshua J. Daymude, Kristian Hinnenthal, Andréa W. Richa, and Christian Scheideler. Computing by programmable particles. *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 615–681, 2019. doi:10.1007/978-3-030-11072-7_22.

[16] Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *International Journal on Computational Geometry and Applications*, 22(3):187–206, 2012. doi:10.1142/S0218195912500045.

[17] Sándor P. Fekete. Coordinating swarms of objects at extreme dimensions. In *International Workshop on Combinatorial Algorithms (IWOCA)*, pages 3–13, 2020. doi:10.1007/978-3-030-48966-3_1.

[18] Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected coordinated motion planning with bounded stretch. *Autonomous Agents and Multi-Agent Systems*, 37(2), 2023. doi:10.1007/S10458-023-09626-5.

[19] Sándor P. Fekete, Peter Kramer, Christian Rieck, Christian Scheffer, and Arne Schmidt. Efficiently reconfiguring a connected swarm of labeled robots. *Autonomous Agents and Multi-Agent Systems*, 38(2), 2024. doi:10.1007/s10458-024-09668-3.

[20] Sándor P. Fekete, Eike Niehs, Christian Scheffer, and Arne Schmidt. Connected reconfiguration of lattice-based cellular structures by finite-memory robots. *Algorithmica*, 84(10):2954–2986, 2022. doi:10.1007/s00453-022-00995-z.

[21] Robert Fitch, Zack J. Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2460–2467, 2003. doi:10.1109/IROS.2003.1249239.

[22] Robert Fitch, Zack J. Butler, and Daniela Rus. Reconfiguration planning among obstacles for heterogeneous self-reconfiguring robots. In *International Conference on Robotics and Automation (ICRA)*, pages 117–124, 2005. doi:10.1109/ROBOT.2005.1570106.

[23] Jonas Friemel, David Liedtke, and Christian Scheffer. Efficient shape reconfiguration by hybrid programmable matter. In *European Workshop on Computational Geometry (EuroCG)*, pages 14:1–14:8, 2025. arXiv:2501.08663.

[24] Javier Garcia, Michael Yannuzzi, Peter Kramer, Christian Rieck, and Aaron T. Becker. Connected reconfiguration of polyominoes amid obstacles using RRT*. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 6554–6560, 2022. doi:10.1109/IROS47612.2022.9981184.

[25] Javier Garcia, Michael Yannuzzi, Peter Kramer, Christian Rieck, Sándor P. Fekete, and Aaron T. Becker. Reconfiguration of a 2D structure using spatio-temporal planning and load transferring. In *International Conference on Robotics and Automation (ICRA)*, pages 8735–8741, 2024. doi:10.1109/ICRA57147.2024.10611057.

[26] Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Fabian Kuhn, Dorian Rudolph, and Christian Scheideler. Shape recognition by a finite automaton robot. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 52:1–52:15, 2018. doi:10.4230/LIPIcs.MFCS.2018.52.

[27] Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Fabian Kuhn, Dorian Rudolph, Christian Scheideler, and Thim Strothmann. Forming tile shapes with simple robots. *Natural Computing*, 19(2):375–390, 2020. doi:10.1007/s11047-019-09774-2.

[28] Christine E. Gregg, Damiana Catanoso, Olivia Irene B. Formoso, Irina Kostitsyna, Megan E. Ochalek, Taiwo J. Olatunde, In Won Park, Frank M. Sebastianelli, Elizabeth M. Taylor, Greenfield T. Trinh, and Kenneth C. Cheung. Ultralight, strong, and self-reprogrammable mechanical metamaterials. *Science Robotics*, 9(86), 2024. doi:10.1126/scirobotics.adi2746.

[29] Christine E. Gregg, Joseph H. Kim, and Kenneth C. Cheung. Ultra-light and scalable composite lattice materials. *Advanced Engineering Materials*, 20(9):1800213, 2018. doi:10.1002/adem.201800213.

[30] Kristian Hinnenthal, David Liedtke, and Christian Scheideler. Efficient shape formation by 3D hybrid programmable matter: An algorithm for low diameter intermediate structures. In *Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, pages 15:1–15:20, 2024. doi:10.4230/LIPICS.SAND.2024.15.

[31] Ben Jenett and Kenneth Cheung. BILL-E: Robotic platform for locomotion and manipulation of lightweight space structures. In *Adaptive Structures Conference (ASC)*, 2017. doi:10.2514/6.2017-1876.

[32] Benjamin Jenett, Amira Abdel-Rahman, Kenneth Cheung, and Neil Gershenfeld. Material–robot system for assembly of discrete cellular structures. *IEEE Robotics and Automation Letters*, 4(4):4019–4026, 2019. doi:10.1109/LRA.2019.2930486.

[33] Benjamin Jenett, Daniel Cellucci, Christine Gregg, and Kenneth Cheung. Meso-scale digital materials: modular, reconfigurable, lattice-based structures. In *International Manufacturing Science and Engineering Conference (MSEC)*, 2016. doi:10.1115/MSEC2016-8767.

[34] Benjamin Jenett, Christine Gregg, Daniel Cellucci, and Kenneth Cheung. Design of multifunctional hierarchical space structures. In *Aerospace Conference*, pages 1–10, 2017. doi:10.1109/AERO.2017.7943913.

[35] Matthias Konitzny, Yitong Lu, Julien Leclerc, Sándor P. Fekete, and Aaron T. Becker. Gathering physical particles with a global magnetic field using reinforcement learning. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 10126–10132, 2022. doi:10.1109/IROS47612.2022.9982256.

[36] Irina Kostitsyna, Tim Ophelders, Irene Parada, Tom Peters, Willem Sonke, and Bettina Speckmann. Optimal in-place compaction of sliding cubes. In *Scandinavian Symposium on Algorithm Theory (SWAT)*, pages 31:1–31:14, 2024. doi:10.4230/LIPICS.SWAT.2024.31.

[37] Othon Michail, George Skretas, and Paul G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 102:18–39, 2019. doi:10.1016/j.jcss.2018.12.001.

[38] Eike Niehs, Arne Schmidt, Christian Scheffer, Daniel E. Biediger, Michael Yannuzzi, Benjamin Jenett, Amira Abdel-Rahman, Kenneth C. Cheung, Aaron T. Becker, and Sándor P. Fekete. Recognition and reconfiguration of lattice-based cellular structures by simple robots. In *International Conference on Robotics and Automation (ICRA)*, pages 8252–8259, 2020. doi:10.1109/ICRA40945.2020.9196700.

[39] Ana L. Santos, Dongdong Liu, Anna K. Reed, Aaron M. Wyderka, Alexis van Venrooy, John T. Li, Victor D. Li, Mikita Misiura, Olga Samoylova, Jacob L. Beckham, Ciceron Ayala-Orozco, Anatoly B. Kolomeisky, Lawrence B. Alemany, Antonio Oliver, George P. Tegos, and James M. Tour. Light-activated molecular machines are fast-acting broad-spectrum antibacterials that target the membrane. *Science Advances*, 8(22):eabm2055, 2022. doi:10.1126/sciadv.abm2055.

[40] Zhuoran Zhang, Xian Wang, Jun Liu, Changsheng Dai, and Yu Sun. Robotic micromanipulation: Fundamentals and applications. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:181–203, 2019. doi:10.1146/ANNUREV-CONTROL-053018-023755.

[41] Yangsheng Zhu, Mingjing Qi, Zhiwei Liu, Jianmei Huang, Dawei Huang, Xiaojun Yan, and Liwei Lin. A 5-mm untethered crawling robot via self-excited electrostatic vibration. *IEEE Transactions on Robotics*, 38(2):719–730, 2022. doi:10.1109/TRO.2021.3088053.

Improved Wake-Up Time For Euclidean Freeze-Tag Problem

Sharareh Alipour* Arash Ahadi† Kajal Baghestani‡

Abstract

The Freeze-Tag Problem (FTP) involves activating a set of initially asleep robots as quickly as possible, starting from a single awake robot. Once activated, a robot can assist in waking up other robots. Each active robot moves at unit speed. The objective is to minimize the makespan, i.e., the time required to activate the last robot. A key performance measure is the wake-up ratio, defined as the maximum time needed to activate any number of robots in any primary positions. This work focuses on the geometric (Euclidean) version of FTP, where the initial distance between each asleep robot and the single active robot is at most 1. For $(\mathbb{R}^2, \ell_2)$, we improve the previous upper bound of 4.62 ([7], CCCG 2024) to 4.31. Note that it is known that 3.82 is a lower bound for the wake-up ratio. In $\mathbb{R}^3$, we propose a new strategy that achieves a wake-up ratio of 12 for $(\mathbb{R}^3, \ell_1)$ and 12.76 for $(\mathbb{R}^3, \ell_2)$, improving upon the previous bounds of 13 and $13\sqrt{3}$, respectively, reported in [2].

1 Introduction

The Freeze-Tag Problem (FTP), introduced by Arkin et al. [4], involves activating a set of n inactive robots as quickly as possible. The process starts with a single active robot, while others remain stationary until they are woken up. Once activated, a robot can assist in waking up other robots. The objective is to minimize the makespan, which is the time required to activate the last robot.

FTP has practical applications in robotics, particularly in swarm control tasks such as environment exploration [8, 9, 15, 16], robot formation [13, 14], and searching [15]. It is also relevant in network design, including broadcast and IP multicast problems [3, 5, 11].

This paper examines the Euclidean (geometric) version of FTP. In this version the input consists of the positions of n asleep robots and one active robot in $\mathbb{R}^d$, along with a distance norm ℓ_p for some dimension d and norm parameter p. Each robot's position is represented as a point and each active robot moves at unit speed. The objective is to activate all robots. The makespan is defined as the minimum time required to active the last robot; while the wake-up ratio is the maximum makespan among any finite number of robots in any primary positions.

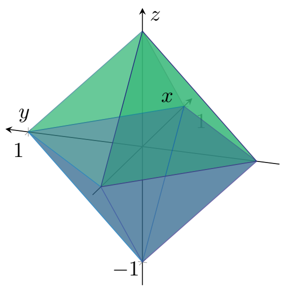


Figure 1: The unit ball in $(\mathbb{R}^3, \ell_1)$

Note that for any point set where the awake robot is within distance r of all sleeping robots and an upper bound c exists for the makespan, scaling the unit ball allows constructing a wake-up tree with makespan at most $r \times c$. Consequently, this results in a c-approximation algorithm for FTP, as r is a trivial lower bound on the makespan.

In $\mathbb{R}^2$, the unit ℓ_2-ball is a cyclic disc, while the unit ℓ_1-ball is a square. In $\mathbb{R}^3$, the unit ℓ_2-ball is a sphere, and the unit ℓ_1-ball appears as shown in Figure 1.

1.1 Related Work

Studies [1, 10, 12] confirm that FTP is NP-hard in $(\mathbb{R}^3, \ell_p)$ for all $p \geq 1$. Similarly, FTP is NP-hard for $(\mathbb{R}^2, \ell_2)$, though complexity results for other norms remain unresolved [1]. It is suspected that FTP is also NP-hard for $(\mathbb{R}^2, \ell_1)$ [3].

Let $\gamma_{d,p}$ be the wake-up time in $(\mathbb{R}^d, \ell_p)$. Despite the complexity of the problem, few studies focus on the wake-up ratio. Bonichon et al. showed that for $(\mathbb{R}^2, \ell_1)$, it is at most 5, leading to an upper bound of $5\sqrt{2}$ for $(\mathbb{R}^2, \ell_2)$ [6]. Consider an instance for FTP in $(\mathbb{R}^2, \ell_2)$ containing exactly 4 robots located at $(0, 1)$, $(0, -1)$, $(1, 0)$, and $(-1, 0)$. It is easy to see that the makespan of this instance is $1+2\sqrt{2} \approx 3.83$. In [6] it is conjectured that this input takes the maximum makespan among all instances; in other words $\gamma_{2,2} \approx 3.83$. Recently, the upper bound for $\gamma_{2,2}$ was reduced to 5.41 by [2] and 4.62 by [7].

Arkin et al. [4] formulated the FTP as a problem of constructing a rooted spanning tree with minimized

*TeIAS, Khatam University, sharareh.alipour@gmail.com
†TeIAS, Khatam University, aarash.ahadi.academic@gmail.com
‡Sharif University of Technology, kajal.baghestani@gmail.com

weighted depth. The root represents the initially active robot, which has one child, while the n inactive robots serve as nodes with at most two children each. Edges represent distances in the metric space. This structure, known as a wake-up tree, defines the wake-up time as its weighted depth. In Subsection 2.1 we use this spanning tree.

For $\gamma_{3,1}$ Alipour et al. proposed an algorithm with a wake-up ratio of 13, which corresponds to $13\sqrt{3}$ for $\gamma_{3,2}$ [2]. See table 1.

1.2 Our Contribution

For the Freeze-Tag problem in $(\mathbb{R}^2, \ell_2)$, we use the crown strategy from [2, 7] with a more detailed and slightly more complex version. For further details, we propose algorithms based on the positions of the three nearest asleep robots to the center; for each FTP instance, we select the one that minimizes the makespan.

In Section 2, we present the upper bound for $\gamma_{2,2}$; and in Section 3, we present the upper bound of 12 for $\gamma_{3,1}$. The upper bound of 12.76 for $\gamma_{3,2}$ is proved in Appendix **C**. See Table 1 for a summary of the results.

(Dim, Norm)	Previous Bound	New Bound
$(\mathbb{R}^2, \ell_2)$	4.62 [7]	4.31 (Thm. 1)
$(\mathbb{R}^3, \ell_1)$	13 [2]	12 (Thm. 5)
$(\mathbb{R}^3, \ell_2)$	22.52 [2]	12.76 (Thm. 6)

Table 1: Summary of results.

2 FTP in $(\mathbb{R}^2, \ell_2)$

In this section we show that 4.31 is an upper bound for the wake-up ratio of the FTP in $(\mathbb{R}^2, \ell_2)$.

Theorem 1 $\gamma_{2,2} \leq 4.31$.

In Subsection 2.1, we present several algorithms for solving the Freeze-Tag Problem. For any given FTP instance, we select the algorithm that yields the smallest makespan among the proposed options. We then use a computer program to compute the makespan across a range of input instances. Although the space of possible inputs for FTP is infinite, our program evaluates only a finite subset. However, we carefully select representative cases such that every arbitrary input has a makespan close to that of one of the tested instances. As a result, the program introduces a bounded approximation error. In Appendix **B**, we provide a detailed analysis of this approximation error.

We begin by introducing some concepts and notations. Consider a Freeze-Tag instance with n asleep robots, denoted $p_1, \ldots, p_n$, and a single active robot p_0 located at the origin in $(\mathbb{R}^2, \ell_2)$. Let $r_i = |p_i - O|_2$ denote the Euclidean distance from p_i to the origin O, with the assumption that $r_1 \leq r_2 \leq \cdots \leq r_n$. A circle $C(r)$ refers to the circle centered at O with radius r.

We define a crown, denoted by $\mathcal{R}(1-r, \theta)$, as a sector-shaped region bounded by an inner radius r, an outer radius 1, and a central angle θ (see Figure 4). It consists of all points p_i such that $r \leq |p_i - O|_2 \leq 1$ and whose angular coordinates fall within the sector defined by θ. The width of the crown, denoted by w, is given by $w = 1 - r$.

We use $|a - b|$ to denote the absolute value of the difference between a and b. Throughout this paper, we use p_i to refer both to the robot and its initial position.

2.1 Algorithms

We present several algorithms, each of which outperforms the others in specific instances. All of our algorithms are based on the crown strategy. In all algorithms we use the following lemmas from [7]. Let ϕ denote the golden ratio, defined as $\frac{1+\sqrt{5}}{2}$.

Lemma 2 *(Corollary 1 of [7]) There exists a strategy to wake up all of the robots in a crown of angle Θ and width w starting with one awake robot at a corner in time at most $\Theta + \phi w$.*

Lemma 3 *(Lemma 5 of [7]) There exists a strategy to wake up all of the robots in a crown of angle Θ and width w starting with two awake robots at a corner on the exterior side of the crown in time at most $\Theta + \left(\frac{\phi^4}{\phi^3+\Theta}\right) w$.*

Lemma 4 is an extension of Corollary 3 from [7]. Their proof relies on Proposition 14 from [6] and applies only to angles less than or equal to π. In Lemma 4, we demonstrate that the statement holds for any arbitrary angle. The proof is in Appendix **A**.

Lemma 4 *There exists a strategy to wake up all of the robots in a crown of angle Θ and width w starting with one awake robot at a corner on the interior side of the crown in time at most $\Theta + \left(1 + \frac{\phi^4}{\phi^3+\Theta}\right) w$.*

First, using a small improvement of algorithm of [7], we improve the bound of 4.62 to 4.54. Next by some other methods we decrease this value to 4.31.

A Small Improvement of The Algorithm of [7]

In [7], the bound 4.62 has been obtained by trade-off between two strategies. In the STRATEGY 1, p_0 first moves toward p_1. Once p_1 is activated, both p_0 and p_1 move to the origin O. Next, p_0 moves toward p_2 and activates it, while p_1 moves to a point t in the unit circle such that $\angle p_2 O t = \frac{2}{3}\pi$ and $\|O - t\| = r_2$. Finally each of p_0, p_1 and p_2 activates a crown $\mathcal{R}(1 - r_2, \frac{2}{3}\pi)$.

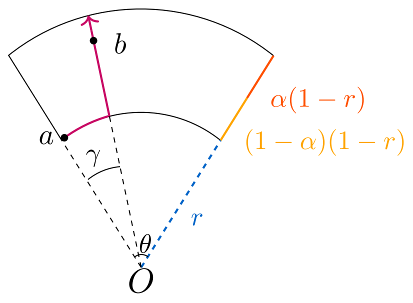


Figure 2: Activating a crown by one activated robot on the inner corner

Now we improve this strategy. Our improved strategy is denoted by three-crowns. Note that the step of coming back to the origin O can be improved. In fact, p_0 can move directly to p_2 and also p_1 can move directly to t. Next each of p_0, p_1 and p_2 activates a crown. However we should compute the optimal angle for the crowns that should be activated by these three points.

Let the angles of each of $\mathcal{R}_0$ and $\mathcal{R}_2$ be x; so the angle of $\mathcal{R}_1$ is $2\pi - 2x$. By Lemma 4, the total time of activating $\mathcal{R}_0$ and $\mathcal{R}_2$ is

$$T_{0,2} = r_1 + ||p_1 - p_2||_2 + x + \left(1 + \frac{\phi^4}{\phi^3 + x}\right)(1 - r_2).$$

Also, by Lemma 4 the total time of activating $\mathcal{R}_1$ is

$$T_1 = r_1 + ||p_1 - t||_2 + (2\pi - 2x) + \left(1 + \frac{\phi^4}{\phi^3 + 2\pi - 2x}\right)(1 - r_2).$$

We set x such that the maximum of $T_{0,2}$ and T_1 takes the minimum possible value (in this case, $T_{0,2} = T_1$). Thus, the three-crowns algorithm activates all robots in at most

$$\tau_1 := \min_{0 \le x \le \pi} \max\{T_{0,2}, T_1\}.$$

In the STRATEGY 2 of [7], first p_0 moves to p_1; next p_0 starts to activate a crown $\mathcal{R}_0 = \mathcal{R}(1 - r_1, \pi)$ and p_1 starts to activate a crown $\mathcal{R}_1 = \mathcal{R}(1 - r_1, \pi)$; such that the union of $\mathcal{R}_0, \mathcal{R}_1$ and $C(r_1)$ is the unit cycle. Since there is no asleep robot in $C(r_1)$, this strategy activates all robots. By Lemma 4, the total time of this strategy is

$$\tau_2 := r_1 + \pi + \left(1 + \frac{\phi^4}{\phi^3 + \pi}\right)(1 - r_1).$$

We call this strategy two-crowns algorithm.

The wake-up ratio of 4.62 is found by solving a max-min optimization. For each FTP input, the make-span is calculated using two strategies, and the one with the smallest make-span is chosen. The input with the largest make-span has a value of 4.62. However, by using the three-crowns algorithm (a modified version of STRATEGY 1) and the two-crowns algorithm (which is the same as STRATEGY 1), we can achieve a wake-up ratio of $\min\{\tau_1, \tau_2\} = 4.54$, where the minimum is given over all values of $0 \le r_1 \le r_2 \le 1$.

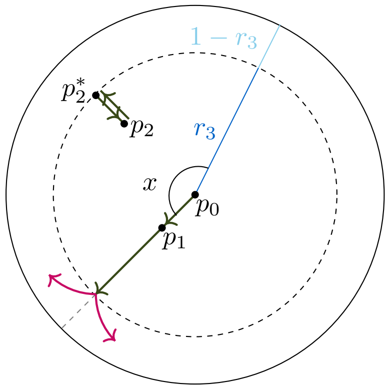


Figure 3: Two-crowns algorithm with respect to r_3

Two-crowns algorithm with respect to r_3

In the two-crowns algorithm, the active robot p_0 at the origin first moves toward p_1. Once p_1 is activated, there are now two active robots at its position. These robots then move $r_3 - r_1$ units in the direction of $\overrightarrow{Op_1}$. Next, the crown $\mathcal{R}(1 - r_3, 2\pi)$ is divided into two crowns

$$\mathcal{R}_{\text{left}} = \mathcal{R}(1 - r_3, 2\pi - x) \quad \text{and} \quad \mathcal{R}_{\text{right}} = \mathcal{R}(1 - r_3, x)$$

both starting from the radius that contains p_1. The value of x is determined later. One of these crowns is activated by p_0, and the other by p_1. Since there are no robots, except p_2, inside the circle $C(r_3)$, by activating p_2 and both $\mathcal{R}_{\text{left}}$ and $\mathcal{R}_{\text{right}}$, the entire unit circle is activated.

Without loss of generality, suppose that robot p_2 is on the right side of $\overrightarrow{Op_1}$. See Figure 3. Consider an instance $p_1, p_2^*, p_3, \dots, p_n$ of the FTP, similar to $p_1, p_2, \dots, p_n$, with one difference: the two robots p_2 and p_2^* and the point O belong to a line, and also $\|O - p_2^*\|_2 = r_3$. By Lemma 4, the crown $\mathcal{R}_{\text{right}}$ can be activated in

$$x + \left(1 + \frac{\phi^4}{\phi^3 + x}\right)(1 - r_3)$$

time units. We now present a strategy to activate p_2 and all robots except p_2^* in $\mathcal{R}_{\text{right}}$. Consider the wake-up tree of activating the crown $\mathcal{R}_{\text{right}}$ by the strategy of Lemma 4; the parent of p_2^* activates p_2. For more details, the parent of p_2^* first moves to the position of p_2^*, then moves to p_2, and finally both p_2 and the parent move to the position of p_2^*. Other nodes and edges of the wake-up tree do not change.

Since $\|p_2^* - p_2\|_2 = r_3 - r_2$, this strategy takes

$$T_{\text{right}} = x + \left(1 + \frac{\phi^4}{\phi^3 + x}\right)(1 - r_3) + 2(r_3 - r_2)$$

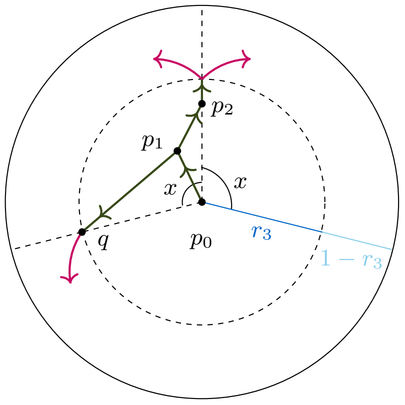


Figure 4: Three-crowns algorithm with respect to r_3

time units.

Activating $\mathcal{R}_{\text{left}}$ by p_1 spends

$$T_{\text{left}} = (2\pi - x) + \left(1 + \frac{\phi^4}{\phi^3 + 2\pi - x}\right)(1 - r_3)$$

time units. Also moving p_0 and p_1 to a corner of their crowns spends r_3 time units.

We set the angle x such that the maximum of T_{right} and T_{left} is minimized (in this case, $T_{\text{right}} = T_{\text{left}}$). Consequently, the total time of the algorithm is

$$r_3 + \min_{0 \le x \le \pi} \max\{T_{\text{right}}, T_{\text{left}}\}.$$

Three-crowns algorithm with respect to r_3

In the three-crowns algorithm, p_0 first moves toward p_1. Once p_1 is activated, there are two active robots at its position. Next, p_1 moves toward p_2 and activates it. Then, both p_1 and p_2 move outward along $\overrightarrow{Op_2}$ for $r_3 - r_2$ units.

At this point, each of p_1 and p_2 activates a crown: p_1 activates a crown $\mathcal{R}_1 = \mathcal{R}(1 - r_3, x)$ and p_2 activates a crown $\mathcal{R}_2 = \mathcal{R}(1 - r_3, x)$, where the value of x will be determined later.

Meanwhile, p_0 continues to activate robots in the remaining crown $\mathcal{R}(1 - r_3, 2\pi - 2x)$ as follows. Let q be the point such that $\angle p_2Oq = x$ and $\|O - q\|_2 = r_3$. In this case, p_0 moves toward q; next, p_0 activates a crown $\mathcal{R}_0 = \mathcal{R}(1 - r_3, \pi - 2x)$. See Figure 4.

By Lemma 4, the total time of activating $\mathcal{R}_1$ and $\mathcal{R}_2$ is

$$T_{1,2} = r_1 + \|p_1 - p_2\|_2 + (r_3 - r_2) + \left(x + \left(1 + \frac{\phi^4}{\phi^3 + x}\right)(1 - r_3)\right).$$

Also the total time of activating $\mathcal{R}_0$ is

$$T_0 = r_1 + \|p_1 - q\|_2 + \Big((2\pi - 2x) + \Big(1 + \frac{\phi^4}{\phi^3 + 2\pi - 2x}\Big)(1 - r_3)\Big).$$

We set the value x such that $T_{1,2} = T_0$. Consequently, the total time of the three-crowns algorithm is

$$\min_{0 \le x \le \pi} \max\{T_{1,2}, T_0\}.$$

(Two or Four)-Crowns Algorithm

First, we outline the overall structure of the four-crowns algorithm. In this algorithm, p_0 activates a robot q. Then, both p_0 and q activate new robots, denoted as p' and q', respectively. Finally, the ring is divided into the crowns $\mathcal{R}_{p_0}$, $\mathcal{R}_{p'}$, $\mathcal{R}_q$, and $\mathcal{R}_{q'}$, each assigned to one of p_0, p', q, and q'. Since p_0 and p' start their crowns simultaneously, the angles of $\mathcal{R}_{p_0}$ and $\mathcal{R}_{p'}$ should be equal in the optimum case. However, if $\angle p'Oq' = \pi - \beta$ for some $\beta > 0$, then before starting to activate the crowns, some of those 4 robots should move to new positions p'' and q'' such that $\angle p''Oq'' = \pi$. After this adjustment, the robots can start activating the crowns.

Let $p^\dagger$ be the robot that is farthest in angular distance from p_1 with the angle $\angle p_1Op^\dagger = \pi - \beta$ for some β. As discussed earlier, if β is small, the four-crowns algorithm may achieve an efficient wake-up time; however, this is not the case when β is large. In such case, the two-crowns algorithm is more sufficient. Specifically, it is enough to remove the region with angle β from both crowns, as no robots are located in these areas. In other words, in the two-crowns algorithm, two crowns with an angle of $\pi - \beta$ should be activated instead of the two crowns with an angle of π. So, based on the value of β, we decide to choose one of the algorithms (two or four)-crowns. These algorithms are discussed in detail in the next two parts.

Four-Crowns Algorithm with Respect to β

In this algorithm, first p_0 moves to p_2. Next, p_0 moves to p_1, and p_2 moves to $p^\dagger$, where $p^\dagger$ is the robot that is farthest in angular distance from p_1; let β be an angle such that $\angle p_1Op^\dagger = \pi - \beta$.

Let t be a point where $\angle p^\dagger Ot = \pi$ and $\|t - O\|_2 = r_3$. In the next step, both p_0 and p_1 move toward t to activate their crowns, $\mathcal{R}_0 = \mathcal{R}(1 - r_3, x)$ and $\mathcal{R}_1 = \mathcal{R}(1 - r_3, x)$, respectively. In order to activate the two other crowns $\mathcal{R}_2 = \mathcal{R}(1 - r_3, \pi - x)$ and $\mathcal{R}_\dagger = \mathcal{R}(1 - r_3, \pi - x)$ (those of p_2 and $p^\dagger$), note that if the robot $p^\dagger$ is positioned near the inner arc of $\mathcal{R}_\dagger$, it is more efficient for p_2 and $p^\dagger$ to start their crowns from the inner corner of the crown rather than from the outer corner, i.e., we consider the corner of these two crowns which is the nearest one to $p^\dagger$ and in the worst case $p^\dagger$ is exactly

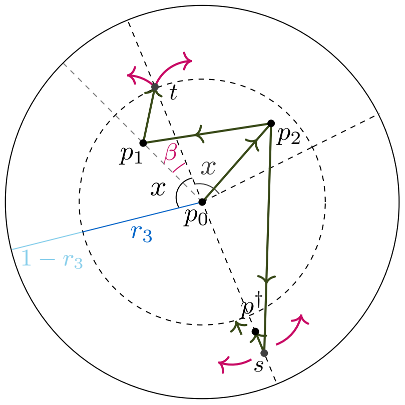


Figure 5: Four crowns are activated according to the four-crowns algorithm

in the middle, so we consider this case and first we go to the middle point s and then continue to the corner which is closer to $p^\dagger$ (in this way we activate $p^\dagger$ too). In other words, s is a point on the radius $Op^\dagger$, with a distance of $r_3 + \frac{1-r_3}{2}$ from O. Next, p_2 moves to $p^\dagger$ in the direction of $\overrightarrow{Op^\dagger}$ or $\overleftarrow{Op^\dagger}$. Afterward, both p_2 and $p^\dagger$ continue along this direction until they reach a corner of their crowns. At this point, p_2 and $p^\dagger$ begin activating their crowns. See Figure 5.

Here we compute an upper bound for the makespan of the four-crowns algorithm. By Lemma 4, activating $\mathcal{R}_0$ and $\mathcal{R}_1$ is possible in

$$T_{0,1} = r_2+\|p_2-p_1\|_2+\|p_1-t\|_2+\Big(x+\Big(1+\frac{\phi^4}{\phi^3+x}\Big)(1-r_3)\Big)$$

time units. Also by Lemma 2, activating $\mathcal{R}_2$ and $\mathcal{R}_\dagger$ is possible in

$$T_{2,\dagger} = r_2+\|p_2-s\|_2+\frac{1-r_3}{2}+\Big((\pi-x)+(1+\phi)(1-r_3)\Big)$$

time units; we recall that $\phi = \frac{1+\sqrt{5}}{2}$. As the previous algorithms, we set the angle x such that $T_{0,1} = T_{2,\dagger}$. In other words the wake-up time of four-crowns algorithm is

$$\min_{0\le x\le\pi} \max\{T_{0,1}, T_{2,\dagger}\}.$$

Two-Crowns Algorithm with Respect to β

This algorithm is similar to the second strategy in [7]; the only difference is that both crowns have an angle of $\pi-\beta$ (instead of π).

Therefore the total time of this algorithm is

$$r_1 + (\pi-\beta) + \Big(1+\frac{\phi^4}{\phi^3+\pi-\beta}\Big)(1-r_1).$$

Consequently, we consider both the four-crowns and two-crowns algorithms and select the one with the shorter wake-up time.

Algorithms with activating p_3 before activating crowns

In some of the previous algorithms, the thickness of the crowns is $1-r_3$. Therefore, if r_3 is close to 1, activating the crowns becomes fast. Here, we present some similar algorithms that works better for small values of r_3.

Note that if r_3 is small, then r_1 and r_2 will also be small. As a result, we can apply new algorithms similar to the three-crowns algorithm and four-crowns algorithm.

For more details, as a three-crowns type algorithm, p_0 can first activate one of p_1, p_2, or p_3. Then, each of the two active robots activates one of the two asleep robots among p_1, p_2, and p_3. Let Ω be the angle between these two robots with respect to the origin O. In the next step, each of the four active robots moves by the angle $\frac{1}{2}|\pi-\Omega|$ to reach two points on the circle $C(r_3)$. Finally, each of the four robots begins to activate a crown; two of them which are in the same position activate crowns with the angle x, and the other two activate crowns with the angle $\pi-x$.

With respect to the first activated robot after p_0, the previous paragraph gives us 3 algorithms. The fastest one is determined by the positions of p_1, p_2 and p_3.

As a three-crowns type algorithm, p_0 can first move to one of p_1, p_2 or p_3; next we can have a strategy similar to the three-crowns algorithm using one of two asleep robots among p_1, p_2 and p_3. Therefore we have 6 algorithms. Note that in each of these 6 algorithms, the width of the crowns is $1-r_i$, where $\{p_1,p_2,p_3\}-\{p_i\}$ are two activated robots in the three-crowns algorithm.

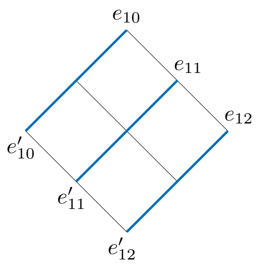
2.2 The Main Algorithm

For each instance of the Freeze-Tag Problem, we apply all the discussed algorithms and select the one that minimizes the makespan time. In Appendix **B**, we compute the wake-up time for the main algorithm.

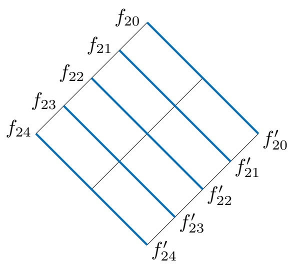
3 FTP in $\mathbb{R}^3$

We provide upper bounds for the wake-up ratio of FTP in $(\mathbb{R}^3,\ell_1)$ and $(\mathbb{R}^3,\ell_2)$ using a similar approach for both cases.

Given n points inside a unit ℓ_1-ball (ℓ_2-ball), we propose an algorithm to wake up the robots. Let $\mathcal{D}$ be the unit ℓ_1-ball (ℓ_2-ball) in $\mathbb{R}^3$, and define $\mathcal{D}^+$ as the part of $\mathcal{D}$ with positive z-coordinates, with $\mathcal{D}^- = \mathcal{D}\setminus\mathcal{D}^+$. Without loss of generality, assume $\mathcal{D}^+$ contains at least as many robots as $\mathcal{D}^-$. The algorithm first activates all robots in $\mathcal{D}^+$. Then, each robot in $\mathcal{D}^+$ activates at most one robot in $\mathcal{D}^-$ simultaneously. We now present the details for the ℓ_1-ball and ℓ_2-ball cases.



(a) $\mathcal{Q}_2$ with 4 squares



(b) $\mathcal{Q}_3$ with 8 rectangles

Figure 6: Two partitions of the square $\mathcal{S}$

3.1 FTP in $(\mathbb{R}^3, \ell_1)$

To activate the robots in $\mathcal{D}^+$, the active robot p_0 moves to the asleep robot with the smallest z-coordinate, say p_1 (ties are broken arbitrarily). With two active robots at p_1, we divide $\mathcal{D}^+$ into two regions, assigning one to each robot. We will explain in more detail how $\mathcal{D}^+$ is divided. Two awake robots move toward the asleep robot with the smallest z-coordinate in their regions. In each step of the algorithm ties are broken arbitrarily. If no asleep robots are left in a region, the assigned robot stops. When a robot activates another in its region, the region splits, and the two active robots continue the process by handling their parts until all robots are awake.

We explain how the partitioning works. We start with the square $\mathcal{S}$ with corners $(1,0,0)$, $(0,1,0)$, $(-1,0,0)$, and $(0,-1,0)$. We define a sequence of partitions $\mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \ldots$ of the area of $\mathcal{S}$. Initially, $\mathcal{Q}_0$ consists of the entire square $\mathcal{S}$. At each step, $\mathcal{Q}_i$ is obtained by subdividing each region in $\mathcal{Q}_{i-1}$ into two regions as follows.

For every natural numbers k and i consider the following points in the plane $z = 0$.

$$e_{ki} = (\frac{i}{2^k}, 1 - \frac{i}{2^k}), \qquad e'_{ki} = (\frac{i}{2^k} - 1, -\frac{i}{2^k}),$$
$$f_{ki} = (-\frac{i}{2^k}, 1 - \frac{i}{2^k}), \qquad f'_{ki} = (1 - \frac{i}{2^k}, -\frac{i}{2^k}),$$

and
$$E_k = \{\overline{e_{ki}e'_{ki}} \,|\, 0 \le i \le 2^k\}, \qquad F_k = \{\overline{f_{ki}f'_{ki}} \,|\, 0 \le i \le 2^k\}.$$

For every integer $t \ge 1$, the partition $\mathcal{Q}_{2t-1}$ divides $\mathcal{S}$ into 2^{2t-1} rectangles using the line segments of $E_{t-1} \cup F_t$. The partition $\mathcal{Q}_{2t}$ then divides $\mathcal{S}$ into 2^{2t} squares using the segments of $E_t \cup F_t$. Therefore by definition, $\mathcal{Q}_0$ contains the single square $\mathcal{S}$, and $\mathcal{Q}_1$ consists of two rectangles. See Figure 6.

In the first step, p_0 starts at the origin of $\mathcal{D}$ and is responsible for the entire $\mathcal{D}^+$. It moves toward p_1, the asleep robot with the smallest z-coordinate in $\mathcal{D}^+$. Using $\mathcal{Q}_1$, we divide S into two rectangles. The planes parallel to the z-axis that passes through the segments of E_1, split $\mathcal{D}^+$ into two parts; p_0 takes one, and p_1 takes the other.

In the next step, p_1 moves toward p_2, the asleep robot with the smallest z-coordinate in its assigned part. Once p_2 is awake, we partition the corresponding region using $\mathcal{Q}_1$, splitting it into two parts with planes that passes trough segments of E_1 and are parallel to the z-axis. At this time, p_1 is responsible for one part, and p_2 for the other. This process repeats until all points are awake.

We compute an upper bound for the wake-up time of the algorithm. Consider the wake-up tree for the points in $\mathcal{D}^+$. Let $p_0 = q_0, q_1, \ldots, q_m$ be any arbitrary path from the root to a leaf in the wake-up tree, i.e., in each node of this path there is a robot, either as same as its father or activated by its father. The robot q_{i+1} is activated by one of the 2 active robots in its parent node.

We provide an upper bound for the wake-up time along this path. Let the coordinates of q_i be (x_i, y_i, z_i). We provide an upper bound for $\sum_{i=0}^{m-1} |x_i - x_{i+1}| + |y_i - y_{i+1}| + |z_i - z_{i+1}|$. In the algorithm, $z_i \ge z_{i-1}$. Since $z_0 = 0$ and the maximum z-coordinate in $\mathcal{D}^+$ is at most 1, it holds that

$$\sum_{i=0}^{m-1} |z_i - z_{i+1}| \le 1. \tag{1}$$

Now, we compute an upper bound for $\sum_{i=0}^{m-1} |x_i - x_{i+1}| + |y_i - y_{i+1}|$.

Let a_i be the projection of q_i onto the plane $z = 0$, i.e., $a_i = (x_i, y_i)$. So, $\|a_i - a_{i+1}\|_1 = |x_i - x_{i+1}| + |y_i - y_{i+1}|$. In the first step of algorithm, since q_0 is at the origin, we have $\|a_0 - a_1\|_1 \le 1$. For $i \ge 1$, when there are two active robots in q_i, then these two robots should activate the region in $\mathcal{Q}_i$ that contains them. So, the diameter of this region is an upper bound for $\|a_i - a_{i+1}\|_1$. If i is even then the diameter of each region of $\mathcal{Q}_i$ is $2^{\frac{2-i}{2}}$ and for odd i, the diameter of $\mathcal{Q}_i$ is $2^{\frac{3-i}{2}}$. This implies

$$\|a_i - a_{i+1}\|_1 \le \begin{cases} 2^{\frac{2-i}{2}} & \text{for even } i, \\ 2^{\frac{3-i}{2}} & \text{for odd } i. \end{cases}$$

Therefore

$$\sum_{i=1}^{m-1} \|a_i - a_{i+1}\|_1 \le$$
$$1 + (2 + 2 + 1 + 1 + \tfrac{1}{2} + \tfrac{1}{2} + \ldots) = 9. \tag{2}$$

So, by 1 and 2, the wake-up time of $\mathcal{D}^+$ is at most $9 + 1 = 10$. At the final step, since the number of robots in $\mathcal{D}^+$ is not fewer than the number of robots in $\mathcal{D}^-$, each robot in $\mathcal{D}^+$ activates at most one robot in $\mathcal{D}^-$. Thus, this step requires at most 2 time units, as the diameter of $\mathcal{D}$ is 2. So, the wake-up ratio of FTP in $(\mathbb{R}^3, \ell_1)$ is at most 12.

Theorem 5 $\gamma_{3,1} \le 12$.

References

[1] Z. Abel, H. A. Akitaya, and J. Yu. Freeze tag awakening in 2d is np-hard. In *27th Fall Workshop on Computational Geometry*, pages 105–107, 2017.

[2] S. Alipour, K. Baghestani, M. Mirzaei, and S. Sahraei. Geometric freeze-tag problem. *CoRR*, abs/2412.19706, 2024.

[3] E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. Mitchell, and M. Skutella. The freeze-tag problem: how to wake up a swarm of robots. *Algorithmica*, 46:193–221, 2006.

[4] E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. B. Mitchell, and M. Skutella. The freeze-tag problem: how to wake up a swarm of robots. In D. Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 568–577. ACM/SIAM, 2002.

[5] E. M. Arkin, M. A. Bender, and D. Ge. Improved approximation algorithms for the freeze-tag problem. In A. L. Rosenberg and F. M. auf der Heide, editors, *SPAA 2003: Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 7-9, 2003, San Diego, California, USA (part of FCRC 2003)*, pages 295–303. ACM, 2003.

[6] N. Bonichon, A. Casteigts, C. Gavoille, and N. Hanusse. Freeze-tag in l_1 has wake-up time five with linear complexity. In D. Alistarh, editor, *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*, volume 319 of *LIPIcs*, pages 9:1–9:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[7] N. Bonichon, C. Gavoille, N. Hanusse, and S. Odak. Euclidean Freeze Tag Problem. In *The Canadian Conference on Computational Geometry (CCCG 2024)*, St. Catharines, Canada, July 2024.

[8] A. M. Bruckstein, C. L. Mallows, and I. A. Wagner. Probabilistic pursuits on the grid. *The American mathematical monthly*, 104(4):323–343, 1997.

[9] D. W. Gage. Minimum-resource distributed navigation and mapping. In H. Choset, D. W. Gage, and M. R. Stein, editors, *Mobile Robots XV and Telemanipulator and Telepresence Technologies VII, Boston, MA, USA, November 5, 2000*, volume 4195 of *SPIE Proceedings*, pages 96–103. SPIE, 2000.

[10] M. P. Johnson. Easier hardness for 3d freeze-tag. *Proc. 27th Fall Worksh. Comp. Geom.(FWCG)*, 2017.

[11] J. Könemann, A. Levin, and A. Sinha. Approximating the degree-bounded minimum diameter spanning tree problem. *Algorithmica*, 41(2):117–129, 2005.

[12] L. L. C. Pedrosa and L. de Oliveira Silva. Freeze-tag is np-hard in 3d with l_1 distance. In C. G. Fernandes and S. Rajsbaum, editors, *Proceedings of the XII Latin-American Algorithms, Graphs and Optimization Symposium, LAGOS 2023, Huatulco, Mexico, September 18-22, 2023*, volume 223 of *Procedia Computer Science*, pages 360–366. Elsevier, 2023.

[13] K. Sugihara and I. Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *J. Field Robotics*, 13(3):127–139, 1996.

[14] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999.

[15] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. Efficiently searching a graph by a smell-oriented vertex process. *Ann. Math. Artif. Intell.*, 24(1-4):211–223, 1998.

[16] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Trans. Robotics Autom.*, 15(5):918–933, 1999.

[17] E. N. Yazdi, A. Bagheri, Z. Moezkarimi, and H. Keshavarz. An o(1)-approximation algorithm for the 2-dimensional geometric freeze-tag problem. *Inf. Process. Lett.*, 115(6-8):618–622, 2015.

Appendix

A. Proof of Lemma 4

Let a be the active robot at the corner of the inner arc of the crown. Let b be the closest angular point to a, and let $\gamma = \angle aOb$. First, a moves along the inner arc at the angle γ. See Figure 2. This takes $T_1 = (1-w)\gamma$ time units. Next, a moves to b, and both active robots follow the direction $\overrightarrow{Ob}$ until they reach the outer arc of the crown. This takes $T_2 = w$ time units. By Lemma 3, these two robots can activate the rest of the remaining crown $\mathcal{R}(w, \Theta-\gamma)$ in $T_3 = (\Theta-\gamma) + \left(\frac{\phi^4}{\phi^3+(\Theta-\gamma)}\right) w$ time units. To prove the lemma it suffices to show that the total time for these steps is at most $\Theta + \left(1+\frac{\phi^4}{\phi^3+\Theta}\right) w$. We have

$$\frac{\phi^4}{\phi^3+\Theta-\gamma} - \frac{\phi^4}{\phi^3+\Theta} = \frac{\phi^4\gamma}{(\phi^3+\Theta-\gamma)(\phi^3+\Theta)} \le \frac{\phi^4\gamma}{\phi^3\phi^3} \le \gamma.$$

So, it holds that

$$\begin{aligned} T_1+T_2+T_3 &= (1-w)\gamma + w + (\Theta-\gamma) + \Big(\frac{\phi^4}{\phi^3+\Theta-\gamma}\Big)w \\ &= \Theta + \Big(1-\gamma+\frac{\phi^4}{\phi^3+\Theta-\gamma}\Big)w \le \Theta + \Big(1+\frac{\phi^4}{\phi^3+\Theta}\Big)w. \end{aligned}$$

B. Computing the Wake-Up Ratio in $(\mathbb{R}^2, \ell_2)$

Due to the mathematical complexity involved in computing the maximum makespan across all our algorithms, we approximated this value using a computational approach.

Consider an input of FTP in $(\mathbb{R}^2, \ell_2)$. The makespan times of our algorithms are determined by the parameters r_1, r_2, r_3, and the angles $\mu_{1,2} = \angle p_1Op_2$, $\mu_{1,3} = \angle p_1Op_3$, and $\mu_{2,3} = \angle p_2Op_3$. These six values collectively specify the positions of p_1, p_2, and p_3.

It is important to note that, for the two-crowns algorithm, when $r_2 > 0.87$, the resulting makespan is given by:

$$1 + \pi + \left(\frac{\phi^4}{\phi^3 + \pi}\right)(1 - 0.87) \leq 4.27.$$

Thus, we can assume that $r_1 \leq r_2 \leq 0.87$ in this context.

Additionally, note that either $\mu_{2,3} = \mu_{1,2} + \mu_{1,3}$ or $\mu_{2,3} = |\mu_{1,2} - \mu_{1,3}|$. We consider both cases for $\mu_{2,3}$ in the computational program.

The performance of the (two or four)-crowns algorithm is influenced by the value of the angle β. Therefore, we consider the maximum makespan over all $\beta \in [0, \pi]$.

In our computer program [1], we divide the interval $[0, 1]$ into 200 equal parts and the angle 2π into 628 equal parts. Specifically, the program computes the makespan time for every $r_1, r_2, r_3 \in \{0, \frac{1}{200}, \frac{2}{200}, \dots, 1\}$ with $r_1 \leq r_2 \leq r_3$, and every $\mu_{1,2}, \mu_{1,3} \in \{0, 0.01, 0.02, \dots, 3.14 \approx \pi\}$; see Figure 7 (a). For each combination of these parameters, the program computes the minimum makespan time among all our algorithms.

The final output of the program is the maximum makespan time among all values of $r_1, r_2, r_3, \beta, \mu_{1,2}, \mu_{1,3}$, and $\mu_{2,3}$. The output of our computer program is less than 4.2773. In other words,

$$\max \min \text{(makespan time)} \leq 4.2773,$$

where the maximum is taken over all values of $r_1, r_2, r_3, \beta, \mu_{1,2}, \mu_{1,3}$, and $\mu_{2,3}$, and the minimum is taken over all algorithms in Subsection 2.1.

The positions of the robots are arbitrary within the unit ball, but we consider only a finite set of such positions. Consequently, the approximation of 4.2773 introduces some error. We now present an upper bound for this error.

Our intervals divide the unit ball into 31,500 blocks, but only 2×100 of these blocks (corresponding to the first and last intervals of the angles) have intersections. Consider an input of FTP, where p_1, p_2, and p_3 belong to blocks B_1, B_2, and B_3, respectively. By our division, for any block there exists a rectangle with dimensions at most $\frac{1}{100} \times \frac{1}{200}$. Therefore, for each $1 \leq i \leq 3$, there is a corner c_i of block B_i such that $\|p_i - c_i\|_2 \leq \epsilon$, where

$$\epsilon = \frac{1}{2}\sqrt{\left(\frac{1}{100}\right)^2 + \left(\frac{1}{200}\right)^2},$$

See Figure 7 (b). The main algorithm is as follows: Apply the computer program to the instance $c_1, c_2, c_3, p_4, p_5, \dots, p_n$; then, add

$$2\|p_1 - c_1\|_2 + 2\|p_2 - c_2\|_2 + 2\|p_3 - c_3\|_2$$

to the output (makespan time). This value is added to the makespan because, for each $1 \leq i \leq 3$, in order to activate c_i with a robot q, the robot q can move to c_i, then to p_i, and finally return to c_i. These steps add $\|p_i - c_i\|_2$ times the unit to the makespan.

Consequently, we have

$$2\epsilon_1 + 2\epsilon_2 + 2\epsilon_3 \leq \frac{3\sqrt{5}}{200} \leq 0.0336$$

as an upper bound for the error of our program. Thus, our total bound is

[1] Available on GitHub

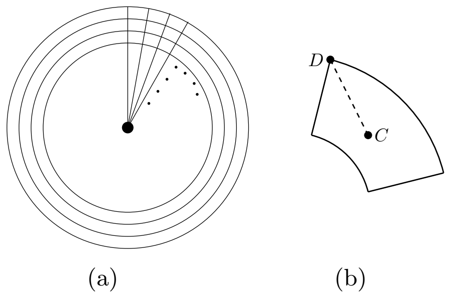


Figure 7: (a) dividing a unit disc into 62800 parts, (b) the distance of each point in a part from its closet corner is at most $|CD| = \epsilon$.

$$4.2773 + 0.0336 \approx 4.31.$$

C: FTP in $(\mathbb{R}^3, \ell_2)$

Our approach for $(\mathbb{R}^3, \ell_2)$ is similar to $(\mathbb{R}^3, \ell_1)$. Let $\mathcal{D}$ be the unit ℓ_2-ball in $\mathbb{R}^3$. Define $\mathcal{D}^+$ as the part of $\mathcal{D}$ with positive z-coordinates, and let $\mathcal{D}^- = \mathcal{D} \setminus \mathcal{D}^+$. Without loss of generality, assume $\mathcal{D}^+$ contains at least as many robots as $\mathcal{D}^-$.

The initial active robot first activates all robots in $\mathcal{D}^+$. Then, each activated robot in $\mathcal{D}^+$ activates a corresponding robot in $\mathcal{D}^-$.

Now we explain the strategy for activating the robots in $\mathcal{D}^+$ First, p_0 moves toward a point p_1 with minimum z coordinates in $\mathcal{D}^+$ now that there are two active robots, $\mathcal{D}^+$ is partitioned into two parts and each of the two robots is responsible for its own part, i.e, it moves toward the asleep robot with minimum z coordinates and activate that robot. We continue this process until all the robots get awake. The way that we partition the sphere is similar as for $(\mathbb{R}^3, \ell_1)$. For integers $k \geq 0$ and $0 \leq i \leq 2^k$ let

$$e_{ki} = (-1 + \frac{i}{2^{k-1}}, 1, 0), \qquad e'_{ki} = (-1 + \frac{i}{2^{k-1}}, -1, 0),$$
$$f_{ki} = (1, -1 + \frac{i}{2^{k-1}}, 0), \qquad f'_{ki} = (-1, -1 + \frac{i}{2^{k-1}}, 0),$$

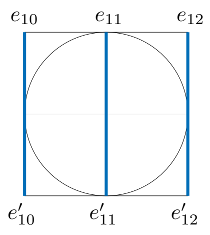
and

$$E_k = \{\overline{e_{ki}e'_{ki}} \,|\, 0 \leq i \leq 2^k\}, \quad F_k = \{\overline{f_{ki}f'_{ki}} \,|\, 0 \leq i \leq 2^k\}.$$

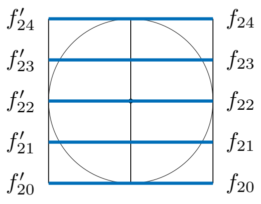
Let $\mathcal{S}'$ be a square with the corners $(1, 1, 0), (-1, 1, 0), (-1, -1, 0)$, and $(1, -1, 0)$. We define a sequence of partitioning $\mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \dots$ of the unit ℓ_2-disk in the plane $z = 0$ as follows.

For every integer $t \geq 1$, the partition $\mathcal{Q}_{2t-1}$ divides $\mathcal{S}'$ into 2^{2t-1} rectangles using the line segments of $E_{t-1} \cup F_t$. The partition $\mathcal{Q}_{2t}$ then divides $\mathcal{S}'$ into 2^{2t} squares using the segments of $E_t \cup F_t$. Therefore by definition, $\mathcal{Q}_0$ contains the single square $\mathcal{S}'$, and $\mathcal{Q}_1$ consists of two rectangles. See Figure 8.

We compute an upper bound for the wake-up time of the algorithm. Consider the wake-up tree for the points in $\mathcal{D}^+$. Let $p_0 = q_0, q_1, \dots, q_m$ be any arbitrary path from the root to a leaf in the wake-up tree, i.e., in each node of this path



(a) $\mathcal{Q}_2$ with 4 squares



(b) $\mathcal{Q}_3$ with 8 rectangles

Figure 8: Two partitions of the square $\mathcal{S}'$

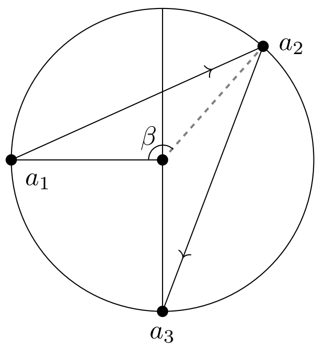


Figure 9: The maximum value of $\|a_1 - a_2\|_2 + \|a_2 - a_3\|_2$ is achieved when each of the points a_1, a_2, and a_3 is at a distance of 1 from the origin and $\angle a_1 O a_3 = \frac{3}{2}\pi$

there are 2 active robots, one was activated by the other one. The robot q_{i+1} is activated by one of the 2 active robots in its parent node. We provide an upper bound for the wake-up time along this path, $\sum_{i=0}^{m-1} \|q_{i+1} - q_i\|_2$. Let $q_i = (x_i, y_i, z_i)$, then

$$\sum_{i=0}^{m-1} \|q_{i+1} - q_i\|_2$$

$$= \sum_{i=0}^{m} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 + (z_i - z_{i+1})^2}$$

$$\leq \sum_{i=0}^{m} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} + |z_i - z_{i+1}|.$$

According to our algorithm $\sum_{i=0}^{m-1} |z_i - z_{i+1}| \leq 1$. Now, let $a_i = (x_i, y_i)$ be the projection of q_i on the plane $z = 0$. So, When q_i moves toward q_{i+1}, we only need to compute $\|a_i - a_{i+1}\|_2$, which gives an upper bound for

$$\sum_{i=0}^{m-1} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}.$$

Without loss of generality assume that $a_1 = (x_1, 0)$. In the first step, since q_0 is at the origin, we have $\|a_0 - a_1\|_2 \leq 1$. Next we show an upper bound for $\|a_1 - a_2\|_2 + \|a_2 - a_3\|_2$. Obviously the maximum happens when the robots are on the boundary of $\mathcal{S}'$, assuming that $\angle a_1 a_2 a_3 = \alpha$, then $\|a_1 - a_2\|_2 + \|a_2 - a_3\|_2 = 2\sin\frac{\alpha}{2} + 2\sin\frac{3\pi - 2\alpha}{4}$. The maximum happens when $\alpha = \frac{3\pi}{4}$. See Figure 9. Thus,

$$\|a_1 - a_2\|_2 + \|a_2 - a_3\|_2 \leq 4\sin\frac{3\pi}{8}. \quad (3)$$

For $i \geq 3$, when there are two active robots in q_i, then these two robots should activate the region in $\mathcal{Q}_i$ that contains them. The diameter of this region is an upper bound for $\|a_i - a_{i+1}\|_2$. If i is even then the diameter of each region of $\mathcal{Q}_i$ is $2^{\frac{2-i}{2}}\sqrt{5}$ and for odd i, the diameter of $\mathcal{Q}_i$ is $2^{\frac{3-i}{2}}\sqrt{2}$. This implies

$$\|a_i - a_{i+1}\|_2 \leq \begin{cases} 2^{\frac{2-i}{2}}\sqrt{5} & \text{for even } i \\ 2^{\frac{3-i}{2}}\sqrt{2} & \text{for odd } i \end{cases}$$

Therefore

$$\sum_{i=3}^{m-1} \|a_i - a_{i+1}\|_2 \leq$$

$$\sqrt{2} + \frac{\sqrt{5}}{2} + \frac{\sqrt{2}}{2} + \frac{\sqrt{5}}{4} + \cdots = 2\sqrt{2} + \sqrt{5}. \quad (4)$$

At the final step, since the number of robots in $\mathcal{D}^+$ is not fewer than the number of robots in $\mathcal{D}^-$, each robot in $\mathcal{D}^+$ activates at most one robot in $\mathcal{D}^-$. Thus, this step requires at most 2 time units, as the diameter of $\mathcal{D}$ is 2.

Thus, by 3 and 4, the wake-up time of $\mathcal{D}$ is at most

$$1 + 1 + 4\sin\frac{3\pi}{8} + 2\sqrt{2} + \sqrt{5} + 2 \leq 12.7601.$$

Consequently, the wake-up ratio of FTP in $(\mathbb{R}^3, \ell_2)$ is at most 12.7601.

Theorem 6 $\gamma_{3,2} \leq 12.7601$.

Note that our algorithm for computing the upper bound for $\sum_{i=0}^{m-1} \|a_i - a_{i+1}\|_2$ is similar to the algorithm in [17], where they give an upper bound of 10.06 for the wake-up ratio of FTP in $(\mathbb{R}^2, \ell_2)$. But we present an improved analysis, in more details we give a better upper bound on the maximum of $\|a_1 - a_2\|_2 + \|a_2 - a_3\|_2$. This helps to improve the upper bound.

Minimum-Weight Half-Plane Hitting Set[*]

Gang Liu[†] Haitao Wang[†]

Abstract

Given a set P of n weighted points and a set H of n half-planes in the plane, the hitting set problem is to compute a subset P' of points from P such that each half-plane contains at least one point from P' and the total weight of the points in P' is minimized. The previous best algorithm solves the problem in $O(n^{7/2}\log^2 n)$ time. In this paper, we present a new algorithm with runtime $O(n^{5/2}\log^2 n)$.

1 Introduction

Let H be a set of n half-planes and P a set of n points in the plane such that each point has a weight. We say that a point of P *hits* a half-plane of H if the half-plane covers the point. A subset P' of P is a *hitting set* for H if every half-plane of H is hit by a point in P'; P' is a *minimum-weight* hitting set if the total weight of all points of P' is the smallest among all hitting sets of H.

In this paper, we consider the *half-plane hitting set problem* which is to compute a minimum-weight hitting set for H. The problem has been studied before [3, 8, 11–13]. Har-Peled and Lee [8] first proposed an $O(n^6)$ time algorithm. Liu and Wang [11] gave an improved solution by reducing the problem to $O(n^2)$ instances of the *lower-only* half-plane hitting set problem, where all half-planes are lower ones. Consequently, if the lower-only problem can be solved in $O(T)$ time, the general problem is solvable in $O(n^2 \cdot T)$ time. Liu and Wang [11] derived an $O(n^2 \log n)$ time algorithm for the lower-only problem, thus solving the general half-plane hitting set problem in $O(n^4 \log n)$ time. More recently, Liu and Wang [12] proposed an improved $O(n^{3/2}\log^2 n)$ time algorithm for the lower-only problem, leading to an $O(n^{7/2}\log^2 n)$ time solution for the general problem.

We present a new algorithm of $O(n^{5/2}\log^2 n)$ time. This improves the previous best solution [12] by a linear factor. More specifically, our algorithm runs in $O(\kappa \cdot n^{3/2}\log^2 n)$, where κ is the minimum number of points of P covered by any half-plane of H. As such, our algorithm could be more efficient if κ is relatively small in certain applications. For example, if there exists a half-plane in H that covers only $O(1)$ points of P, then the time complexity of our algorithm is bounded by $O(n^{3/2}\log^2 n)$, matching the performance of the previous best algorithm for the lower-only case [12].

1.1 Related work

In the *unweighted* half-plane hitting set problem, all points have the same weight. As it is a special case of the weighted problem, all above results are applicable to the unweighted case. Recently, Liu and Wang proposed an $O(n \log n)$-time algorithm for the unweighted case [14]. Additionally, Wang and Xue [20] proved a lower bound of $\Omega(n \log n)$ time under the algebraic decision tree model for the unweighted case, even if all half-planes are lower ones. Therefore, Liu and Wang's algorithm is optimal.

A closely related problem is the half-plane *coverage* problem. Given P and H as above with each half-plane associated with a weight, the problem is to compute a minimum-weight subset of half-planes whose union covers all points of P. Logan and Wang [18] reduced this problem to $O(n^2)$ instances of the *lower-only* half-plane coverage problem, where all half-planes are lower ones. Consequently, if the lower-only problem can be solved in $O(T')$ time, then the general problem is solvable in $O(n^2 \cdot T')$ time. Notice that the lower-only coverage problem is dual to the lower-only hitting set problem (i.e., the two problems can be reduced to each other in linear time). Therefore, an algorithm for the lower-only hitting set problem can be used to solve the lower-only coverage problem with the same time complexity, and vice versa. In fact, the $O(n^{3/2}\log^2 n)$ time algorithm by Liu and Wang [12] for the lower-only hitting set problem was originally described to solve the lower-only coverage problem.

It should be noted that in the general case where both upper and lower half-planes are present in H, the hitting set problem and the coverage problem are no longer dual to each other. While the unweighted general half-plane hitting set problem can be solved in $O(n \log n)$ time [14], the currently best algorithm for the unweighted general half-plane coverage problem, which is due to Wang and Xue [20], runs in $O(n^{4/3}\log^{5/3} n \log^{O(1)} \log n)$ time.

The hitting set problem, as a fundamental problem, has been extensively explored in the literature. Various geometric versions of this problem have garnered significant attention, and many are known to be NP-

[*]This research was supported in part by NSF under Grants CCF-2300356. A full version of this paper is available at http://arxiv.org/abs/2506.16979.

[†]Kahlert School of Computing, University of Utah, Salt Lake City, USA, {gangliu,haitao.wang}@cs.utah.edu

hard [2,3,6,7,10,15,16]. For instance, in the disk hitting set problem, given a set of disks and a set of points in the plane, the goal is to compute a smallest subset of points that hit all the disks. This problem remains NP-hard even when all disks have the same radius [5,9,16]. However, certain variants of the problem can be solved in polynomial time. Liu and Wang [11] studied a line-constrained version, where the centers of all disks lie on a line, and they developed a polynomial-time algorithm for this case. For related problems in similar geometric settings, see also [12,13,17,18].

1.2 Our approach

To solve the half-plane hitting set problem on P and H, instead of reducing the problem to $O(n^2)$ instances of the lower-only case as in the previous work [11], we follow a problem reduction method in [14] for the unweighted case and reduce the problem to the following *circular-point coverage* problem: Given a set $\mathcal{A}$ of weighted arcs and a set B of points on a circle C, compute a minimum-weight subset of arcs whose union covers all points. In our reduction, the points of B are defined by the half-planes of H and the arcs of $\mathcal{A}$ are defined by the points of P. While each half-plane of H defines exactly one point in B and thus $|B| = n$, each point of P may generate as many as $n/2$ (disjoint) arcs in $\mathcal{A}$ and thus $|\mathcal{A}| = O(n^2)$. Our reduction ensures that a point $p \in P$ hits a half-plane $h \in H$ if and only if the point in B defined by h is covered by one of the arcs generated by p. An optimal solution to the hitting set problem on P and H can be easily obtained from an optimal solution to the circular-point coverage problem: Suppose that A_{opt} is a minimum-weight subset of arcs in $\mathcal{A}$ whose union covers B; for each arc in A_{opt}, if the arc is generated by a point $p \in P$, then we add p to P_{opt}. We prove that the subset P_{opt} thus obtained is a minimum-weight hitting set for H (note that this implies that no two arcs from A_{opt} are generated by the same point).

To solve the circular-point coverage problem, while the unweighted case of this problem can be solved in $O((|\mathcal{A}|+|B|)\log(|\mathcal{A}|+|B|))$ time [20], the currently best algorithm for the weighted case, which is due to Atallah, Chen, and Lee [1], runs in $O(\kappa \cdot (|\mathcal{A}|+|B|)+(|\mathcal{A}|+|B|)\log(|\mathcal{A}|+|B|))$ time, where κ is the minimum number of arcs covering any point of B. If applied to our problem, κ is equal to the minimum number of points of P covered by any half-plane of H, and thus $\kappa \leq n$ and applying the algorithm of [1] can solve our problem in $O(\kappa \cdot n^2 + n^2 \log n)$ time, which is $O(n^3)$ in the worst case.

We take a different route to tackle the problem. We further reduce the problem to κ instances of an interval coverage problem. In each instance, we have $O(n^2)$ weighted intervals and $O(n)$ points on a line, and the goal is to compute a minimum-weight subset of intervals whose union covers all points. Clearly, solving each problem instance requires $\Omega(n^2)$ time as the number of intervals could be $\Omega(n^2)$. We instead solve each problem instance in an "indirect" way and the "indirect" solutions of all κ problem instances together also lead to an optimal solution to our original circular-point coverage problem. The advantage of computing an indirect solution is that we do not have to explicitly have the $O(n^2)$ intervals and instead only need to use the original input P and H. Our algorithm can compute an indirect solution for each problem instance in $O(n^{3/2} \log^2 n)$ time. Consequently, we can solve our circular-point coverage problem and thus the hitting set problem in a total of $O(\kappa \cdot n^{3/2} \log^2 n)$ time.

1.3 Outline

The rest of the paper is organized as follows. After defining the notation in Section 2, we reduce the problem to the circular-point coverage problem in Sections 3 and then solve it in Section 4. As discussed above, we solve the circular-point coverage problem by reducing it to κ instances of an interval coverage problem. Our algorithm for the interval coverage problem is also described in Section 4.

2 Preliminaries

We follow the notation in Section 1, e.g., P, H, n, etc. We assume that each half-plane must be hit by a point of P; otherwise, a hitting set does not exist. We can check whether this condition is satisfied in $O(n \log n)$ time (e.g., by first computing the convex hull of P, and then for each half-plane $h \in H$, by using the convex hull we can determine whether h contains a point of P in $O(\log n)$ time). For each point $p \in P$, let $w(p)$ represent the weight of p; we assume $w(p) > 0$ since otherwise p could always be included in the solution.

For each half-plane $h \in H$, its *normal* is the vector perpendicular to its bounding line and toward the interior of h. If two half-planes $h, h' \in H$ have the same normal, then one of them must contain the other, say, $h \subseteq h'$. In this case, h' is redundant because any point hitting h also hits h'. We can efficiently identify redundant half-planes in $O(n \log n)$ time by sorting the half-planes by their normals. In the following, we assume that half-planes of H have distinct normals.

3 Reducing to a circular-point coverage problem

In this section, we reduce the half-plane hitting set problem for P and H to a circular-point coverage problem for a set $\mathcal{A}$ of arcs and a set B of points on a circle C. In what follows, we first define the circular-point coverage problem, i.e., $\mathcal{A}$, B, and C. We then prove the correctness of the reduction, i.e., explain how and why a solution to the circular-point coverage problem can lead to a solution to the half-plane hitting set problem. Our reduction is similar to the approach in [14] for the unweighted case. The difference is that here we have to consider the weights of the points. One may consider our approach an extension of [14].

3.1 Defining the circular-point coverage problem

First, we define C as a unit circle.

Defining $\boldsymbol{B}$. For each half-plane $h \in H$, we define a point b on C as the intersection of C with the ray originating from the center of C and parallel to the normal of h. We call h the *defining half-plane* of b. Let B be the set of all such points on C defined by the half-planes of H. Since no two half-planes in H have the same normal, points in B are distinct on C.

Let $b_1, b_2, \ldots, b_n$ be the points of B ordered counterclockwise on C; we consider it a cyclic list. We use $B[i,j]$ to refer to the (contiguous) sublist of B counterclockwise from b_i to b_j, inclusive, i.e., if $i \leq j$, then $B[i,j] = b_i, b_{i+1}, \ldots, b_j$; otherwise, $B[i,j] = b_i, \ldots, b_n, b_1, \ldots, b_j$. For any two points b and b' on C, let $C[b, b']$ denote the arc of C counterclockwise from b to b', inclusive.

Defining $\boldsymbol{\mathcal{A}}$. For each point $p \in P$, we define a set $A(p)$ of arcs on C. For each maximal sublist $B[i,j]$ of B such that p hits all defining half-planes of the points in $B[i,j]$, we add the arc $C[b_i, b_j]$ to $A(p)$. In the special case where p hits all the half-planes of H, we let $A(p)$ consist of the single arc $C[b_1, b_n]$. Note that $A(p)$ may contain at most $\lfloor n/2 \rfloor$ arcs, and all these arcs are pairwise disjoint. We say that p is the *defining point* of the arcs of $A(p)$. For each arc $\alpha \in A(p)$, we let $w(\alpha) = w(p)$ denote its weight. Define $\mathcal{A} = \bigcup_{p \in P} A(p)$. Clearly, $|\mathcal{A}| = O(n^2)$.

3.2 Correctness of the reduction

Consider the circular-point coverage problem for B and $\mathcal{A}$. Suppose A_{opt} is an optimal solution, i.e., A_{opt} is a minimum-weight subset of $\mathcal{A}$ for covering B. We create a subset P_{opt} of P as follows: For each arc in A_{opt}, we add its defining point to P_{opt}.

We next prove that P_{opt} is an optimal solution to the half-plane hitting set problem for P and H. At first glance, one potential issue is that two arcs of A_{opt} might be defined by the same point of P and thus a point might be added to P_{opt} multiple times. We will show that this is not possible, implying that $\sum_{p \in P_{\text{opt}}} w(p) = \sum_{\alpha \in A_{\text{opt}}} w(\alpha)$. These are proved in Corollary 2, which follows mostly from the following lemma (see the full version for the proof).

Lemma 1 *If there exists a subset of arcs of $\mathcal{A}$ with total weight W that forms a coverage for B, then there exists a subset of P with total weight at most W that forms a hitting set for H. Symmetrically, if there exists a subset of P with total weight W that forms a hitting set for H, then there exists a subset of arcs of $\mathcal{A}$ with total weight at most W that forms a coverage for B.*

Corollary 2
1. *The total weight of a minimum-weight subset of $\mathcal{A}$ for covering B is equal to the total weight of a minimum-weight subset of P for hitting H.*
2. *No two arcs of A_{opt} are defined by the same point.*
3. *P_{opt} is an optimal solution to the half-plane hitting set problem.*

Proof. The first corollary statement directly follows from Lemma 1.

For the second corollary statement, notice that P_{opt} is a hitting set of H, which follows a similar argument to the first direction of Lemma 1. Assume to the contrary that A_{opt} has two arcs α and α' defined by the same point of P. Then, since the weight of every arc is positive, by the definition of P_{opt}, $\sum_{p \in P_{\text{opt}}} w(p) < \sum_{\alpha \in A_{\text{opt}}} w(\alpha)$ holds. As A_{opt} is a minimum-weight subset of $\mathcal{A}$ for covering B and P_{opt} is a hitting set of H, we obtain that the total weight of a hitting set for H is smaller than the total weight of a minimum-weight subset of $\mathcal{A}$ for covering B, a contradiction to the first corollary statement.

For the third statement, due to the second statement, by the definition of P_{opt}, it holds that $\sum_{p \in P_{\text{opt}}} w(p) = \sum_{\alpha \in A_{\text{opt}}} w(\alpha)$. As A_{opt} is a minimum-weight subset of $\mathcal{A}$ for covering B, and P_{opt} is a hitting set for H, by the first corollary statement, P_{opt} must be a minimum-weight hitting set for H. □

By Corollary 2, we have successfully reduced our half-plane hitting set problem on H and P to the circular-point coverage problem on B and $\mathcal{A}$.

4 Solving the circular-point coverage problem

In this section, we present an algorithm to solve the circular-point coverage problem in Section 3, i.e., computing a subset of arcs of $\mathcal{A}$ such that their union covers all points of B and their total weight is minimized (such a subset is called an *optimal solution* of the problem; we often use A_{opt} to denote an optimal solution). Recall $|\mathcal{A}| = O(n^2)$ and $|B| = n$.

4.1 Reducing to interval coverage

Our strategy is to reduce the problem to multiple instances of an interval coverage problem and then solve

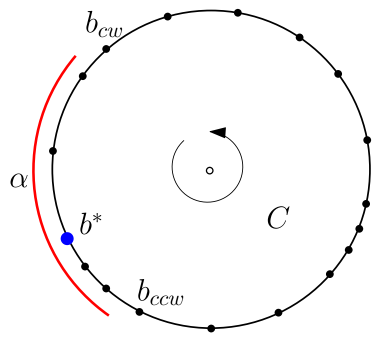


Figure 1: Illustration of the definitions of b_{ccw} and b_{cw}.

all instances, whose solutions together will lead to a solution to the circular problem.

We start by finding a point $b^* \in B$ that is covered by the fewest arcs in $\mathcal{A}$. This can be easily done in $O(n^2 \log n)$ time by sorting the endpoints of the arcs of $\mathcal{A}$ along with the points of B. We have the following easy observation.

Observation 1 *For any optimal solution A_{opt} to the circular-point coverage problem, b^* must be covered by an arc $\alpha \in A_{opt}$ and A_{opt} does not have any other arc that fully contains α.*

Proof. As A_{opt} is an optimal solution, all points of B are covered by the union of the arcs of A_{opt}. Therefore, b^* must be covered by an arc $\alpha \in A_{\text{opt}}$. Assume to the contrary that A_{opt} has another arc α' that fully contains α. Then, if we remove α from A_{opt}, the remaining arcs of A_{opt} still form a coverage of B. But this contradicts the optimality of A_{opt}. $\square$

For each arc $\alpha \in \mathcal{A}$ that covers b^*, define B_α as the subset of points of B that are not covered by α, and define $\mathcal{A}_\alpha$ as the subset of arcs of $\mathcal{A} \setminus \{\alpha\}$ that do not fully contain α and are not defined by the point $p(\alpha)$, where $p(\alpha)$ is the point of P that defines α.

We consider the following *interval coverage problem for α*: Find a subset of arcs of $\mathcal{A}_\alpha$ such that their union covers B_α and the total weight of the arcs is minimized. This is a 1D interval coverage problem (not a circular one). Indeed, let b_{ccw} (resp., b_{cw}) be the first point of B counterclockwise from b^* not covered by α (see Figure 1). By definition, no arc of $\mathcal{A}_\alpha$ fully contains α. Hence, no arc of $\mathcal{A}_\alpha$ contains all three points b_{cw}, b^*, and b_{ccw}. This means that we can break the circle C at b^* into an arc, denoted by C_{b^*}, such that every arc of $\mathcal{A}_\alpha$ is contained in C_{b^*}. Furthermore, we can map C_{b^*} to the x-axis ℓ from left to right containing all points of B_α from b_{ccw} counterclockwise to b_{cw}, so that each arc of $\mathcal{A}_\alpha$ becomes an interval of ℓ. Therefore, the problem is essentially a 1D interval coverage problem. Let A^*_α denote an optimal solution of the problem.

For any subset $\mathcal{A}'$ of arcs of $\mathcal{A}$, we use $W(\mathcal{A}')$ denote the total weight of all arcs of $\mathcal{A}'$.

By slightly abusing the notation, let $\mathcal{A}_{b^*}$ denote the subset of arcs of $\mathcal{A}$ that contain b^*. The following lemma explains why we are interested in the above interval coverage problem.

Lemma 3 *Suppose that $\alpha' = \operatorname{argmin}_{\alpha \in \mathcal{A}_{b^*}}(w(\alpha) + W(A^*_\alpha))$, i.e., among all arcs $\alpha \in \mathcal{A}_{b^*}$, α' is the one with the minimum $w(\alpha) + W(A^*_\alpha)$. Then, $\{\alpha'\} \cup A^*_{\alpha'}$ is an optimal solution to the circular-point coverage problem for $\mathcal{A}$ and B.*

Proof. Let A_{opt} be an optimal solution to the circular-point coverage problem for $\mathcal{A}$ and B. By definition, the arcs of $\{\alpha'\} \cup A^*_{\alpha'}$ form a coverage of B. Since A_{opt} is an optimal solution, we have $W(A_{\text{opt}}) \leq W(\{\alpha'\} \cup A^*_{\alpha'}) = w(\alpha') + W(A^*_{\alpha'})$. In the following, we show that $w(\alpha') + W(A^*_{\alpha'}) \leq W(A_{\text{opt}})$, which will prove the lemma.

By Observation 1, b^* is covered by an arc $\alpha \in A_{\text{opt}}$ and A_{opt} does not have another arc that fully contains α. By Corollary 2, no two arcs of A_{opt} are defined by the same point of P. Consequently, by the definition of $\mathcal{A}_\alpha$, we have $A_{\text{opt}} \setminus \{\alpha\} \subseteq \mathcal{A}_\alpha$. Since α does not cover any point of B_α by definition, $A_{\text{opt}} \setminus \{\alpha\}$ must form a coverage of B_α, and thus $W(A^*_\alpha) \leq W(A_{\text{opt}} \setminus \{\alpha\})$. As such, by the definition of α', we obtain $w(\alpha') + W(A^*_{\alpha'}) \leq w(\alpha) + W(A^*_\alpha) \leq w(\alpha) + W(A_{\text{opt}} \setminus \{\alpha\}) = W(A_{\text{opt}})$. $\square$

With Lemma 3, it is attempting to solve the interval coverage problem for all arcs $\alpha \in \mathcal{A}_{b^*}$. However, this would not be efficient because solving the problem for each arc α takes $\Omega(n^2)$ time due to $|\mathcal{A}_\alpha| = \Omega(n^2)$ in the worst case (and thus the overall time complexity would be $\Omega(n^3)$ in the worst case). Instead, we take an "indirect" strategy as follows.

An indirect route. Let H_α denote the set of half-planes of H that define the points of B_α. Let $P_\alpha = P \setminus \{p(\alpha)\}$, where $p(\alpha)$ is the point of P that defines α. Given the arc α, B_α and P_α can be easily computed in $O(n)$ time. Clearly, $|H_\alpha| = |B_\alpha| \leq n-1$ and $|P_\alpha| = n-1$.

For each arc $\alpha \in \mathcal{A}_{b^*}$, our strategy is to design an efficient algorithm to compute a subset $P'_\alpha \subseteq P_\alpha$ and a value W_α such that P'_α is a hitting set for H_α and $W(P'_\alpha) \leq W_\alpha \leq W(A^*_\alpha)$. We refer to (P'_α, W_α) as an "indirect solution" to the interval coverage problem for α. The following lemma explains why such indirect solutions are sufficient for our purpose.

Lemma 4 *Suppose that $\alpha^* = \operatorname{argmin}_{\alpha \in \mathcal{A}_{b^*}}(w(\alpha) + W_\alpha)$. Then $\{p(\alpha^*)\} \cup P'_{\alpha^*}$ is an optimal solution to the half-plane hitting set problem for P and H, where $p(\alpha^*)$ is the point of P that defines α^*.*

Proof. Let P_{opt} be an optimal solution to the half-plane hitting set problem, and A_{opt} an optimal solution to the circular-point coverage problem for $\mathcal{A}$ and B. By Corollary 2, $W(P_{\text{opt}}) = W(A_{\text{opt}})$.

Since all points of $B \setminus B_{\alpha^*}$ are covered by the arc α^*, by definition, all half-planes of $H \setminus H_{\alpha^*}$ are hit by $p(\alpha^*)$. Since each half-plane of H_{α^*} is hit by a point of P'_{α^*}, we obtain that $\{p(\alpha^*)\} \cup P'_{\alpha^*}$ is a hitting set for H. As such, it holds that $W(P_{\text{opt}}) \le w(p(\alpha^*)) + W(P'_{\alpha^*}) = w(\alpha^*) + W(P'_{\alpha^*}) \le w(\alpha^*) + W_{\alpha^*}$. In the following, we argue that $w(\alpha^*) + W_{\alpha^*} \le W(P_{\text{opt}})$, which will lead to $W(P_{\text{opt}}) = w(p(\alpha^*)) + W(P'_{\alpha^*})$ and thus prove the lemma.

Suppose that $\alpha' = \operatorname{argmin}_{\alpha \in \mathcal{A}_{b^*}} (w(\alpha) + W(A^*_\alpha))$. By Lemma 3, $W(A_{\text{opt}}) = w(\alpha') + W(A^*_{\alpha'})$. By the definition of α^*, $w(\alpha^*) + W_{\alpha^*} \le w(\alpha') + W_{\alpha'}$. Since $W_{\alpha'} \le W(A^*_{\alpha'})$, we obtain $w(\alpha^*) + W_{\alpha^*} \le w(\alpha') + W(A^*_{\alpha'}) = W(A_{\text{opt}}) = W(P_{\text{opt}})$. □

4.2 Computing the interval coverage indirection solutions

We now present our algorithm to compute an indirect solution (P'_α, W_α) for the interval coverage problem for each α; the runtime of the algorithm is $O(n^{3/2} \log^2 n)$. In the following, we first describe the algorithm, then prove its correctness, and finally discuss how to implement it efficiently.

4.2.1 Algorithm description

Recall that $|H_\alpha| = |B_\alpha| \le n-1$ and $|P_\alpha| = n-1$. For notational convenience, we simple let $|H_\alpha| = |B_\alpha| = |P_\alpha| = n$.

We first sort the points of B_α counterclockwise along C_α as $b_1, b_2, \ldots, b_n$. For each b_i, let h_i denote its defining half-plane in H_α. For each half-plane h, denote by $P_\alpha(h)$ the subset of points of P_α inside h, and $P_\alpha(\overline{h})$ the subset of points of P_α outside h.

Our algorithm processes the half-planes of H_α in their index order. For each half-plane $h_i \in H_\alpha$, the algorithm computes a value δ_i. The algorithm also maintains a value *cost*(p) for each point $p \in P_\alpha$, which is initialized to its weight $w(p)$. The pseudocode of the algorithm is given in Algorithm 1.

The algorithm is essentially a dynamic program. We will prove later that the value W_α returned by the algorithm, which is δ_n, is smaller than or equal to $W(A^*_\alpha)$. According to the algorithm, δ_n is equal to the total weight of a subset of points of P_α; we let P'_α be the subset. To find P'_α, we can slightly modify the algorithm following the standard dynamic programming backtracking technique. Specifically, if δ_n is equal to *cost*(p) for some point $p \in P_\alpha(h_n)$, then p hits h_n and we add p to P'_α. Suppose that *cost*(p) is equal to $w(p) + \delta_i$ for some index i. Then, *cost*(p) does not change in the

Algorithm 1:

```
Input: $P_\alpha$ and $H_\alpha = \{h_1, h_2, \ldots, h_n\}$
Output: An indirect solution $(P'_\alpha, W_\alpha)$
1 $cost(p) \leftarrow w(p)$, for all points $p \in P_\alpha$;
2 for $i \leftarrow 1$ to $n$ do
3     $\delta_i \leftarrow \min_{p \in P_\alpha(h_i)} cost(p)$;    // FindMinCost Operation
4     $cost(p) \leftarrow w(p) + \delta_i$ for all points $p \in P_\alpha(\overline{h_i})$;
        // ResetCost Operation
5 end
6 return $\delta_n$ as $W_\alpha$;    // $P'_\alpha$ consists of the points $p$ whose weights $w(p)$ are included in $\delta_n$ and $P'_{opt}$ can be found by the standard backtracking technique
```

j-th iteration for any $i < j < n-1$, meaning that $p \in P_\alpha(h_j)$. Hence, p hits all half-planes $h_{i+1}, \ldots, h_n$. Further, by definition δ_i is equal to *cost*(p') for some point $p' \in P_\alpha(h_i)$; we add p' to P'_α. We continue this backtracking process until a point whose cost is equal to its own weight is added to P'_α (in which case following the above analysis P'_α is a hitting set for H_α). Note that in the above process if a point p is added to P'_α multiple times, then $w(p)$ is included in δ_n for each copy of p. As such, since $W(P'_\alpha)$ represents the total weight of all distinct points of P'_α, it holds that $W(P'_\alpha) \le \delta_n = W_\alpha$. To prove the correctness of the algorithm, it now remains to prove that $W_\alpha \le W(A^*_\alpha)$, which will be done in Section 4.2.2.

For reference purposes, we use FindMinCost to refer to the operation in Line 3 and use ResetCost to refer to the operation in Line 4 of Algorithm 1. The efficiency of the algorithm hinges on how to implement these two *key operations*, which will be discussed later in Section 4.2.3.

4.2.2 Algorithm correctness: $W_\alpha \le W(A^*_\alpha)$

As discussed above, it remains to prove that $W_\alpha \le W(A^*_\alpha)$. To this end, we use an "algorithmic proof": We first show that an optimal solution A^*_α can be found by a dynamic programming algorithm and then use the algorithm to argue that $W_\alpha \le W(A^*_\alpha)$.

A dynamic programming algorithm. With B_α and $\mathcal{A}_\alpha$, computing A^*_α can be done by a straightforward dynamic programming algorithm [18]. To prove $W_\alpha \le W(A^*_\alpha)$, we first describe the algorithm.

Recall that the points of B_α are ordered along C_α as $b_1, b_2, \ldots, b_n$. As discussed in Section 1, we can map them on the x-axis ℓ so that b_1 (resp., b_n) is the leftmost (resp., rightmost) point of B_α and each arc of $\mathcal{A}_\alpha$ becomes an interval on ℓ. For convenience, we use b_0 to refer to a point of ℓ to the left of b_1 so that no arc of

$\mathcal{A}_\alpha$ contains it.

For each arc $\beta \in \mathcal{A}_\alpha$, define I_β as the index of the rightmost point of $B_\alpha \cup \{b_0\}$ strictly to the left of the left endpoint of β. With the point b_0, I_β is well defined. For $1 \le i \le j \le n$, define $B_\alpha[i,j] = \{b_i, b_{i+1}, \dots, b_j\}$ and $H_\alpha[i,j] = \{h_i, h_{i+1}, \dots, h_j\}$.

For each $i \in [1,n]$, define δ_i^* as the minimum total weight of a subset of arcs of $\mathcal{A}_\alpha$ whose union covers all points of $B_\alpha[1,i]$. The goal is to compute δ_n^*, which is equal to $W(A_\alpha^*)$. For convenience, we let $\delta_0^* = 0$. For each arc $\beta \in \mathcal{A}_\alpha$, define $cost(\beta) = w(\beta) + \delta_{I_\beta}^*$. One can verify that the following holds: $\delta_i^* = \min_{\beta \in \mathcal{A}_\alpha(b_i)} cost(\beta)$, where $\mathcal{A}_\alpha(b_i)$ is the subset of arcs of $\mathcal{A}_\alpha$ that cover b_i. This is the recursive relation of the dynamic program.

We sweep a point q on ℓ from left to right. During the sweep, the subset $\mathcal{A}_\alpha(q) \subseteq \mathcal{A}_\alpha$ of arcs that cover q is maintained. We assume that the cost of each arc of $\mathcal{A}_\alpha(q)$ is already known and the values δ_i^* for all points $b_i \in B_\alpha$ to the left of q have been computed. An event happens when q encounters an endpoint of an arc of $\mathcal{A}_\alpha$ or a point of B_α. If q encounters a point $b_i \in B$, then we find the arc of $\mathcal{A}_\alpha(q)$ with the minimum cost and assign the cost value to δ_i^*. If q encounters the left endpoint of an arc β, we set $cost(\beta) = w(\beta) + \delta_{I_\beta}^*$ (assuming that the index I_β is already knonw) and insert β into $\mathcal{A}_\alpha(q)$. If q encounters the right endpoint of an arc β, we remove β from $\mathcal{A}_\alpha(q)$. The algorithm finishes once q encounters b_n, at which event δ_n^* is computed.

Proving $W_\alpha \le W(A_\alpha^*)$. To prove $W_\alpha \le W(A_\alpha^*)$, since $W_\alpha = \delta_n$ and $W(A_\alpha^*) = \delta_n^*$, it suffices to show that $\delta_n \le \delta_n^*$. In the following, we argue that $\delta_i \le \delta_i^*$ holds for all $1 \le i \le n$. We do so by induction.

As the base case, we first argue $\delta_1 \le \delta_1^*$. By definition, $\delta_1 = \min_{p \in P_\alpha(h_1)} w(p)$. For δ_1^*, since $I_\beta = 0$ for every arc $\beta \in \mathcal{A}_\alpha(b_1)$ and $\delta_0^* = 0$, we have $\delta_1^* = \min_{\beta \in \mathcal{A}_\alpha(b_1)} w(\beta)$. By definition, an arc $\beta \in \mathcal{A}_\alpha$ covers b_1 only if the point $p(\beta)$ of P_α defining β hits h_1, and $w(p(\beta)) = w(\beta)$. Therefore, β is in $\mathcal{A}_\alpha(b_1)$ only if $p(\beta)$ is in $P_\alpha(h_1)$. This implies that $\delta_1 \le \delta_1^*$.

Consider any i with $2 \le i \le n$. Assuming that $\delta_j \le \delta_j^*$ for all $1 \le j < i$, we now prove $\delta_i \le \delta_i^*$. Recall that $\delta_i = \min_{p \in P_\alpha(h_i)} cost(p)$ and $\delta_i^* = \min_{\beta \in \mathcal{A}_\alpha(b_i)} cost(\beta)$. As argued above, each arc $\beta \in \mathcal{A}_\alpha(b_i)$ is defined by a point in $P_\alpha(h_i)$ with the same weight.

Consider an arc $\beta \in \mathcal{A}_\alpha(b_i)$. Let p be the point of $P_\alpha(h_i)$ that defines β. To prove $\delta_i \le \delta_i^*$, it suffices to show that $cost(p) \le cost(\beta)$. By definition, $cost(\beta) = w(\beta) + \delta_{I_\beta}^*$. For notational convenience, let $j = I_\beta$. By definition, all points of $B_\beta[j+1,i]$ are covered by β but b_j is not. Therefore, all half-planes of $H_\alpha[j+1,i]$ are hit by p but h_j is not. As such, during the ResetCost operation of the j-th iteration of Algorithm 1, $cost(p)$ will be set to $w(p) + \delta_j$; furthermore, $cost(p)$ will not be reset again during the i'-th iteration for all $j+1 \le i' \le i$. Hence, we have $cost(p) = w(p) + \delta_j$ at the beginning of the i-th iteration of the algorithm. Since $\delta_j \le \delta_j^*$ holds by the induction hypothesis and $w(p) = w(\beta)$, we obtain $cost(p) \le cost(\beta)$. This proves $\delta_i \le \delta_i^*$.

This proves $W_\alpha \le W(A_\alpha^*)$. The correctness of Algorithm 1 is thus established.

4.2.3 Algorithm implementation

We are able to implement the two key operations FindMinCost and ResetCost to achieve an overall time complexity of $O(n^{3/2} \log^2 n)$ for the entire Algorithm 1. The implementation is similar to that for an algorithm of Liu and Wang [12] for a line separable unit-disk coverage problem. More specifically, given a set of points and a set of weighted unit disks such that the points are separated from the disk centers by the x-axis ℓ, their problem is to compute a minimum-weight subset of disks whose union covers all points. As such, to cover points, their algorithm uses unit disks (whose centers are all below ℓ) while ours uses half-planes. We can essentially use the same way as theirs to implement our algorithm with the following changes. Their algorithm initially constructs a cutting [4, 19] on the arcs of the boundaries of the disks above the line ℓ. Instead, we build a cutting on the bounding lines of the half-planes of H_α. For the rest of the algorithm, we simply replace their disks with our half-planes. The analysis and time complexities are very similar. We sketch the main idea in the full version for completeness.

Lemma 5 *An indirect solution (P'_α, W_α) for the interval coverage problem for each $\alpha \in \mathcal{A}_{b^*}$ can be computed in $O(n^{3/2} \log^2 n)$ time.*

4.3 Putting it all together

Combining Lemmas 4 and 5, the hitting set problem on P and H can be solved in $O(\kappa \cdot n^{3/2} \log^2 n)$ time, where $\kappa = |\mathcal{A}_{b^*}| \le n$. Equivalently, κ is the minimum number of points of P covered by any half-plane of H. We thus obtain the following result.

Theorem 6 *Given a set P of n weighted points and a set H of n half-planes in the plane, one can compute in $O(n^{5/2} \log^2 n)$ time a minimum-weight subset of P as a hitting set for H. More specifically, the runtime is $O(\kappa \cdot n^{3/2} \log^2 n)$, where κ is the minimum number of points of P covered by any half-plane of H.*

References

[1] Mikhail J. Atallah, Danny Z. Chen, and D. T. Lee. An optimal algorithm for shortest paths on weighted interval and circular-arc graphs, with applications. *Algorithmica*, 14:429–441, 1995. doi:10.1007/BF01192049.

[2] Norbert Bus, Nabil H. Mustafa, and Saurabh Ray. Practical and efficient algorithms for the geometric hitting set problem. *Discrete Applied Mathematics*, 240:25–32, 2018. doi:10.1016/j.dam.2017.12.018.

[3] Timothy M. Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Computational Geometry: Theory and Applications*, 47:112–124, 2014. doi:10.1016/j.comgeo.2012.04.001.

[4] Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993. doi:10.1007/BF02189314.

[5] Stephane Durocher and Robert Fraser. Duality for geometric set cover and geometric hitting set problems on pseudodisks. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG)*, 2015. https://research.cs.queensu.ca/cccg2015/CCCG15-papers/10.pdf.

[6] Guy Even, Dror Rawitz, and Shimon Shahar. Hitting sets when the VC-dimension is small. *Information Processing Letters*, 95:358–362, 2005. doi:10.1016/j.ipl.2005.03.010.

[7] Shashidhara K. Ganjugunte. *Geometric hitting sets and their variants*. PhD thesis, Duke University, 2011. https://dukespace.lib.duke.edu/items/391e2278-74f0-408f-9be7-4c97cf72e352.

[8] Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *Journal of Computational Geometry*, 3:65–85, 2012. doi:10.20382/jocg.v3i1a4.

[9] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

[10] Jian Li and Yifei Jin. A PTAS for the weighted unit disk cover problem. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 898–909, 2015. doi:10.1007/978-3-662-47672-7_73.

[11] Gang Liu and Haitao Wang. Geometric hitting set for line-constrained disks. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS)*, pages 574–587, 2023. doi:10.1007/978-3-031-38906-1_38.

[12] Gang Liu and Haitao Wang. On line-separable weighted unit-disk coverage and related problems. In *Proceedings of the 49th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 70:1–70:16, 2024. doi:10.4230/LIPIcs.MFCS.2024.70.

[13] Gang Liu and Haitao Wang. Unweighted geometric hitting set for line-constrained disks and related problems. In *Proceedings of the 49th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 68:1–68:15, 2024. doi:10.4230/LIPIcs.MFCS.2024.68.

[14] Gang Liu and Haitao Wang. An optimal algorithm for half-plane hitting set. In *Proceedings of the Symposium on Simplicity in Algorithms (SOSA)*, pages 509–516, 2025. doi:10.1137/1.9781611978315.38.

[15] El O. Mourad, Fohlin Helena, and Srivastav Anand. A randomised approximation algorithm for the hitting set problem. *Theoretical Computer Science*, 555:23–34, 2014. doi:10.1007/978-3-642-36065-7_11.

[16] Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44:883–895, 2010. doi:10.1007/s00454-010-9285-9.

[17] Logan Pedersen and Haitao Wang. On the coverage of points in the plane by disks centered at a line. In *Proceedings of the 30th Canadian Conference on Computational Geometry (CCCG)*, pages 158–164, 2018. https://home.cs.umanitoba.ca/~cccg2018/papers/session4A-p1.pdf.

[18] Logan Pedersen and Haitao Wang. Algorithms for the line-constrained disk coverage and related problems. *Computational Geometry: Theory and Applications*, 105-106:101883:1–18, 2022. doi:10.1016/j.comgeo.2022.101883.

[19] Haitao Wang. Unit-disk range searching and applications. *Journal of Computational Geometry*, 14:343–394, 2023. doi:10.20382/jocg.v14i1a13.

[20] Haitao Wang and Jie Xue. Algorithms for halfplane coverage and related problems. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, pages 79:1–79:15, 2024. doi:10.4230/LIPIcs.SoCG.2024.79.

Covering radii of 3-zonotopes and the shifted lonely runner conjecture

David Alcántara[*] Francisco Criado[†] Francisco Santos[‡]

Abstract

We show that the shifted Lonely Runner Conjecture (sLRC) holds for 5 runners. We also determine that there are exactly 3 primitive tight instances of the conjecture, only two of which are tight for the non-shifted conjecture (LRC). Our proof is computational, relying on a rephrasing of the sLRC in terms of covering radii of certain zonotopes (Henze and Malikiosis, 2017), and on an upper bound for the (integer) velocities to be checked (Malikiosis, Santos and Schymura, 2024+).

As a tool for the proof, we devise an algorithm for bounding the covering radius of rational lattice polytopes, based on constructing dyadic fundamental domains.

This is an extended abstract for the preprint [1].

1 Introduction

The lonely runner conjecture (LRC) states that if $n+1$ runners run along a circle of length one with constant, distinct, velocities, all starting at the origin, then for every runner there is a time at which all other runners are at distance at least $1/(n+1)$ from it. It was posed in 1968 by J. Wills [14] in the language of diophantine approximation, and is currently proved up to $n = 6$ [2]. The conjecture has attracted quite some attention due to the simplicity of its statement and because it admits various interpretations, from its original diophantine approximation statement, to visibility obstruction, billiard trajectories or nowhere zero flows in graphs, among others. See [12] for a very recent survey. We are interested in the so-called *shifted* version, a generalization in which runners are allowed to have different starting points. This version appeared in print for the first time in 2019 [3].

In both the original and the shifted versions, the runner we are looking at can be fixed at the origin, since only relative velocities are important. Hence the shifted conjecture becomes the following (the original LRC is the special case where $s_i = 0$ for all i):

Conjecture 1 (sLRC) *Let $v_1, \dots, v_n,\ s_1, \dots, s_n \in \mathbb{R}$ be real numbers, with the v_i distinct and non-zero. Then, there is a $t \in \mathbb{R}$ such that for every $i \in [n] := \mathbb{Z}/n\mathbb{Z}$, the distance between $v_i t + s_i$ and the closest integer, $\operatorname{dist}(v_i t + s_i, \mathbb{Z}) \geq \frac{1}{n+1}$.*

Since the order of the runners is not relevant, we assume without loss of generality that $v_1 < \cdots < v_n$.

This shifted version of the Lonely Runner Conjecture is only currently proved up to $n = 3$ ("four runners") [5, 13].[1] For our proof we use that in Conjecture 1 (and in the original LR conjecture) there is no loss of generality in assuming all velocities v_i to be positive integers [5,6, Section 4.1]. We then rely on the following result of Malikiosis, Santos and Schymura:

Theorem 1 ([11, Corollary 1.11]) *sLRC holds for $n = 4$ for all integer velocities with sum at least 196.*

That is, only the velocity vectors $(v_1, v_2, v_3, v_4) \in \mathbb{Z}$ with $1 \leq v_1 < v_2 < v_3 < v_4$ and $v_1 + v_2 + v_3 + v_4 \leq 195$ need to be checked. We can also assume $\gcd(v_1, v_2, v_3, v_4) = 1$ since dividing all velocities by a common factor c does not change the problem: the positions at time t of the original problem coincide with the positions at time ct of the new one. With these considerations our main result is:

Theorem 2 *There are 2 133 561 velocity vectors $(v_1, v_2, v_3, v_4) \in \mathbb{Z}$ with $1 \leq v_1 < v_2 < v_3 < v_4$, $v_1 + v_2 + v_3 + v_4 \leq 195$ and $\gcd(v_1, v_2, v_3, v_4) = 1$. The sLRC holds for all of them.*

Corollary 3 *sLRC (Conjecture 1) holds for $n = 4$ (five runners).*

We also show there are exactly 3 primitive integer velocity vectors that are *tight*, meaning that for them the bound $\frac{1}{5}$ is the best possible.

Theorem 4 *The only integer velocity vectors $(v_1, v_2, v_3, v_4) \in \mathbb{Z}^4$ with $1 \leq v_1 < v_2 < v_3 < v_4$, and $\gcd(v_1, v_2, v_3, v_4) = 1$ for which there are starting points $s_1, \dots, s_n \in \mathbb{R}$ such that for every time $t \in \mathbb{R}$, there is an index $i \in [n]$ such that $\operatorname{dist}(v_i t + s_i, \mathbb{Z}) \leq \frac{1}{5}$, are $(1,2,3,4)$, $(1,3,4,6)$, and $(1,3,4,7)$.*

[*]Departmento de Matemáticas, Estadística y Computación, Universidad de Cantabria, Santander, Spain, david.alcantara@unican.es. Funded by grant PRE2022-000020 of MCIN/AEI/10.13039/501100011033.

[†]Departamento de Matemáticas, CUNEF Universidad, Madrid, Spain, francisco.criado@cunef.edu

[‡]Departmento de Matemáticas, Estadística y Computación, Universidad de Cantabria, Santander, Spain, francisco.santos@unican.es

[1]Observe that Rifford [13] refers to this as the case $n = 4$

Clearly, being tight for the nonshifted version implies tight for the shifted one. As an example, it is easy to show, as was already observed by Wills [15], that the vector $(1, \ldots, n)$ is tight for the nonshifted version, for all $n \in \mathbb{N}$: suppose that at a certain time t we have no $v_i t$ in $[-1/(n+1), 1/(n+1)] + \mathbb{Z}$. Then the pigeon-hole principle implies that at that time at least two runners, say i and i' must be in the same interval $[j/(t+1), (j+1)/(t+1)] + \mathbb{Z}$ for some $j \in [n-1]$. But then, assuming w.l.o.g. that $i' > i$, we have the contradiction that[2]

$$v_i t, v_{i'} t \in \left[\tfrac{j}{n+1}, \tfrac{j+1}{n+1}\right] + \mathbb{Z} \qquad \Rightarrow$$
$$v_{i'-i} t = v_{i'} t - v_i t \in \left[-\tfrac{1}{n+1}, \tfrac{1}{n+1}\right] + \mathbb{Z},$$

The vectors $(1,2,3,4)$ and $(1,3,4,7)$ are known to be tight for the non-shifted LRC and, in fact, they are the only ones for $n = 4$ [7]. The vector $(1,3,4,6)$ is new and shows that shifted tightness does not imply the unshifted one.

Our method (as well as the proof of Theorem 1 in [11]) is based on the relation between the Lonely Runner conjecture (both shifted and original one) to $(n-1)$-dimensional zonotopes with n generators [3, 8]. In particular, the sLRC can be restated as a bound on the covering radius of a certain class of zonotopes in $\mathbb{R}^{n-1}$.

Our proofs of Theorems 2 and 4 are computational; for each primitive velocity vector we build a fundamental domain of the integer lattice $\mathbb{Z}^3$ that fits in the dilated zonotope associated to it. This certifies that its covering radius satisfies the bound.

To prove tightness of the instances of Theorem 4, we explicitly find the *last covered points* of the zonotopes. These points correspond to the starting points of the sLRC which are tight for those velocity vectors.

Our algorithm to construct fundamental domains can in fact decide the covering radius of arbitrary lattice polytopes in any dimension.

2 Zonotopal statement of the LRC

We here recall the reformulation of Conjecture 1 in terms of zonotopal geometry, derived in [3, 8, 11].

A *zonotope* is any Minkowski sum of line segments. As such, any zonotope Z can be written as

$$\mathbf{c} + \sum_{i=1}^{n} [\mathbf{0}, \mathbf{u}_i] = \left\{ \mathbf{c} + \sum_{i=1}^{n} \lambda_i \mathbf{u}_i : \lambda_i \in [0,1] \ \forall i \right\},$$

for a certain finite set $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{R}^d$ of vectors, called the *generators* of Z, and a certain point $\mathbf{c}$. This point is not important for us, since all that we do is invariant under translation. One natural choice is $\mathbf{c} = \mathbf{0}$ but often a more convenient choice is $\mathbf{c} = \frac{1}{2}\sum_{i=1}^{n} \mathbf{u}_i$, since it makes the zonotope become $Z = \frac{1}{2}\sum_{i=1}^{n}[-\mathbf{u}_i, \mathbf{u}_i]$, and be centrally symmetric around the origin.

[2]This argument is an instance of the general proof of Dirichlet's approximation theorem.

2.1 Lonely runner zonotopes and volume vectors

Definition 1 *A* Lonely Runner (LR) Zonotope *is any zonotope $Z \subset \mathbb{R}^{n-1}$ generated by a set of n integer vectors $\mathbf{U} = \{\mathbf{u}_i : 1 \le i \le n\} \subset \mathbb{Z}^{n-1}$ in linear general position; that is, such that every $n-1$ of them are a linear basis of $\mathbb{R}^{n-1}$.*

The volume vector *of Z is the vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{Z}_{>0}$ defined by*

$$v_i := |\det(\mathbf{U} \setminus \{\mathbf{u}_i\})|. \tag{1}$$

When all entries of the volume vector are distinct we say that Z is a strong Lonely Runner (sLR) Zonotope.

We call $\mathbf{v}$ the *volume vector* of Z, because its entries are the volumes of the n parallelepipeds that make up Z. In particular we have that $\mathrm{vol}(Z) = \sum_{i=1}^{n} v_i$ (see details, e.g., in [11]). Observe also that the generators and the volume vector satisfy

$$v_1\mathbf{u}_1 \pm \cdots \pm v_n\mathbf{u}_n = 0$$

for some choice of signs. In fact, this equation (together with positivity of the v_i) characterizes $\mathbf{v}$ for given generators, modulo a scalar factor.

In the following result and the rest of the paper, a unimodular transformation is an affine transformation with integer coefficients and determinant ± 1.

Proposition 5 (Prop. 2.2, [1], see also §1.2, [11]) *For every integer $\mathbf{v} = (v_1, \ldots, v_n) \in {\mathbb{Z}_{>0}}^n$ there is some LR zonotope with integer generators and with volume vector $\mathbf{v}$. If $\gcd(v_1, \ldots, v_n) = 1$, then any two such zonotopes are equivalent by a unimodular transformation.*

2.2 Covering radius and the sLRC

A *convex body* in $\mathbb{R}^d$ is a convex compact subset. We assume our convex bodies to be *nondegenerate*, that is, that they have non-empty interior or, equivalently, that they are not contained in a hyperplane. This includes all bounded full-dimensional polytopes.

Definition 2 (Covering radius) *Let $C \subseteq \mathbb{R}^d$ be a convex body. The* covering radius of C, *denoted $\mu(C)$, is the smallest dilation factor $\rho > 0$ such that*

$$\rho C + \mathbb{Z}^d = \mathbb{R}^d.$$

The covering radius is invariant under real translations and unimodular transformations of C since they amount to translations and unimodular transformations of $\rho C + \mathbb{Z}^d$.

The zonotopal restatement of the *Shifted Lonely Runner Conjecture* is the following. Our statement is taken from [11] but the result is implicit in [3,5,8].

Proposition 6 ([8], see also [11, Proposition 1.8]) *Let $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{Z}^n_{>0}$ with pairwise distinct entries and $\gcd(v_1, \dots, v_n) = 1$. Then, the following are equivalent:*

1. *A time t as required by the* Shifted Lonely Runner Conjecture *exists for velocities $\mathbf{v}$.*
2. *The sLR zonotope Z with volume vector v has $\mu(Z) \le \frac{n-1}{n+1}$.*

2.3 Covering radius via fundamental domains

In order to apply this result one does not need to compute $\mu(Z)$ (which is quite expensive, see e.g. [5]), but only check whether a certain number is a bound for it. This checking is closely related to finding a fundamental domain inside a scaled copy of Z.

Recall that a *fundamental domain* of $\mathbb{R}^d$ (with respect to $\mathbb{Z}^d$) is a set containing exactly one representative of each coset $\mathbf{p} + \mathbb{Z}^d$, $\mathbf{p} \in \mathbb{R}^d$.The definition of covering radius trivially translates to:

Lemma 7 *Let C be a convex body in $\mathbb{R}^d$ and $\rho > 0$. Then, the following are equivalent:*

1. *$\mu(C) \le \rho$.*
2. *ρC contains representatives of all $\mathbb{R}^d/\mathbb{Z}^d$.*
3. *ρC contains a fundamental domain.*
4. *No open set $W \in \mathbb{R}^d$ satisfies $\rho C \cap (W + \mathbb{Z}^d) = \emptyset$.*

Proof. The implications (1) $\Leftrightarrow$ (2) $\Leftrightarrow$ (3) and (2) $\Rightarrow$ (4) are obvious. Let us prove (4) $\Rightarrow$ (1). $\mu(C) > \rho$ implies there is a $\mathbf{p} \notin \rho C + \mathbb{Z}^d$. Since $\rho C + \mathbb{Z}^d$ is closed, its complement $W := \mathbb{R}^d \setminus \rho C$ is open and

$$\rho C \cap (W + \mathbb{Z}^d) \subset (\rho C \cap (\mathbb{R}^d \setminus \rho C)) + \mathbb{Z}^d = \emptyset. \qquad \square$$

2.4 The denominator of the covering radius

It is well-known and easy to show that the covering radius of a rational polytope is rational (see, e.g., [10, Proposition 5.1]). We give an explicit bound for its denominator in terms of the defining equations. The denominator of a rational number ρ is defined as the minimum positive integer s such that $s\rho$ is an integer. Having a bound for the denominator allows our algorithms to certify an exact upper bound for $\mu(P)$ from an approximate one, as follows.

Proposition 8 *Let P be a rational polytope and let $D \in \mathbb{N}$ be an upper bound for the denominator of $\mu(P)$. (For example, but not necessarily, a bound obtained by Corollary 10). Let $\rho = r/s$ with $r, s \in \mathbb{Z}$ and $s > 0$. Then, the following equivalences hold:*

1. *$\mu(P) \le \rho$ if and only if $\mu(P) < \rho + \frac{1}{sD}$*
2. *$\mu(P) \ge \rho$ if and only if $\mu(P) > \rho - \frac{1}{sD}$*

Proof. One direction is obvious in both cases. For the other one, we know that $\mu(P) = \frac{r'}{s'}$ for integers r', s' with $0 < s' \le D$. Assuming $\frac{r'}{s'} \ne \frac{r}{s}$ we have that

$$|\mu(P) - \rho| = \left|\frac{r'}{s'} - \frac{r}{s}\right| = \left|\frac{r's - rs'}{s's}\right| \ge \frac{1}{ss'} \ge \frac{1}{sD}.$$

Hence, either $\mu(P) = \rho$, $\mu(P) \ge \rho + \frac{1}{sD}$ or $\mu(P) \le \rho - \frac{1}{sD}$. $\square$

Our bound uses the concept of *last covered point.*

Definition 3 (Last covered point [4,5]) *Let $C \subseteq \mathbb{R}^d$ be a convex body. A* last covered point *for C is any $\mathbf{p} \in \mathbb{R}^d$ with $\mathbf{p} \notin (\mu(C)C)^\circ + \mathbb{Z}^d$.*

Since $\mu(C)$ is invariant under translation, we may assume $\mathbf{0} \in C^\circ$ without loss of generality. This simplifies some arguments because it implies that $\rho C + \mathbf{q}$ is monotone increasing in ρ. Therefore, once a point is covered by some copy $\rho C + \mathbf{q}$, it is also covered by the same copy for any larger ρ'.

In particular, under the assumption $\mathbf{0} \in C^\circ$ we have that a point $\mathbf{p}$ is last covered if $\rho C + \mathbb{Z}^d$ does not contain $\mathbf{p}$ for any $\rho < \mu(C)$, which explains the name.

Observation 1 *The set of last covered points is always non-empty.*

Proof. Assuming without loss of generality $\mathbf{0} \in C^\circ$, for each point $\mathbf{p} \in \mathbb{R}^d$ let $\rho_{\mathbf{p}} = \min\{\rho \in \mathbb{R}_{\ge 0} : \mathbf{p} \in (\rho C)^\circ + \mathbb{Z}^d\}$ be the *covering time* of the point $\mathbf{p}$. This definition is invariant by integer translations, and continuous. Since $\mathbb{R}^d/\mathbb{Z}^d$ is compact, there must be some point $\mathbf{p}$ with maximal covering time, i.e. $\rho_{\mathbf{p}} = \mu(C)$. This point is a last covered point. $\square$

In the rest of the section, $P \in \mathbb{R}^d$ is a polytope defined by the system of inequalities $Ax \le b$ for some matrix $A \in \mathbb{R}^{m \times d}$ and vector $b \in \mathbb{R}^m$. For an element $i \in [m]$ or subset $I \subset [m]$, A_i, b_i, A_I, b_I, etc. denote the restriction of a matrix or vector to the rows labelled by i or I.

Lemma 9 ([5, Lemma 3.1]) *Let $P = \{Ax \le b\}$ be a polytope and let $\rho = \mu(P)$. Then, there is*

- *a subset $R \subset [m]$ of rows with $|R| = d + 1$ and $\det(A_R|b_R) \ne 0$ and*
- *a lattice point $\mathbf{q}_i \in \mathbb{Z}^d$ for each $i \in R$,*

such that the system

$$A_i(\mathbf{x} - \mathbf{q}_i) = \rho b_i \quad \forall i \in R \tag{2}$$

has a unique solution in $\mathbb{R}^{d+1}$ and this solution is a last covered point.

The proof uses an extremal argument and Farkas' lemma. See Section A.1 in the appendix for details.

Corollary 10 *Let P be a rational polytope described by $A\mathbf{x} \le b$ with $A \in \mathbb{Z}^{m\times d}$ and $b \in \mathbb{Z}^m$. Then $\mu(P)$ is a rational number and its denominator is bounded by*

$$\max_{R\in\binom{[m]}{d+1}} |\det(A_R|b_R)|.$$

Proof. Apply Cramer's rule to the variable t in the system of Lemma 9. □

3 Our Algorithms

In this section we introduce the algorithm we have used to find fundamental domains within each relevant sLR zonotope, postponing some details until Section 4.

Obtaining a representative zonotope for a velocity vector is discussed in [11]. We further simplify our representatives, reducing the length of their generators using the LLL algorithm, as described in Section 4.1.

3.1 Certifying an upper bound for the cov. radius

We here describe an algorithm to decide whether a facet-defined polytope $P = \{\mathbf{x} \in \mathbb{R}^d : A\mathbf{x} \le b$ contains a fundamental domain. By Lemma 7, this is equivalent to certifying a given upper bound ρ for the covering radius of a polytope.

We consider a special family of fundamental domains of the integer lattice, given by unions of *dyadic voxels.*

Definition 4 *A* dyadic d-voxel *of* level $\ell \in \mathbb{Z}_{\ge 0}$ *is a half-open cube of the form*

$$\mathbf{c} + \frac{1}{2^\ell}[0,1)^d,$$

for some dyadic point $\mathbf{c} \in \frac{1}{2^\ell}\mathbb{Z}^d$. The integer point $\lfloor \mathbf{c} \rfloor$ is the displacement *of the voxel, and the difference $2^\ell(\mathbf{c} - \lfloor \mathbf{c} \rfloor)$ is the* type *of the voxel.*

All dyadic voxels of the same type are equivalent by integer translation and the voxel types are naturally arranged as an infinite rooted 2^ℓ-ary tree with the voxels of level ℓ at depth ℓ. We call this the infinite dyadic tree.[3]

A *dyadic fundamental domain* is a fundamental domain obtained as a finite union of dyadic voxels.

Every dyadic fundamental domain can be expressed as (the leaves of) a full-subtree of the infinite dyadic tree, with leaves labelled by their displacements.

[3]One can represent each type of level ℓ as a vector $\mathbf{b} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$ where each $\mathbf{b}_i$ is a binary string of length ℓ. In this representation $\mathbf{b}$ is an ancestor of $\mathbf{b}'$ in the infinite dyadic tree if and only if each $\mathbf{b}_i$ is an initial segment, or prefix, of the corresponding $\mathbf{b}'_i$. Equivalently, if the voxel of type $\mathbf{b}$ with zero displacement is contained in that of type $\mathbf{b}'$.

The simplest of such domains is the unit cube, that is, the root of the dyadic tree. Our algorithm performs a search in the infinite dyadic tree, starting with the root and iteratively subdividing all leaves which cannot be translated to fit in our zonotope, until either (a) all leaves fit inside, which certifies that we have constructed a dyadic fundamental domain contained in P or (b) the center of one leaf is found to have no translation inside P, certifying that no such fundamental domain exists.

This algorithm is illustrated in Figure 1.

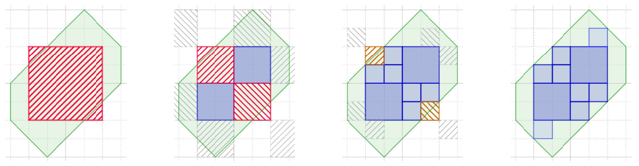


Figure 1: States of our Algorithm at different depths, applied to $\frac{1}{2}Z$ where Z is the 2-dimensional sLR zonotope with volume vector $(1, 2, 4)$. Notice that there are two choices for each voxel in the last step.

The decision of whether a voxel admits an integer translation that fits in our zonotope requires checking the feasibility of an integer linear program.

Proposition 11 *Let $P = \{A\mathbf{x} \le \mathbf{b}\} \subset \mathbb{R}^d$ be a polytope and let $V = \mathbf{c} + [0, \epsilon)^d$ be a voxel. Then, P contains an integer translation of V if and only if the following Integer Linear Program is feasible:*

$$\text{find } \mathbf{x} \in \mathbb{Z}^d \quad \text{subject to } A\mathbf{x} \le \mathbf{b} - A\mathbf{c} - A_{\ge 0}\,\boldsymbol{\epsilon}, \quad (3)$$

where $\boldsymbol{\epsilon} \in \mathbb{R}^d$ is the vector with all entries equal to ϵ and $A_{\ge 0}$ denotes the matrix with entry (i,j)-th entry equal to $\max\{0, A_{ij}\}$, for every (i,j).

Proof. A half-empty voxel is contained in P if and only the closed voxel is, that is, if and only of all the vertices of the translated closed voxel are in P. Hence, for a given $\mathbf{x} \in \mathbb{Z}^d$, the voxel $\mathbf{x} + V = \mathbf{x} + \mathbf{c} + [0, \epsilon)^d$ is contained in P if and only if all the points $\mathbf{y} \in \{0, \epsilon\}^d$ satisfy the inequalities $A(\mathbf{y} + \mathbf{c} + \mathbf{x}) \le \mathbf{b}$. Now, for each row A_i of A, the maximum value of the functional A_i on the set $\{0, \epsilon\}^d$ is precisely $(A_i)_{\ge 0}\,\boldsymbol{\epsilon}$. □

The search algorithm as described so far has one issue: P may contain a fundamental domain but no dyadic one, and in this case our algorithm does not terminate. For rational polytopes we can solve this issue thanks to Proposition 8.

Theorem 12 *Let P be a rational polytope and let D be an upper bound for the denominator of $\mu(P)$. Let $\rho = r/s$ with $r, s \in \mathbb{Z}$ and $s > 0$.*

1. *If $\mu(P) \le \rho$ then $\left(\rho + \frac{1}{2sD}\right)P$ contains a dyadic fundamental domain.*

2. If $\mu(P) > \rho$ then there is an $\ell \in \mathbb{Z}_{\geq 0}$ and a dyadic point $\mathbf{c} \in \frac{1}{2^\ell}\{0, \ldots, 2^\ell - 1\}^d$ such that $\left(\rho + \frac{1}{2sD}\right) P$ does not intersect $\mathbf{c} + \mathbb{Z}^d$.

See Section 4.2 for a proof.

Hence, rather than applying our algorithm to the zonotopes dilated by $\frac{3}{5}$ we dilate them by $(\frac{3}{5} + \frac{1}{10D})$, which does not affect the result, yet ensures the algorithm terminates.

4 Algorithm details

4.1 Enumeration, construction, and preprocessing of sLR zonotopes

According to Theorem 1 we only need to enumerate sLR zonotopes up to volume 195. We first construct the list of possible volume vectors, that is, the 4-tuples $v = (v_1, \ldots, v_4) \in \mathbb{Z}^4$ with $0 < v_1 < v_2 < v_3 < v_4$. As observed in the introduction we can assume that $\gcd(v_1, v_2, v_3, v_4) = 1$. Moreover, by Proposition 5, with this restriction there is a unique sLR zonotope (modulo unimodular equivalence) for each volume vector. Enumerating such 4-tuples is algorithmically trivial and took less than a second in a standard PC:

Proposition 13 *There are exactly* 2 133 561 *vectors* $(v_1, v_2, v_3, v_4) \in \mathbb{Z}$ *with* $1 \leq v_1 < v_2 < v_3 < v_4$, $\gcd(v_1, v_2, v_3, v_4) = 1$ *and* $\sum v_i \leq 195$.

We then need to generate a representative zonotope from its volume vector v. This is done with Algorithm 1, which follows the 'existence' part of the proof of Proposition 5 given in [11].

Algorithm 1: Compute generators for a LR zonotope from its volume vector.

```
Input  : $v = (v_1, \ldots, v_n) \in \mathbb{Z}^n_{>0}$, with $\gcd(v_1, \ldots, v_n) = 1$.
Output: A matrix $M = (\mathbf{u}_1, \ldots, \mathbf{u}_n) \in \mathbb{Z}^{(n-1)\times n}$
        such that $\mathbf{u}_1, \ldots, \mathbf{u}_n$ generate a LR
        zonotope with volume vector $v$.
1 Let $M' := \left(\begin{array}{ccc|c} -v_n & & & v_1 \\ & \ddots & & \vdots \\ & & -v_n & v_{n-1} \end{array}\right)$.
2 Let $H \in \mathbb{Z}^{(n-1)\times n}$ be the column-wise Hermite
  normal form of $M$, and let $B \in \mathbb{Z}^{(n-1)\times(n-1)}$
  consist of the first $n-1$ columns of $H$.
3 Apply an LLL-reduction to the rows of $B^{-1}M'$
  and let $M \in \mathbb{Z}^{(n-1)\times n}$ have as rows the resulting
  reduced vectors.
4 return $M$.
```

Step 1 in the algorithm creates an integer matrix $M' \in \mathbb{Z}^{(n-1)\times n}$ whose columns generate a LR zonotope with volume vector a scalar multiple of $(v_1, \ldots, v_n)$. Step 2 then uses a column-wise Hermite normal form of M' to construct a basis (the columns of the matrix B in the algorithm) of the lattice Λ generated by the $\mathbf{u}'_i$. Observe that $\mathrm{rk}(M') = n-1$ implies that the last column of its Hermite normal form H is zero, and B is simply equal to H without that column.

Now, B^{-1} is the matrix of a linear isomorphism $\Lambda \xrightarrow{\cong} \mathbb{Z}^{n-1}$, so the columns of $B^{-1}M'$ would already be valid generators for a LR zonotope with volume vector $(v_1, \ldots, v_n)$. However, the generators obtained in this way typically have some large entries, resulting in 'long and skinny' zonotopes that are are poorly conditioned for our method to compute covering radii. To overcome this we preprocess the generators in step 3 of the algorithm, by performing an LLL lattice basis reduction to the rows of $B^{-1}M'$.[4] This produces a matrix M whose columns are unimodularly equivalent to those of $B^{-1}M'$, but with smaller entries.

For our covering radius computations we need to convert the generators of the zonotope into an inequality description of it. This, for an arbitrary zonotope $Z \subset \mathbb{R}^d$ with generators $U = \{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$ is done as follows, where we are identifying $\bigwedge^{d-1} \mathbb{R}^d \cong (\mathbb{R}^d)^*$ in the natural way.

Proposition 14 *Let* $Z = \frac{1}{2}\sum_{i=1}^n [-\mathbf{u}_i, \mathbf{u}_i]$ *be the* $\mathbf{0}$-*symmetric zonotope with generators* $\mathbf{u}_1, \ldots, \mathbf{u}_n$. *Then*

$$Z = \left\{ \mathbf{x} \in \mathbb{R}^d : -b_S \leq a_S \mathbf{x} \leq b_S : S \in \binom{[n]}{d-1} \right\},$$

where

$$a_S := \bigwedge_{i \in S} \mathbf{u}_i \in (\mathbb{R}^d)^* \qquad \text{and} \qquad b_S := \frac{1}{2}\sum_{i=1}^n |a_S\, \mathbf{u}_i|.$$

Proof. Each facet of a zonotope is a zonotope itself, generated by the $\mathbf{u}_i$ contained in, and spanning, a linear hyperplane. Hence, every normal vector is indeed of the form a_S for some $(d-1)$-subset S of U.

By central symmetry, there are two parallel facets with normal vectors $\pm a_S$. The corresponding facet inequalities are $-b_S \leq a_S \mathbf{x} \leq b_S$, since $\pm b_S$ are the minimum and maximum values taken by a_S in the set

$$\left\{ \sum_{i=1}^n \pm \mathbf{u}_i \right\},$$

which contains all vertices of Z. $\square$

4.2 Building a dyadic fundamental domain

In this section we present Algorithm 2, the concrete algorithm that explores the infinite dyadic tree to decide

[4] We have implemented the LLL algorithm with $\delta = 3/4$. Higher values of $\delta(0, 1)$ would give better zonotopes, but would increase the running time.

the covering radius of an arbitrary lattice polytope, as discussed in Section 3.

The algorithm requires a facet description of the polytope, which can be derived from the generators of a zonotope by Proposition 14.

The termination of this algorithm follows from Theorem 12, of which we give now proof.

Proof. (of Theorem 12) For part (1) we only need to use that

$$\mu(P^+) = \frac{\mu(P)}{\rho + \frac{1}{2sD}} \le \frac{\rho}{\rho + \frac{1}{2sD}} < 1.$$

For each $\ell \in \mathbb{N}$ let D_ℓ be the union of all the dyadic boxes of depth ℓ contained in P^+. Since D_ℓ converges (e.g. in the Hausdorff metric) to P^+ when ℓ goes to infinity, we have that $\mu(D_\ell)$ converges to $\mu(P^+)$. In particular, there is an ℓ such that $\mu(D_\ell) < 1$. Hence, D_ℓ contains a fundamental domain, and this fundamental domain can be obtained taking one representative for each type of voxel in the union D_ℓ.

For part (2) we use that $\mu(P) > \rho$ implies (by Corollary 8) that $\mu(P) \ge \rho + \frac{1}{sD}$. Hence

$$\mu(P^+) = \frac{\mu(P)}{\rho + \frac{1}{2sD}} \ge \frac{\rho + \frac{1}{sD}}{\rho + \frac{1}{2sD}} > 1.$$

The statement then follows from the density of the dyadic points $\mathbb{Z}[\frac{1}{2}]^d$ in $\mathbb{R}^d$ and Lemma 7, which asserts the existence of an open set $W \subset \mathbb{R}^d \setminus (P^+ + \mathbb{Z}^d)$. □

Theorem 15 *Algorithm 2 always terminates and it correctly decides whether $\mu(P) \le \rho$ for any lattice polytope P and $\rho \in \mathbb{Q}_+$.*

Proof. Observe that the algorithm returns a certificate in either case. Let us first show their correctness.

If the algorithm finishes with a set of dyadic voxels, these voxels are a full subtree of the infinite dyadic tree by construction, and hence they form a dyadic fundamental domain. Furthermore, all of these voxels are contained in P^+, so $\mu(P) \le (\rho + \frac{1}{2sD})$ and Corollary 8 implies $\mu(P) \le \rho$.

On the other hand, if the algorithm finishes with a point $\mathbf{c}$ such that P^+ does not intersect $\mathbf{c}+\mathbb{Z}^d$, Lemma 7 implies $\mu(P) > (\rho + \frac{1}{2sD}) > \rho$.

To prove that the algorithm terminates we handle the two cases separately.

If $\mu(P) \le \rho$, Theorem 12 guarantees the existence of a dyadic fundamental domain D contained in P^+. Let ℓ be the maximum depth of the voxels in D. Then every voxel type of depth $\ge \ell$ has a representative contained in P^+, so the algorithm will never enter the "else" in line 14 with a voxel of depth $\ge \ell$. Hence, the algorithm can perform the while loop only finitely many times before N becomes empty.

Algorithm 2: Decide whether $\mu(P) \le \rho$.

```
Input  : A rational polytope $P = \{A\mathbf{x} \le \mathbf{b}\}$
         (with $A$ and $\mathbf{b}$ integer) and a rational
         number $\rho = r/s$, with $r, s \in \mathbb{Z}_+$.
Output: A dyadic fundamental domain $S$ or a
         dyadic point $\mathbf{c}$ certifying whether
         $\mu(P) \le \rho$ or not, as in Theorem 12.
1  Let $D$ be a bound on the denominator of $\mu(P)$,
     such as $\mu_{det}(A, \mathbf{b})$.
2  Let
     $$P^+ = \left(\rho + \frac{1}{2sD}\right) P = \left\{A\mathbf{x} \le \left(\rho + \frac{1}{2sD}\right) \mathbf{b}\right\}$$
3  Initialise a queue $N$ of 'nodes to be processed'
     containing the unit cube
4  Initialise an empty list $S$ of 'voxels in the
     fundamental domain'
5  while there are nodes in $N$ do
6      Let $V = \mathbf{c} + [0, \frac{1}{\ell^d})^d$ be one such node of
         maximum size.
7      Delete $V$ from $N$ and
8      if $P^+$ does not intersect $\mathbf{c} + \mathbb{Z}^d$ then
9          return $\mathbf{c}$
10     else
11         if $\exists\, \mathbf{p} \in \mathbb{Z}^d$ with $\mathbf{p} + V \subset P^+$ then
12             add the voxel $\mathbf{p} + V$ to $S$
13         else
14             add the $2^d$ children of $V$ to $N$
15 return $S$
```

If $\mu(P) > \rho$, Theorem 12 guarantees the existence of a dyadic point $\mathbf{c}$ with $(\mathbf{c} + \mathbb{Z}^d) \cap P^+ = \emptyset$. Let ℓ be the minimal depth of such a point. Since the algorithm processes the infinite dyadic tree in a breadth-first search manner, in a finite number of steps it will check all the dyadic points of depth ℓ (either implicitly for those contained in voxels of depth $\le \ell$ and with $\mathbf{p} + V \subset P^+$, or explicitly for those not contained in such voxels). □

References

[1] D. Alcántara, F. Criado, F. Santos. Covering radii of 3-zonotopes and the shifted Lonely Runner Conjecture, preprint, 2025. https://arxiv.org/abs/2506.13379

[2] J. Barajas and O. Serra. The lonely runner with seven runners. *Electron. J. Combin.*, 15:#48, 18 pp. (electronic), 2008.

[3] M. Beck, S. Hoşten, and M. Schymura. Lonely Runner Polyhedra. *Integers*, 19:#A29, 13 pp, 2019.

[4] G. Codenotti, F. Santos, and M. Schymura. The covering radius and a discrete surface area for non-hollow simplices. *Discrete & Computational Geometry*, 67(1):65–111, 2022.

[5] J. Cslovjecsek, R. D. Malikiosis, M. Naszódi, and M. Schymura. Computing the covering radius of a polytope with an application to lonely runners. *Combinatorica*, 42(4):463–490, Feb. 2022.

[6] T. W. Cusick. View-obstruction problems. *Aequationes Math.*, 9:165–170, 1973.

[7] T. W. Cusick and C. Pomerance. View-obstruction problems III. *J. Number Theory*, 19:131–139, 1984.

[8] M. Henze and R. D. Malikiosis. On the covering radius of lattice zonotopes and its relation to view-obstructions and the lonely runner conjecture. *Aequationes Math.*, 91(2):331–352, 2017.

[9] Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, Dec. 2017.

[10] R. Kannan. Lattice translates of a polytope and the frobenius problem. *Combinatorica*, 12(2):161–177, June 1992.

[11] R. D. Malikiosis, F. Santos, and M. Schymura. Linearly-exponential checking is enough for the lonely runner conjecture and some of its variants, preprint, 2024. https://arxiv.org/abs/2411.06903v2

[12] G. Perarnau and O. Serra. The Lonely Runner Conjecture turns 60, preprint, 2024. https://arxiv.org/abs/2409.20160.

[13] L. Rifford. On the time for a runner to get lonely, 2021. *Acta Appl. Math.*, 180, 15, 2022.

[14] J. M. Wills. Zur simultanen homogenen diophantischen Approximation. I. *Monatsh. Math.*, 72:254–263, 1968.

[15] J. M. Wills. Zur simultanen homogenen diophantischen Approximation. II. *Monatsh. Math.*, 72:368–381, 1968.

Appendix

A.1 Bound for the denominator of the cov. radius

Proof. (of Lemma 9) As in Observation 1, we may assume $\mathbf{0} \in P^\circ$ without loss of generality, or equivalently, that $b_i > 0$ for all $i \in [m]$.

Each facet of a translated polytope $\{\rho P + \mathbf{q} : \mathbf{q} \in \mathbb{Z}^d\}$ is labelled by a point $\mathbf{q} \in \mathbb{Z}^d$ and an index $i \in [m]$. For each last covered point $\mathbf{p}$, let $R_\mathbf{p}$ be the set of indices i such that $\mathbf{p}$ lies in the i-th facet of $\rho P + \mathbf{q}_i$, for some $\mathbf{q}_i \in \mathbb{Z}^d$. Observe that $\mathbf{p}$ lies in the affine subspace $L_\mathbf{p} := \{\mathbf{x} \in \mathbb{R}^d : \forall i \in R_\mathbf{p}, A_i(\mathbf{x} - \mathbf{q}_i) = \rho b_i\}$, since $A_i(\mathbf{x} - \mathbf{q}_i) = \rho b_i$ is the facet equation for the i-th facet of $\rho P + \mathbf{q}_i$.

Choose a last covered point $\mathbf{p}$ so that $R_\mathbf{p}$ is maximal. Maximality implies that $L_\mathbf{p} = \{\mathbf{p}\}$, since otherwise moving the point within $L_\mathbf{p}$ until an extra facet of some $\{\rho P + \mathbf{q} : \mathbf{q} \in \mathbb{Z}\}$ is met (which happens at the latest when we are about to leave a certain $\rho P + \mathbf{q}_i$ containing $\mathbf{p}$) gives us a last covered point $\mathbf{p}'$ with $R_{\mathbf{p}'}$ strictly containing $R_\mathbf{p}$.

The fact that $L_\mathbf{p} = \{\mathbf{p}\}$ implies that the matrix $A_{R_\mathbf{p}}$ consisting of rows used in $R_\mathbf{p}$ has full rank, equal to d.

Now, observe that the vectors A_i for $i \in R_\mathbf{p}$ must have a positive linear dependence. Otherwise, by (one of many versions of) Farkas' lemma, there is a vector $\mathbf{v} \in \mathbb{R}^d$ such that $\langle A_i, \mathbf{v} \rangle > 0$ for all $i \in R_\mathbf{p}$. Then, $\mathbf{p}$ would not be last covered, as $\mathbf{p} + \varepsilon \mathbf{v}$ would not be covered by any $P + \mathbf{q}_i$. Locally, these are the only translated copies of P that could potentially cover $\mathbf{p}$. Therefore, no translated copy of P covers $\mathbf{p} + \varepsilon \mathbf{v}$, so $\mathbf{p} + \varepsilon \mathbf{v}$ would have larger covering time than $\mathbf{p}$, contradicting the assumption of $\mathbf{p}$ being last-covered.

The positive linear dependence of the vectors A_i for $i \in R_\mathbf{p}$ implies that the system of equalities

$$A_i(\mathbf{x} - \mathbf{q}_i) = t b_i, \quad i \in R_\mathbf{p},$$

where t is considered an extra variable, has no solution with $t \in [0, \rho)$. Indeed, let $\lambda_i \in \mathbb{R}_{\geq 0}$ for $i \in R_\mathbf{p}$ be the coefficients of the linear dependence we defined above. Then,

$$\begin{aligned} A_i((\mathbf{x} - \mathbf{p}) + \mathbf{p} - \mathbf{q}_i) &= (t - \rho + \rho) b_i, & i \in R_\mathbf{p}, \\ A_i(\mathbf{x} - \mathbf{p}) + A_i(\mathbf{p} - \mathbf{q}_i) &= \rho b_i + (t - \rho) b_i, & i \in R_\mathbf{p}, \\ A_i(\mathbf{x} - \mathbf{p}) &= (t - \rho) b_i, & i \in R_\mathbf{p}, \end{aligned}$$

$$\begin{aligned} \sum_{i \in R_\mathbf{p}} \lambda_i A_i(\mathbf{x} - \mathbf{p}) &= \sum_{i \in R_\mathbf{p}} \lambda_i (t - \rho) b_i \\ 0 &= \sum_{i \in R_\mathbf{p}} \lambda_i (t - \rho) b_i \\ 0 &= (t - \rho) \left(\sum_{i \in R_\mathbf{p}} \lambda_i b_i \right). \end{aligned}$$

But since the λ_i are non-negative (and not all of them are zero) and the b_i are positive, then it must be that $t = \rho$.

Thus, the system has only solutions of the form $(\mathbf{x}, \rho)$ and, hence, only the solution $(\mathbf{p}, \rho)$. This implies that the matrix $(A_{R_\mathbf{p}} | - b_{R_\mathbf{p}})$ has rank $d + 1$.

Choose R to be a basis for the rows of $(A_{R_\mathbf{p}} | b_{R_\mathbf{p}})$. □

A.2 Implementation considerations

Our implementation of Algorithm 2 uses the HiGHS MIP solver [9] to determine the feasibility of Integer Linear Problems defined in Proposition 11.

Since the MIP solver relies on numerical methods and hence is subject to numerical errors, we round all proposed solutions and check them for feasibility under exact linear algebra. We encountered no issues of this kind solving any of the ILPs needed to construct certificates for all volume vectors with volume at most 195.

In such cases, in lack of an exact MIP solver, a brute force approach could be used, checking all candidate translations within the bounding box of the zonotope.

Dyadic fundamental domains with small circumradius or small volume of convex hull can be obtained with the same algorithm, but turning the feasibility problem (3) from Proposition 11 into an optimization problem that minimizes some norm. This modification does not affect the search strategy or the types of voxels obtained in the final fundamental domain; it just gives the "best" representative of each type.

For example, the optimization problem for the Minkowski norm of P is particularly simple:

$$\begin{aligned} \text{minimize} \quad & \rho \in \mathbb{R} \\ \text{subject to} \quad & A\mathbf{x} - b\rho \leq -A\mathbf{c} - A_{\geq 0}\,\boldsymbol{\epsilon} \\ & \rho \geq 0 \\ & \mathbf{x} \in \mathbb{Z}^d. \end{aligned}$$

Optimizing with respect to this norm results in a dyadic fundamental domain fitting in the smallest possible dilation of the zonotope, among those with the types given by the breadth-first search.

The convex hull of the domain can be minimized even more by further subdividing all voxel types to reach a regular tree of any given depth, at the expense of solving many more ILPs.

Since our zonotopes are centrally symmetric around the origin, a voxel type will lie in our fundamental domain if and only if the opposite type does. Hence, we only need to check half of the voxels in the first subdivision of the unit cube, which automatically gives that we check only half of each level. This has the advantage of halving the execution time and producing centrally symmetric certificates, which are both smaller and visually clearer.

Implementations can easily avoid having to deal with rational matrices and vectors by scaling by the common denominator.

Square Packing with Asymptotically Smallest Waste Only Needs Good Squares

Hong Duc Bui*

Abstract

We consider the problem of packing a large square with nonoverlapping unit squares. Let $W(x)$ be the minimum wasted area when a large square of side length x is packed with unit squares. In Roth and Vaughan's article that proves the lower bound $W(x) \notin o(x^{1/2})$, a good square is defined to be a square with inclination at most 10^{-10} with respect to the large square. In this article, we prove that in calculating the asymptotic growth of the wasted space, it suffices to consider only packings with only good squares. This allows the lower bound proof in Roth and Vaughan's article to be simplified by not having to handle bad squares.

1 Introduction

There has been much research on the problem of packing unit squares into a large square. Define $W(x)$ to be the minimum area of the wasted part when packing unit squares into a large square of side length x. On one hand, explicit constructions have been given in the literature in [2,3,8], which successively improve the upper bound to $W(x) \in O(x^{0.637})$, $W(x) \in O(x^{0.631})$ and $W(x) \in O(x^{0.625})$ respectively. On the other hand, [7] proves a lower bound that $W(x) \notin o(x^{0.5})$.

In the proof of the lower bound $W(x) \notin o(x^{0.5})$ in [7], a good unit square was defined to be one with inclination $\leq 10^{-10}$ with respect to the large square, and a bad unit square is one that isn't good. The proof of [7, lemma 4] uses the fact that there aren't too many bad unit squares.

In this article, we prove that it suffices to only consider the case where there are no bad squares, simplifying the proof. Furthermore, if some stronger lower bound is to be proven, the proof can make use of this result to avoid handling bad squares.

The method used in this article can be generalized to the case of packing a large quadrilateral with all angles almost equal to a right angle.

More formally, let $W^*(x)$ be the smallest wasted area when a large square of side length x is packed with only good unit squares.

Theorem 1 *$W(x) \in \Theta(W^*(x))$.*

*buihd@u.nus.edu

The main idea of the proof is the following. Given any packing, we show it can be modified to cut away all bad squares while keeping the wasted area sufficiently small. By dividing the large square into a grid of small squares, and using Proposition 17, we simultaneously construct a set of k disjoint rectangles $\{R_1, \ldots, R_k\}$ covering all bad squares, and a set of f disjoint paths $\{P_1, \ldots, P_f\}$ satisfying certain properties. Then, we modify the packing in the region inside the rectangles $\{R_i\}$, and prove that the modification introduces additional wasted area at most linear in the total perimeter of R_i. Using Proposition 19, which is a generalization of the fundamental lemma in [7] to handle squares far apart, we prove that the initial packing has wasted area $\in \Omega(f)$, the number of paths. Finally, the construction in Proposition 17 guarantees the total perimeter of R_i is $O(f)$; therefore, the increase in wasted area is at most linear in the existing wasted area, finishing the proof.

2 Notation

2.1 Notations Pertained to the Geometry of Square Packing

We assume all squares live in an ambient Euclidean plane $\mathbb{R}^2$ with a coordinate axis.

Let S_0 be an axis-aligned square of side length x.

Definition 2 (Packing) *Let S_0 be fixed. A collection of unit squares $\mathcal{A} = \{S_1, S_2, \ldots, S_k\}$ is called a packing of S_0 if the open unit squares S_i are non-overlapping and contained inside S_0. Define $|\mathcal{A}|$ to be the number of unit squares, so $|\mathcal{A}| = k$.*

From now on, $S_1, S_2, \ldots$ will be used to denote unit squares in a packing. This notation is the same as [7].

Definition 3 (S_i as a set of points) *When we use S_i as a set of points on the plane, we implicitly mean the set of points on the boundary of the square. Here i may either be 0 (S_0 is the large square), or $i > 0$.*

For $i > 0$, the set of points on the boundary and inside *the square S_i is denoted S_i^*. The set of points on the boundary* and outside *the square S_0 is denoted S_0^*.*

With this definition, the intersection of S_i^* and S_j^* for any $0 \leq i < j \leq |\mathcal{A}|$ has area 0.

Definition 4 (Wasted area of a packing $W(\mathcal{A})$) *Let $W(\mathcal{A}) = x^2 - |\mathcal{A}|$, where, as mentioned above, x is the side length of S_0.*

Let us formally define the function $W(x)$ mentioned in the introduction.

Definition 5 (The waste function $W(x)$) *Let $\mathcal{A}$ be a packing with the largest number of unit squares for a fixed value of the side length x. Then define the minimum wasted area when packing a square of side length x to be $W(x) = W(\mathcal{A})$.*

Equivalently, $W(x) = \min_{\mathcal{A}} W(\mathcal{A})$ over all packings $\mathcal{A}$ of a square S_0 with side length x.

Definition 6 (Wasted area in any shape) *Fix the large square S_0 and a packing $\mathcal{A}$. Let U be a Lebesgue measurable set on the plane. Then define $W(U) = |U \setminus (S_0^* \cup S_1^* \cup S_2^* \cup \cdots \cup S_k^*)|$. In words, $W(U)$ is the total area in U that is inside the large square and not covered by any unit squares.*

This notation $W(U)$ is distinct from $W(\mathcal{A})$, the wasted area of a packing, and should be understood from the context, depending on whether the argument is a set or a packing. They are both used to denote wasted area, however.

Definition 7 (Angle between squares) *Let S_a and S_b be squares (either an unit square used for packing, or the boundary). We say S_a has the same orientation as S_b if any edge of S_a is parallel to any edge of S_b. Let $\theta(S_a, S_b)$ be the minimum rotation angle in radian of one of the squares to have the same orientation as the other. We allow either clockwise or counterclockwise rotation.*

Note that with our definition, $\theta(S_a, S_b) \leq \frac{\pi}{4}$ for every a and b.

Definition 8 (Distance) *Let p, q be points in the plane and U, V be nonempty sets of points in the plane. Define $d(p, q)$ to be the distance from p to q and $d(U, V) = \min_{u \in U, v \in V} d(u, v)$. Define $d(U, p) = d(U, \{p\})$.*

The minimum in the definition of $d(U, V)$ may not always exist. However, the minimum exists in all instances in this article, because both T and U are closed and at least one of them is compact.

Definition 9 (Path on a plane) *We define a path on the plane to be a continuous function $\gamma \colon [a, b] \to \mathbb{R}^2$ for real numbers $a < b$.*

Define the image of the path on the plane to be $\operatorname{im} \gamma = \{\gamma(x) \mid x \in [a, b]\} \subseteq \mathbb{R}^2$.

Definition 10 (The open ball around an object) *For a closed set S in the plane, define*

$$B(S, r) = \{\textit{point } p \mid d(S, p) < r\}.$$

Following [7], we define the constant $c = 10^{-10}$. This value is only selected for concreteness, our argument works for any $c > 0$.

We also define a good square according to [7]:

Definition 11 (Good squares and bad squares) *A unit square $S_a \in \{S_1, S_2, \dots\}$ is called good if $\theta(S_0, S_a) \leq c$. It is called bad if it is not good.*

Definition 12 (Good packing, $W^*(x)$ function) *A packing $\mathcal{A}' = \{S_1', \dots, S_k'\}$ is called good if all unit squares S_1', ..., S_k' are good. Define $W^*(x) = \min_{\mathcal{A}'} W(\mathcal{A}')$ over all good packings $\mathcal{A}'$ of the large square S_0 with side length x.*

The definition implies that $W^*(x) \geq W(x)$.

2.2 Notations Pertained to the Combinatorial Tools

Definition 13 (Grid, edge, neighborhood) *Define a $n \times m$ grid to be a grid consisting of $n \times m$ squares (which we will call cells) glued side-by-side, such that there are n rows and m columns. An* edge *of the grid is defined to be an edge of any cell.*

The neighborhood *of a cell is defined to be the set of cells that share an edge with that cell. This is also known in the literature as the 4-neighborhood or von Neumann neighborhood of the cell. We also say the two cells to be* adjacent*.*

A cell is said to be on the boundary *of the grid if it has fewer than four adjacent cells.*

See Figure 1 for an illustration. We see that the grid has $(n+1) \times m$ horizontal edges, $n \times (m+1)$ vertical edges, and $n \cdot m - \max(0, n-2) \cdot \max(0, m-2)$ boundary cells.

Definition 14 (Rectangle on a grid) *Consider a $n \times m$ grid. A* rectangle *on the grid is defined to be a tuple of integers (i, i', j, j') such that $1 \leq i \leq i' \leq n$, $1 \leq j \leq j' \leq n$. The cells of that rectangle are defined to be the set of all cells on row i^* and column j^* for all integers $i \leq i^* \leq i'$ and $j \leq j^* \leq j'$.*

A cell is contained in *the rectangle if it belongs to the set of cells of the rectangle as above. Equivalently, we also say that the rectangle* covers *the cell.*

The cellular perimeter *of the rectangle is defined to be $2 \cdot ((i' - i + 1) + (j' - j + 1))$; equivalently, this is the number of grid edges that is on the boundary of the rectangle.*

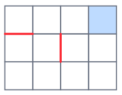


Figure 1: Example of a 3×4 grid. Some edges are colored in red, and a boundary cell is colored in blue.

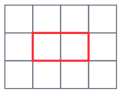


Figure 2: Example of a rectangle on a grid. The rectangle is marked in red.

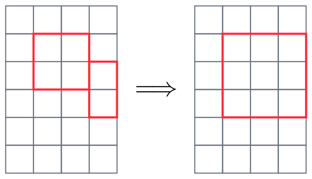


Figure 3: Illustration of merging two rectangles that have at least a point in common. Here, they have a grid edge in common, and the total perimeter decreases by 2 after merging.

See Figure 2 for an illustration.

We need to distinguish cellular perimeter and perimeter because later on there may be confusion if the side length of each cell is not exactly 1 unit in length. Whenever clear, we will just say perimeter.

Definition 15 (Path on a grid) *Let A be a cell in a grid. A* path from A to the boundary *is a sequence of cells $(Q_1, Q_2, \ldots, Q_k)$ such that $Q_1 = A$, Q_k is on the boundary of the grid, all Q_i are different, and for each integer $1 \le i < k$, Q_i and Q_{i+1} are adjacent.*

3 Combinatorial Tools

We need some helpers to prove the theorem.

Lemma 16 *Let C be a set of cells on a grid with $k > 0$ cells. Then there exists a set of pairwise disjoint rectangles on the grid with total perimeter no more than $4k$, such that C is contained in the union of these rectangles.*

Proof. For each cell in C, there exists a unique rectangle with perimeter 4 that contains that cell. We start with the collection of k such rectangles. It can be seen that the total perimeter of these rectangles is exactly $4k$, and C is contained in the union of these rectangles.

While there are still two rectangles with at least a point in common, we merge them, i.e. find the smallest rectangle that contains both such rectangles. See Figure 3 for an illustration.

Because there are only finitely many rectangles, this process terminates in finite time. Also notice that the total perimeter is non-increasing during the process. □

Proposition 17 *Let M be a collection of cells on a $n \times m$ grid, which we call the* marked *cells. Then there exists an integer $f \ge 0$ such that:*

- *For each integer $1 \le i \le f$, there is a path $P_i = (P_{i,1}, P_{i,2}, \ldots, P_{i,k_i})$ from a marked cell to the boundary (that is, $P_{i,1} \in M$ is marked and P_{i,k_i} is on the boundary);*
- *Each cell is used only once over all paths. In other words, there is no (i, j, i', j') such that $P_{i,j} = P_{i',j'}$ but $(i, j) \ne (i', j')$.*
- *There is a collection of rectangles $\{R_1, \ldots, R_k\}$ of the grid such that all marked cells are contained in some rectangle, and their total perimeter is $\le 4f$.*

If we compare this with [7], the arguments in the proof of [7, Lemma 4] can show that the cells in M appear in either $\ge \sqrt{|M|}$ distinct rows or $\ge \sqrt{|M|}$ distinct columns, because otherwise there would be less than $\sqrt{|M|} \cdot \sqrt{|M|}$ locations where marked cells can be at, which is a contradiction. This implies by taking the trivial vertical or horizontal paths along the rows or columns containing a marked cell, we obtain $f \ge \sqrt{|M|}$ distinct paths. Our proof is stronger in that it can relate the value of f with the total perimeter of the rectangles covering the marked cells.

Proof. We make use of the max-flow min-cut theorem. This is a very well-known theorem, one proof can be found in [5].

We want to find the smallest set of cells whose removal disconnects the boundary from all marked cells. To do so, we construct a flow graph, each cell corresponds to a vertex, each vertex has capacity 1, and find its min-cut. In order to impose a capacity 1 on each vertex, we use the vertex splitting technique [4, 6].

Formally, we construct the flow graph as follows.

For each $1 \le i \le n$ and $1 \le j \le m$, construct node $A_{i,j}$ and $B_{i,j}$. Each cell on row i and column j corresponds to two nodes $A_{i,j}$ and $B_{i,j}$, which we will call the A-node and B-node of the cell respectively. Besides, there are source node s and sink node t.

For each cell, connect its A-node to its B-node with a directed edge of capacity 1.

For each cell on the boundary, connect its B-node to the sink t with an edge of capacity ∞.

For each marked cell, connect the source s to the cell's A-node with an edge of capacity ∞.

For each pair of adjacent cells (i, j) and (i', j'), connect $B_{i,j}$ to $A_{i',j'}$ with an edge of capacity ∞.

Note that when all edges of capacity $< \infty$ are removed, there is no path from s to t, therefore the flow is finite. Since all edge weights are integers, the flow is an integer, let this be f. Furthermore, there exists a maximum flow where the amount of flow through each edge is an integer.

If an algorithm such as Ford–Fulkerson is used to compute the flow, because all edge weights are integral, the flow value through each edge is also integral. Therefore, the flow can be decomposed into f unit flows from s to t, each having weight 1, plus some circulations which can be ignored. By taking the cells that corresponds to the edges with capacity 1 along each path, we get a collection of paths $P_i = (P_{i,1}, \dots, P_{i,k_i})$ for each $1 \le i \le f$. By construction, this is a path from $P_{i,1}$ (which is a marked cell) to the boundary.

Because each edge from a cell's A-node to its B-node has capacity 1, each cell is used in at most one of the paths P_i, and each path only consists of distinct cells.

From the procedure above, we have constructed a collection of paths $\{P_i\}$ as required. Now we need to construct the collection of rectangles $\{R_i\}$.

By the max-flow min-cut theorem, there is a collection of edges with total capacity f such that cutting these edges results in no path from the source s to the sink t. Because each edge with finite capacity has capacity 1, there must be f cut edges. Furthermore, each of them connects some cell's A-node to its B-node. Let the set of cells corresponding to these cut edges be C.

Applying Lemma 16, there exists a set of disjoint rectangles $\{R_i\}$ covering C with total perimeter no more than $4f$.

It remains to show the set of disjoint rectangles above also covers all the marked cells. Because the edges corresponding to the cells in C form a cut, for each marked cell, every path from that cell to the boundary contains some cell in C. We will show the contrapositive: if a cell is not covered by any rectangle R_i, then there is a path from that cell to the boundary that does not use any cell in C.

This is a corollary of Alexander duality [1]: because $\widetilde{H}^1$ of a disjoint union of rectangles is the trivial group, so is $\widetilde{H}^0$ of its complement, therefore the complement of $\bigcup R_i$ is connected.

We can also show this by a more elementary argument. Starting from any uncovered cell, move down as far as possible. If the cell below is on the boundary, we are done. Otherwise, the cell below is part of a rectangle. Because the rectangles are disjoint, the region with width 1 around each rectangle is not covered. Therefore, we can move to the left to the end of the rectangle, and if we did not hit the boundary, continue to move downwards. Because we only move downwards or to the left, we never gets in a loop, which guarantees termination because there are only finitely many cells.

For an example, see Figure 4 where the procedure is applied to find a path from the cell with a red cross to the boundary. □

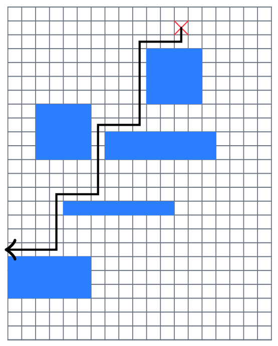


Figure 4: Example of a path from an uncovered cell to the boundary that does not use any covered cell.

4 Geometric Tools

We make use of the fundamental lemma in [7], which we copy below.

Lemma 18 *Let S_a be a unit square used in the packing, and S_b either the boundary of the shape being packed or another unit square. If the distance from S_a to S_b is at most 1, then there is an open disk of radius 2 containing S_a such that the area of wasted space in the open disk is at least $c \cdot \theta(S_a, S_b)$.*

In short:

Difference in angle leads to wasted space.

Lemma 18 only works for squares that are sufficiently close, however. We want to generalize it as follows.

Proposition 19 *Let S_a and S_b be squares, $\gamma\colon [0,l] \to \mathbb{R}^2$ be a path on the plane connecting S_a and S_b. Then there is a constant $r > 0$ and $c' > 0$ such that $W(B(\operatorname{im}\gamma, r)) \ge c' \cdot \theta(S_a, S_b)$.*

Recall $W(-)$ is the wasted area as defined in Definition 6.

Intuitively, we can imagine drawing a path γ from a to b, then the statement says that if there is an angle difference $\theta(S_a, S_b)$ between S_a and S_b, the total wasted area inside the region colored green is at least proportional to $\theta(S_a, S_b)$. See Figure 5 for an illustration.

Observation 1 *The vertical strip T_i in [7, Lemma 6] or the vertical line segment $L(X)$ in [7, Lemma 4] is very similar to a vertical path γ as we use here. Our proof can be seen as a generalization of the argument there.*

The main idea is the following. Lemma 18 gives us *circles* that contains large wasted area. Then we find a sequence of squares $(S_{j_0} = S_a, S_{j_1}, S_{j_2}, \dots, S_{j_{k-1}}, S_{j_k} =$

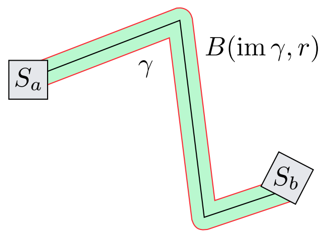


Figure 5: Illustration for Proposition 19 (drawing not to scale, $r = 5$ in this article).

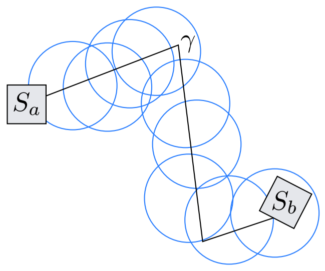


Figure 6: Illustration for the construction of circles.

S_b) along and near the path, apply Lemma 18 on each consecutive pair $(S_{j_i}, S_{j_{i+1}})$, and get a collection of circles whose centers are roughly along the path γ. See Figure 6 for an illustration.

To prove the total wasted area in the union of the circles is sufficiently large, we also need that each point is contained in $O(1)$ circles.

Proof. If there exists any $0 \le t \le l$ such that $W(B(\gamma(t), \frac{1}{4})) \ge c \cdot \theta(S_a, S_b)$, we are done.

Suppose otherwise. Construct a sequence of real numbers $\{t_i\}$ as follows: let $t_1 = 0$, and for each integer $i \ge 2$, let t_i be the largest value of t such that $t_i \ge t_{i-1}$ and $d(\gamma(t_{i-1}), \gamma(t_i)) \le \frac{1}{2}$. Because the domain of γ is compact and γ is continuous, this exists.

Let n be the smallest index such that $t_n = l$, if such index exists. Otherwise let $n = \infty$.

For each $1 \le i \le n$, define point $p_i = \gamma(t_i)$. For all $1 \le j < k < n$, $d(p_j, p_k) \ge \frac{1}{2}$. Since $\operatorname{im}\gamma$ is compact, it is bounded. We see $n < \infty$ by an application of the pigeonhole principle: say $\operatorname{im}\gamma$ is inside a square of side length R, divide it into $\lceil 4R \rceil^2$ small squares of side length $\le \frac{1}{4}$, then no two points (p_j, p_k) for $1 \le j < k < n$ can be in the same square, therefore $n \le \lceil 4R \rceil^2 + 1$.

For each $1 \le j \le n$, define index $i_j \ge 0$ such that $d(p_j, S^*_{i_j}) < \frac{1}{4}$. If there are multiple such indices, pick any; with the exception of $i_1 = a$ and $i_n = b$.

We show such i_j exists: Since we assume for all $0 \le t \le l$ then $W(B(\gamma(t), \frac{1}{4})) < c \cdot \theta(S_a, S_b) < \frac{1}{4}^2\pi$, there is at least one square with a nonzero intersection area with $B(\gamma(t), \frac{1}{4})$.

Next, for each $1 \le j < n$,

$$\begin{aligned} d(S^*_{i_j}, S^*_{i_{j+1}}) &\le d(S^*_{i_j}, p_j) + d(p_j, p_{j+1}) \\ &\quad + d(p_{j+1}, S^*_{i_{j+1}}) \\ &< \frac{1}{4} + \frac{1}{2} + \frac{1}{4} \\ &= 1. \end{aligned}$$

Therefore $d(S_{i_j}, S_{i_{j+1}}) < 1$.

If $\theta(S_{i_j}, S_{i_{j+1}}) = 0$ then $W(B(p_j, 5)) \ge 0 = c \cdot \theta(S_{i_j}, S_{i_{j+1}})$. Otherwise, at least one of S_{i_j} or $S_{i_{j+1}}$ is an unit square, applying Lemma 18, we find that there is an open disk with radius 2 containing either S_{i_j} or $S_{i_{j+1}}$ and have wasted area $\ge c \cdot \theta(S_{i_j}, S_{i_{j+1}})$. Let p be the center of this disk, so the disk is $B(p, 2)$, and let $k \in \{i_j, i_{j+1}\}$ be the index such that $S_k \subseteq B(p, 2)$. Then $d(p_j, S_k) < \frac{3}{4}$, so $d(p_j, p) \le 2 + \frac{3}{4}$, so $B(p, 2) \subseteq B(p_j, 5)$, therefore $W(B(p_j, 5)) \ge c \cdot \theta(S_{i_j}, S_{i_{j+1}})$.

By summing over all j, we get

$$\begin{aligned} \sum_{j=1}^{n-1} W(B(p_j, 5)) &\ge c \cdot \sum_{j=1}^{n-1} \theta(S_{i_j}, S_{i_{j+1}}) \\ &\ge c \cdot \theta(S_{i_1}, S_{i_n}) = c \cdot \theta(S_a, S_b). \end{aligned}$$

For each $1 \le j < k < n$, $d(p_j, p_k) \ge \frac{1}{2}$. Also, for each point p on the plane, $B(p, 5)$ is bounded and can be covered by finitely many squares (say, 1600) of side length $\frac{1}{4}$, each such square can contain at most one point p_j. Therefore,

$$W\left(\bigcup_{j=1}^{n-1} B(p_j, 5)\right) \ge \frac{c}{1600} \cdot \theta(S_a, S_b).$$

The value 1600 can be obtained by embedding $B(p, 5)$ into a closed square with side length 10, then cover it with 40×40 closed squares with side length $\frac{1}{4}$ each.

This gives the desired conclusion. □

We believe this proposition can be generalized to work with arbitrarily small $r > 0$, at a multiplicative cost of r^2 in c'.

5 Main Results

Now we return to the square packing problem. Recall x is the side length of the large square S_0.

Proposition 20 *There exists a constant $c_2 > 1$ independent of x such that: for all packing $\mathcal{A}$, there exists a good packing $\mathcal{A}'$ such that $W(\mathcal{A}') \le c_2 W(\mathcal{A})$.*

Proof. If $x \le 1$, the statement is obvious. We assume $x > 1$.

Divide the large square into $\lceil x \rceil$ rows and that many columns, we get a $\lceil x \rceil \times \lceil x \rceil$ grid, each cell is a square with side length $> \frac{1}{2}$ and ≤ 1.

Let the set of marked cells M be the set of cells that has any (positive area) overlap with a bad square. Apply Proposition 17 on this set M, we get an integer $f \geq 0$, and a collection of paths $\{P_i\}_{1\leq i\leq f}$, where $P_i = (P_{i,1}, \dots, P_{i,k_i})$. Let $O_{i,j}$ be the center of cell $P_{i,j}$, and let path γ_i be the polyline consisting of the shortest segment from any bad square that has an overlap with cell $P_{i,1}$ to $O_{i,1}$, followed by the polyline $O_{i,1}O_{i,2}O_{i,3}\dots O_{i,k_i}$, followed by the shortest segment from O_{i,k_i} to the boundary of S_0.

Apply Proposition 19 on each of the paths γ_i, we get (where c' and r are as in Proposition 19)

$$\sum_{i=1}^{f} W(B(\operatorname{im}\gamma_i, r)) \geq c' \cdot c \cdot f.$$

Because each cell $P_{i,j}$ is only used once and the side length of each cell is between $\frac{1}{2}$ and 1, each point in $\bigcup_{i=1}^{f} B(\operatorname{im}\gamma_i, r)$ is covered by $\Theta(1)$ of the $B(\operatorname{im}\gamma_i, r)$, therefore

$$\begin{aligned} W(\mathcal{A}) &\geq W\Big(\bigcup_{i=1}^{f} B(\operatorname{im}\gamma_i, r)\Big) \\ &\geq \frac{1}{\Theta(1)} \sum_{i=1}^{f} W(B(\operatorname{im}\gamma_i, r)) \\ &\geq \frac{c' \cdot c \cdot f}{\Theta(1)}. \end{aligned}$$

From the application of Proposition 17 above, we also get a collection of rectangles $\{R_1, \dots, R_k\}$ that contains all the marked cells. Use these rectangles, modify $\mathcal{A}$ into $\mathcal{A}'$ as follows. First, delete all unit squares completely contained in $\bigcup_i R_i$, this way all bad squares are deleted. Then, pack as many axis-aligned unit squares with integral vertex coordinates as possible.

Note that this packing procedure can only create wasted space in a rectangular ring of width $\sqrt{2}+1$ along the perimeter of R_i. See Figure 7 for an illustration.

To see this formally, we note the following. To repack the rectangle R_i (the outer, red rectangle) with width w and height h, note that the diameter of a unit square is $\sqrt{2}$, therefore the smaller rectangle with the same center as R_i, width $w - 2\sqrt{2}$ and height $h - 2\sqrt{2}$ does not contain any undeleted unit squares. Because axis-aligned unit squares are packed in, the entirety of this smaller rectangle except possibly a rectangular ring of width 1 and height 1 will be fully packed. Therefore, the additional waste is $\leq 2(\sqrt{2}+1)(w+h)$. If $\min(w,h) \leq 2(\sqrt{2}+1)$, the bound can be seen to hold as well.

Because the side length of each cell is between $\frac{1}{2}$ and 1, for each rectangle R_i, its perimeter is between $\frac{1}{2}$ and 1 times its cellular perimeter. We know from Proposition 17 that the total cellular perimeter of R_i is $\in O(f)$, so the total perimeter of R_i is also $\in O(f)$.

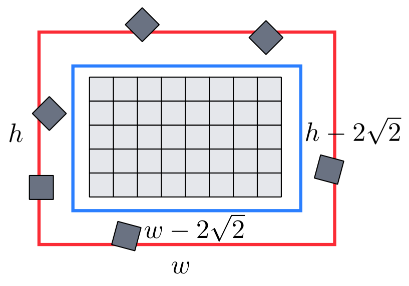


Figure 7: Demonstration of repacking the interior of a rectangle with axis-aligned unit squares with integral coordinates.

Therefore the additional wasted area introduced by the modification procedure is $\in O(f)$, so $W(\mathcal{A}') \leq W(\mathcal{A}) + O(f)$. Since $W(\mathcal{A}) \in \Omega(f)$, we get the desired result. □

Now we can prove Theorem 1.

Proof. As mentioned after Definition 12, $W(x) \leq W^*(x)$. Proposition 20 implies $W^*(x) \leq c_2 W(x)$. □

6 Discussion

6.1 Strengthening the Result

We have shown that only good squares (i.e. squares with tilt bounded by a constant) need to be considered to compute the asymptotic growth of the wasted area. Nonetheless, the tilt in the proposed packing methods [2, 3, 8] is in fact much smaller, $O(x^{-\epsilon})$ for some $\epsilon > 0$. Can the bound on the tilt be sharper, for example $O(1/\log x)$ or even $O(x^{-\epsilon})$ for some $\epsilon > 0$?

6.2 Related Algorithmic Considerations

An algorithm to find disjoint paths in Proposition 17 that is simpler than maximum flow is performing multi-source BFS from marked cells to the boundary, greedily pick the shortest path found each time, avoiding cells that already belong to a path. Let f' be the number of disjoint paths found by this method, then $f' \leq f$. While this method does not provide any guarantee of optimality (it may happen that there is no collection of rectangles with total perimeter $\leq 4f'$ covering all the marked cells, see Section 7.1), numerical experiments suggest the conjecture $f' \in \Omega(f)$. We find this likely because of the special structure of the grid.

In implementing the merging procedure described in Lemma 16, because the perimeter of each rectangle is at most $2(n+m)$, and the perimeter increases by at least 2 after a merge step with another rectangle that is not completely contained inside, the merging procedure terminates after at most $n+m$ iterations. Each iteration can be implemented in $O(nm)$, therefore the total time complexity is $O(nm(n+m))$. It would be interesting to investigate whether there is any algorithm taking less than cubic time to find the rectangle collection.

7 Conclusion

We prove a relation between arbitrary packing and good packing of a large square, therefore anyone proving a lower bound on $W(x)$ in the future similar to [7] only needs to consider good packings.

Acknowledgements

The author would like to thank some friends and Codeforces users for their insightful discussion and for identifying some errors.

References

[1] J. W. Alexander. A proof of the invariance of certain constants of analysis situs. *Transactions of the American Mathematical Society*, 16(2):148–154, 1915.

[2] F. Chung and R. Graham. Packing equal squares into a large square. *Journal of Combinatorial Theory, Series A*, 116(6):1167–1175, Aug. 2009.

[3] P. Erdös and R. Graham. On packing squares with equal squares. *Journal of Combinatorial Theory, Series A*, 19(1):119–123, 1975.

[4] S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4(4):507–518, Dec. 1975.

[5] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[6] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Dec. 1963.

[7] K. Roth and R. Vaughan. Inefficiency in packing squares with unit squares. *Journal of Combinatorial Theory, Series A*, 24, 3 1978.

[8] S. Wang, T. Dong, and J. Li. A new result on packing unit squares into a large square, 2016.

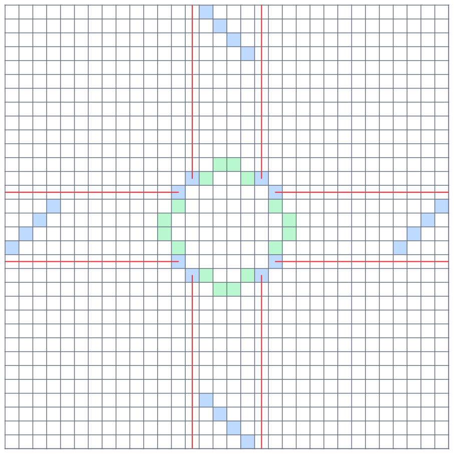


Figure 8: Example grid where greedy multi-source BFS algorithm gives f' disjoint paths and $4f'$ is less than the minimum total perimeter of covering rectangles.

Appendix

7.1 Example Grid where Greedy Multi-source BFS Gives Suboptimal Result

Consider the grid depicted in Figure 8, where the marked cells are colored green or blue. The multi-source BFS algorithm would first greedily find the paths to the blue cells, then it would not be able to find any additional paths to the green cells. However, if the 8 paths marked red are removed to create paths to the green cells first, more paths can be drawn. It can be seen that in this case greedy BFS algorithm finds $f' = 24$ disjoint paths, but there is no collection of rectangles with total perimeter $4f' = 96$ covering all the marked cells.

The Kinetic Hourglass Data Structure for Computing the Bottleneck Distance of Dynamic Data

Elizabeth Munch* Elena Xinyi Wang† Carola Wenk‡

Abstract

The kinetic data structure (KDS) framework is a powerful tool for maintaining various geometric configurations of continuously moving objects. In this work, we introduce the kinetic *hourglass*, a novel KDS designed to compute the bottleneck distance for geometric matching problems. We detail the events and updates required for handling general graphs, accompanied by a complexity analysis. Furthermore, we demonstrate the utility of the kinetic hourglass by applying it to compute the bottleneck distance between two persistent homology transforms (PHTs) derived from shapes in $\mathbb{R}^2$, which are topological summaries obtained by computing persistent homology from every direction in $\mathbb{S}^1$.

1 Introduction

Motion is a fundamental property of the physical world. To address this challenge computationally, Basch et al. proposed the *kinetic data structure* (KDS) framework [6], designed to maintain various geometric configurations for continuously moving objects. The KDS framework has been applied to many geometric problems since it was introduced, including but not limited to finding the convex hull of a set of moving points in the plane [5], the closest pair of such a set [5], a point in the center region [1], kinetic medians and *kd*-trees [2], and range searching; see [19] for a survey. In this work, we extend the framework to the geometric matching problem. Specifically, we are interested in the min-cost matching of a weighted graph with continuously changing weights on the edges. The result is that the bottleneck cost can be updated, rather than recomputing the needed matching from scratch every time.

The static bottleneck distance between persistence diagrams has been studied extensively, both theoretically and practically [7, 14, 16, 22, 23]. Since the bottleneck distance between persistence diagrams can be formulated as a bipartite graph matching problem [15], the work in this abstract can be used to improve bottleneck distance computation for a vineyard [10, 24, 28], more generally in the case of the Persistent Homology Transform [11, 17, 26], or even more generally for Persistent Homology Bundles [4, 20, 21].

*Department of Mathematics, Department of Computational Mathematics, Science, and Engineering, Michigan State University`muncheli@msu.edu`

†Institute of Geometry, Graz University of Technology, `x.wang@tugraz.at`

‡Department of Computer Science, Tulane University, `cwenk@tulane.edu`

2 Background

Broadly, we are interested in the following geometric matching problem. Given an undirected graph $G = (V, E)$, a *matching* M of G is a subset of the edges E such that no vertex in V is incident to more than one edge in M. A vertex v is said to be *matched* if there is an edge $e \in M$ that is incident to v. A matching is *maximal* if it is not properly contained in any other matching. A *maximum* matching M is a matching with the largest cardinality; i.e., for any other matching M', $|M| \geq |M'|$. A maximum matching is always maximal; the converse is not true.

For a bipartite graph $G = (X \sqcup Y, E)$ where $|X| = |Y| = n$ and $|E| = m$, a maximum matching is a *perfect* matching if every $v \in X \sqcup Y$ is matched, and $|M| = n$. This can be expressed as a bijection $\eta : X \to Y$. For a subset $W \subseteq X$, let $N(W)$ denote the *neighborhood* of W in G, the set of vertices in Y that are adjacent to at least one vertex of W. Hall's marriage theorem provides a necessary condition for a bipartite graph to have a perfect matching.

Theorem 1 (Hall's Marriage Theorem) *A bipartite graph $G = (X \sqcup Y, E)$ has a perfect matching if and only if for every subset W of X: $|W| \leq |N(W)|$.*

Building on Hall's Marriage Theorem, we extend our focus to weighted bipartite graphs. Given such a graph, a fundamental optimization problem is to identify matchings that minimize the maximum edge weight, known as the *bottleneck cost*.

Definition 2 *A weighted graph $\mathcal{G} = (G, c)$ is a graph G together with a weight function $c : E \to \mathbb{R}_+$. The* bottleneck cost *of a matching M for such a $\mathcal{G}$ is $\max\{c(e) \mid e \in M\}$. The* bottleneck edge *is the highest weighted edge in M, assuming this is unique. A perfect matching is* optimal *if its cost is minimal among all perfect matchings. An optimal matching is also called a* min-cost *matching.*

To find a maximum matching of a graph, we use augmenting paths.

Definition 3 *For a graph G and matching M, a path P is an* augmenting path *for M if:*

1. *the two end points of P are unmatched in M, and*
2. *the edges of P alternate between edges $e \in M$ and $e \notin M$.*

Theorem 4 (Berge's Theorem) *A matching M in a graph G is a maximum matching if and only if G contains no M-augmenting path.*

The existing algorithms that compute the bottleneck cost are derived from the Hopcroft-Karp maximum matching algorithm, which we briefly review [22]. Given a graph $G = (X \sqcup Y, E)$ where $|E| = n$, and an initial matching M, the algorithm iteratively searches for augmenting paths P. Each phase begins with a breadth-first search from all unmatched vertices in X, creating a layered subgraph of G by alternating between $e \in M$ and $e \notin M$. It stops when an unmatched vertex in Y is reached. From this layered graph, we can find a maximal set of vertex-disjoint augmenting paths of the shortest length. For each P, we augment M by replacing $M \cap P$ with $P \backslash M$. We denote this process as $\text{Aug}(M, P) = M \setminus (M \cap P) \cup (P \setminus M)$. Note that $|\text{Aug}(M, P)|$ is a matching with $|\text{Aug}(M, P)| = |M|+1$, so we can repeat the above process until no more augmenting paths can be found. By Theorem 4, the resulting M is maximum. The algorithm terminates in $O(\sqrt{n})$ rounds, resulting in a total running time of $O(n^{2.5})$. This algorithm was later improved to $O(n^{1.5} \log n)$ for geometric graphs by Efrat et al. by constructing the layered graph via a near-neighbor search data structure [16].

2.1 Bottleneck Distance

Let X and Y be two sets of n points. We consider the bipartite graph obtained by adding edges between points whose weight $\text{c} : E \to \mathbb{R}_{\geq 0}$ is at most λ:

$$G_\lambda = (X \sqcup Y, \{xy \mid \text{c}(xy) \leq \lambda\}).$$

Definition 5 *The* optimal bottleneck cost *is the minimum λ such that G_λ has a perfect matching, denoted as $d_B(G)$.*

We are interested in computing the bottleneck distance in the context of *persistent homology*. Persistent homology is a multi-scale summary of the connectivity of objects in a nested sequence of subspaces; see [13] for an introduction. For the purposes of this section, we can define a *persistence diagram* to be a finite collection of points $\{(b_i, d_i)\}_i$ with $d_i \geq b_i$ for all i. Further details and connections to the input data will be given in Section 4.

Given two persistence diagrams X and Y, a partial matching is a bijection $\eta : X' \to Y'$ on a subset of the points $X' \subseteq X$ and $Y' \subseteq Y$; we denote this by $\eta : X \rightleftharpoons Y$. The cost of a partial matching is the maximum over the L_∞-norms of all pairs of matched points and the distance between the unmatched points to the diagonal:

$$\begin{aligned} c(\eta) = \max \big(&\{\|x - \eta(x)\|_\infty \mid x \in X'\} \\ &\cup \left\{ \tfrac{1}{2}|z_2 - z_1| \mid (z_1, z_2) \in (X \setminus X') \cup (Y \setminus Y') \right\} \big) \end{aligned}$$

and the bottleneck distance is defined as $d_B(X, Y) = \inf_{\eta:X \rightleftharpoons Y} c(\eta)$.

We now reduce finding the bottleneck distance between persistence diagrams to a problem of finding the bottleneck cost of a bipartite graph. Let X and Y be two persistence diagrams given as finite lists of off-diagonal points. For any off-diagonal point $z = (z_1, z_2)$, the orthogonal projection to the diagonal is $z' = ((z_1+z_2)/2, (z_1+z_2)/2)$. Let $\overline{X}$ (resp. $\overline{Y}$) be the set of orthogonal projections of the points in X (resp. Y). Set $U = X \sqcup \overline{Y}$ and $V = Y \sqcup \overline{X}$. We define the complete bipartite graph $G = (U \sqcup V, U \times V, \text{c})$, where for $u \in U$ and $v \in V$, the weight function c is given by

$$\text{c}(uv) = \begin{cases} \|u - v\|_\infty & \text{if } u \in X \text{ or } v \in Y \\ 0 & \text{if } u \in \overline{X} \text{ and } v \in \overline{Y}. \end{cases}$$

An example of the bipartite graph construction is shown in Figure 1. This graph can be used to compute the bottleneck distance of the input diagrams because of the following lemma.

Lemma 6 (Reduction Lemma [15]) *For the above construction of G, $d_B(G) = d_B(X, Y)$.*

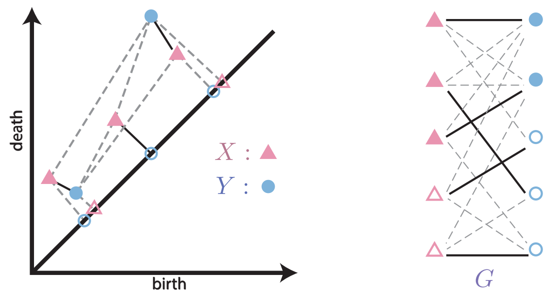


Figure 1: Construction of the bipartite graph G based on the persistence diagrams X and Y.

Naively, given a graph G of size n, we can compute the bottleneck distance by sorting the edge weights and performing a binary search for the smallest λ such that G_λ has a perfect matching in $O(n^2 \log n)$ time, which dominates the improved Hopcroft-Karp algorithm. Using the technique for efficient k-th distance selection for a bi-chromatic point set under the L_∞ distance introduced by Chew and Kedem, this can be reduced to $O(n^{1.5}\log n)$ [8]. Therefore, the overall complexity to compute the bottleneck distance of a static pair of persistence diagrams is $O(n^{1.5}\log n)$.

2.2 Kinetic Data Structure

A kinetic data structure (KDS) [5, 18, 19] maintains a system of objects v that move along a known continuous *flight plan* as a function of time, denoted as $v = f(v)$. *Certificates* are conditions under which the data structure is accurate, and *events* track when the certificate fails and what corresponding *updates* need to be made. The certificates of the KDS form a proof of correctness of the current configuration function at all times. Updates are stored in a priority queue, keyed by event time. To advance to time t, we process all the updates keyed at times before t and pop them from the queue after updating. We continue until t is smaller than the first event time in the queue. Finally, we distinguish between *external events*, i.e., those affecting the configuration function (e.g., maximum or minimum values) we are maintaining, and *internal events*, i.e., those that do not affect the outcome.

The kinetization of a heap results in a *kinetic heap*, which maintains the priority of a set of objects as they change continuously. Maximum and minimum are both examples of priorities. A kinetic heap follows the properties of a static heap such that objects are stored as a balanced tree. The value of a node is stored as a function of time $f_X(t)$. The data structure augments a certificate for every pair of parent-child nodes A and B, which is only valid when $f_A(t) > f_B(t)$ (resp. $f_A(t) < f_B(t)$) for a max (resp. min) heap. Thus, the failure times of the certificate are the times t such that $f_A(t) = f_B(t)$, and updates are done by swapping A and B in the heap while making all relevant parent-child updates, see Figure 2. The kinetic max heap supports operations similar to a static heap, such as create-heap, find-max, insert, and delete. Insertion and deletion are done in $O(\log n)$ time.

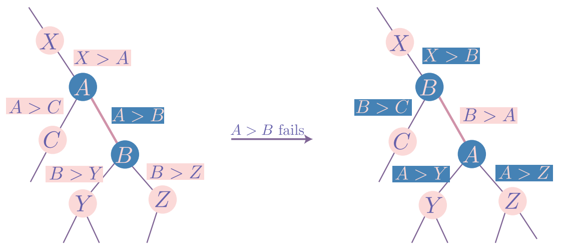


Figure 2: Kinetic heap event updates

More efficient solutions to this problem have been developed, such as the *kinetic hanger*, introduced by da Fonseca et al. in [12]. It modifies the kinetic heap by incorporating randomization to balance the tree, meaning all complexity results are expected rather than deterministic. As shown in [12], the kinetic hanger has $O(1)$ locality. The number of expected events in the affine motion case is $O(n\log n)$ and $O(\lambda_s(n)\log n)$ for n s-intersecting curves. The expected runtime is obtained by multiplying by $O(\log n)$ for each event. We use $\alpha(\cdot)$ to denote the inverse Ackerman function and λ_s is the length bound for the Davenport-Schinzel Sequence; see [3] for details.

3 Kinetic Hourglass

In this section, we introduce a new kinetic data structure that keeps track of the optimal bottleneck cost of a weighted graph $\mathcal{G} = (G, \mathrm{c})$ where c changes continuously with respect to time t. The *kinetic hourglass* data structure is composed of two kinetic heaps; in Section 3.3 we will give the details for replacing these with kinetic hangers. One heap maintains minimum priority, and the other maintains maximum. Assume we are given a connected bipartite graph $G = (V, E)$ with the vertex set $V = X \sqcup Y$, where $|X| = |Y| = n$; and edge set E, where $|E| = m$. If G is a complete bipartite graph, then $m = n^2$. The weight of the edges at time t is given by $\mathrm{c}^t : E \to \mathbb{R}_{\geq 0}$. Denote the weighted bipartite graph by $\mathcal{G}^t = (G, \mathrm{c}^t)$. The weights of those m edges are the objects we keep track of in our *kinetic hourglass*. We assume that these weights, called flight plans in the kinetic data structure setting, are given for all times $t \in [0, T]$.

Let $G^t_\delta \subseteq G$ be the portion of the complete bipartite graph with all edges with weight at most δ at time t; i.e., $V(G^t_\delta) = V$, and $E(G^t_\delta) = \{e \in E \mid \mathrm{c}^t(e) \leq \delta\}$. If the bottleneck distance $\hat{\delta}^t = d_B(\mathcal{G}^t)$ is known, we are focused on the bipartite graph $G^t_{\hat{\delta}}$, which we will denote by $\hat{G}^t$ for brevity. By definition, we know that there is a perfect matching in $\hat{G}^t$, which we denote as M^t, although we note that this is not unique. Further, there is an edge $\hat{e}^t \in M^t$ with $\mathrm{c}(\hat{e}^t) = \hat{\delta}^t$, which we call the *bottleneck edge*. This edge is unique as long as all edges have unique weights, except for when events happen. We separate the remaining edges into the sets

$$L^t = E(\hat{G}^t) \setminus E(M^t), \text{ and}$$
$$U^t = E \setminus E(\hat{G}^t) = \{e \in E \mid \mathrm{c}^t(e) > \hat{\delta}^t\}$$

so that $E = L^t \sqcup M^t \sqcup U^t$.

The kinetic hourglass consists of the following kinetic max heap and kinetic min heap. The *lower heap* $\mathcal{H}_L$ is the max heap containing $L^t \sqcup M^t = E(\hat{G}^t)$. The *upper heap* $\mathcal{H}_U$ is the min heap containing $U^t \cup \{\hat{e}\}$. See Figure 3 for an example of this construction. Note that $\mathrm{c}^t(\hat{e}) = \max\{\mathrm{c}^t(e) \mid e \in L^t \sqcup M^t\}$ and $\mathrm{c}^t(\hat{e}) <$

Scenario	Kinetic Heap	Kinetic Hanger
Lines	$O(n \log^2 n)$	$O(n \log^2 n)$
Line segments	$O(n\sqrt{m} \log^{3/2} m)$	$O(n\alpha(m) \log^2 m)$
s-intersecting curves	$O(n^2 \log^2 n)$	$O(\lambda_s(n) \log^2 n)$
s-intersecting curve segments	$O(mn \log^2 m)$	$O(n/m\lambda_{s+2}(m) \log^2 m)$

Table 1: Deterministic complexity of the kinetic heap and expected complexity of the kinetic hanger. Here, n is the total number of elements, m denotes the maximum number of elements stored at a given time, s is a constant.

$\min\{c^t(e) \mid e \in U^t\}$, meaning that $\hat{e}$ is the root for both heaps.

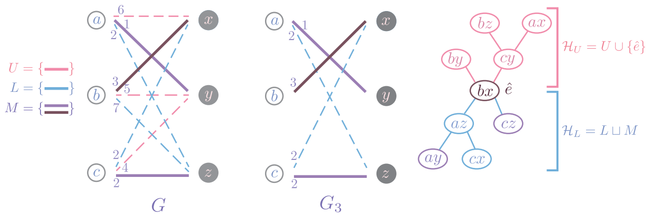


Figure 3: Illustration of construction of the kinetic hourglass.

3.1 Certificates

The certificates for the kinetic hourglass (i.e., properties held by the data structure for all $t \in [0, T]$) are

1. All max-heap certificates for $\mathcal{H}_L$ and min-heap certificates for $\mathcal{H}_U$.
2. Both heaps have the same root, denoted r^t.
3. The edge r^t is the edge with bottleneck cost; i.e., $r^t = \hat{e}^t$ where $c^t(\hat{e}^t) = \hat{\delta}^t$.

Assuming the certificates are maintained, r^t and $\hat{e}^t$ are the same edge. However, in the course of proofs, bottleneck edge of the matching is denoted by $\hat{e}^t$, while we use r^t for the edge stored in the root of the two heaps (or $r^t(\mathcal{H}_U)$ and $r^t(\mathcal{H}_L)$ when a distinction is needed).

3.2 Events

For a particular event time t, we denote the moment of time just before an event by $t^- = t - \varepsilon$ and the moment of time just after by $t^+ = t + \varepsilon$. For two edges, we write $a \preccurlyeq_t b$ to mean that $c^t(a) \leq c^t(b)$. The two heaps have their own certificates, events (both internal and external), and updates as in the standard setting. Our main task of this section is to determine which events in the heaps lead to an external event in the hourglass. We define the external events for the hourglass as those events in $\mathcal{H}_U$ and $\mathcal{H}_L$ which lead to changes of the root, r^t. Thus, internal events are those which do not affect the roots; i.e., $r^{t^-}(\mathcal{H}_U) = r^{t^+}(\mathcal{H}_U)$ and $r^{t^-}(\mathcal{H}_L) = r^{t^+}(\mathcal{H}_L)$.

We first show that an internal event of $\mathcal{H}_U$ or $\mathcal{H}_L$ is an internal event of the hourglass.

Lemma 7 *If the event at time t is an internal event of $\mathcal{H}_U$ or $\mathcal{H}_L$ and the kinetic hourglass satisfies all certificates at time t^-, then the edge giving the bottleneck distance for times t^- and t^+ is the same; that is, $\hat{e}^{t^-} = \hat{e}^{t^+}$.*

Proof. Following the previous notation, we have bottleneck distances before and after given by $\hat{\delta}^{t^-} = c^{t^-}(\hat{e}^{t^-})$ and $\hat{\delta}^{t^+} = c^{t^+}(\hat{e}^{t^+})$. Because we start with a correct hourglass, we know that $\hat{e}^{t^-} = r^{t^-}(\mathcal{H}_U) = r^{t^-}(\mathcal{H}_L) = r^{t^-}$. By definition, an internal event in either heap is a swap of two elements with a parent-child relationship but for which neither is the root so the roots remain unchanged; that is, $r^{t^-} = r^{t^+}$ so we denote it by r for brevity. This additionally means that no edge moves from one heap to the other, so the set of elements in each heap does not change and thus $G_{c^{t^-}(r)} = G_{c^{t^+}(r)}$. Again for brevity, we write this subgraph as Γ.

We need to show that this edge r is the one giving the bottleneck distance at t^+, i.e. $\hat{\delta}^{t^+} = c^{t^+}(r)$ or equivalently that $r = \hat{e}^{t^+}$. All edges of the perfect matching from t^-, M^{t^-}, are contained in Γ; thus M^{t^-} is still a perfect matching at time t^+. We show that any minimal cost perfect matching for t^+ must contain r, and thus M^{t^+} is a *minimal* cost perfect matching. Since r is $\hat{e}^{t^-}$, by removing r, $\Gamma \backslash \{r\}$ will cease to have a perfect matching, else contracting the minimality of the perfect matching at t^-. But as the order is unchanged, this further means that for time t^+, lowering the threshold for the subgraph $\Gamma = G_{c^{t^+}(r)}$ or equivalently removing the edge r will not have a perfect matching, finishing the proof. $\square$

The remaining cases to consider are external events of $\mathcal{H}_U$ and $\mathcal{H}_L$, when a certificate in one of the heaps involving the root fails. This can be summarized in the three cases below. In each case, denote the root at time t^- as $r = r^{t^-}$.

1. (L-Event) Swap priority of r and an $e \in L^t$ in $\mathcal{H}_L$; i.e. $e \preccurlyeq_{t^-} r$ and $r \preccurlyeq_{t^+} e$.
2. (M-Event) Swap priority of r and an $e \in M^t$ in $\mathcal{H}_L$; i.e. $e \preccurlyeq_{t^-} r$ and $r \preccurlyeq_{t^+} e$.

3. (U-Event) Swap priority of r and an $e \in U^t$ in $\mathcal{H}_U$; i.e. $r \preccurlyeq_{t^-} e$ and $e \preccurlyeq_{t^+} r$.

In the remainder of this section, we consider each of those events and provide the necessary updates. The simplest update comes from the first event in the list, since we will show that no additional checks are needed.

Lemma 8 (L-Event) *Assume we swap priority of $r = r^{t^-}$ and an $e \in L^{t^-}$ at t; i.e. $e \preccurlyeq_{t^-} r$ and $r \preccurlyeq_{t^+} e$. Then e moves from $\mathcal{H}_L$ to $\mathcal{H}_U$, and r remains the root and is the edge with the bottleneck cost for t^+, i.e., $r^{t^+} = \hat{e}^{t^+}$.*

Proof. Denote $M = M^{t^-}$. In this case, e is an edge in the lower heap but $e \notin M$. Because $r \preccurlyeq_{t^+} e$, in order to maintain the heap certificates, either e needs to be inserted into the upper heap with r remaining as the root; or if e remains in the lower heap, it needs to become the new root. Note that although they swap orders, the graph thresholded at the cost of r at t^- and the graph thresholded at the cost of e at t^+ are the same; i.e. $G_{c^{t^-}(r)} = G_{c^{t^+}(e)}$. In addition, $G_{c^{t^+}(r)} = G_{c^{t^+}(e)} \setminus \{e\}$. However, since $e \notin M$, M is still a perfect matching in $G_{c^+(r)}$. If there exists a perfect matching in $G_{c^{t^+}(r)} \setminus \{r\}$, this would constitute a perfect matching at time t^- which has lower cost than M, contradicting the assumption that M is a minimal cost matching for that time. Thus M is a minimal cost matching for time t^+. $\square$

Lemma 9 (M-Event) *Assume we swap priority of $r = r^{t^-}$ and an $e \in M^{t^-}$ at t; i.e. $e \preccurlyeq_{t^+} r$ and $r \preccurlyeq_{t^-} e$. Let $G' = \hat{G}^{t^-} \setminus \{e\}$ be the graph with $e = (u, v)$ removed. Exactly one of the following scenarios happens. (See Figure 4 and 5 for examples of the two scenarios.)*

1. *There exists an augmenting path P in G' from u to v. Then e moves into the upper heap ($e \in U^{t^+}$), the root remains the same ($\hat{e}^{t^+} = r^{t^+} = r$), and the matching is updated with the augmenting path, specifically $M^{t^+} = \mathrm{Aug}(M^{t^-}, P)$.*

2. *There is no such augmenting path. Then $M^{t^+} = M^{t^-}$ and $\hat{e}^{t^+} = r^{t^+} = e$; i.e. the only update is that r and e switch places in the lower heap.*

Lemma 10 (U-Event) *Assume we swap priority of $r = r^{t^-}$ and an $e \in U^{t^-}$ at t; i.e. $r \preccurlyeq_{t^-} e$ and $e \preccurlyeq_{t^+} r$. Let $G' = \hat{G}^{t^-} \cup \{e\} \setminus \{r\}$ be the graph with $r = (u, v)$ removed and e included. Exactly one of the following events happens.*

1. *There exists an augmenting path P from u to v. Then r moves into the upper heap ($r \in U^{t^+}$), e becomes the root ($\hat{e}^{t^+} = r^{t^+} = e$), and the matching is updated with the augmenting path, specifically $M^{t^+} = \mathrm{Aug}(M^{t^-}, P)$.*

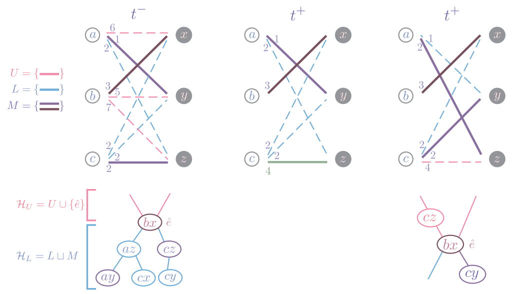


Figure 4: Illustration of Scenario 1 of and M-Event; see Lemma 9.

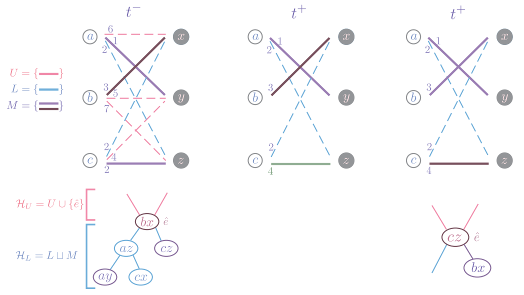


Figure 5: Illustration of Scenario 2 of M-Event; see Lemma 9.

2. *There is no such augmenting path. Then $M^{t^+} = M^{t^-}$ and e moves into the lower heap ($e \in L^{t^+}$), and the root remains the same ($\hat{e}^{t^+} = r^{t^+} = r$).*

We have included the proofs of the above lemmas in Appendix A and B. For further details, see [25]

3.3 Complexity and Performance Evaluation

The kinetic hourglass builds on the kinetic heap and hanger frameworks [12], with complexities derived by multiplying the number of events by the $O(\log n)$ event processing time (Table 1). In contrast, the hourglass requires $O(m)$ work per external event due to augmenting path searches, replacing the $\log n$ factor with m.

For a bipartite graph with m edges, we treat $m = n$ and analyze each edge's behavior only during its presence in a heap. This leads to complexities summarized in Table 2: for linear weights, $O(m^2\sqrt{m}\log^{3/2} m)$ using heaps and $O(m^2\alpha(m)\log m)$ with hangers; for s-intersecting curves, $O(m^3 \log m)$ and $O(m\lambda_{s+2}(m)\log m)$, respectively.

Despite being responsive, local, and compact, the hourglass is not efficient due to its linear per-event cost. We conjecture that amortized analysis may reduce this, though it remains an open problem.

Scenario	Kinetic Heap Hourglass	Kinetic Hanger Hourglass
Line segments	$O(m^{5/2} \log^{1/2} m)$	$O(m^2 \alpha(m) \log m)$
s-intersecting curve segments	$O(m^3 \log m)$	$O(m \lambda_{s+2}(m) \log m)$

Table 2: Deterministic complexity of the kinetic heap hourglass and expected complexity of the kinetic hanger hourglass.

4 Kinetic Hourglass for Persistent Homology Transform

We apply the kinetic hourglass to compute the integrated bottleneck distance between two 0-dimensional persistent homology transforms (PHTs) [11, 17, 26], a directional transform shape signature derived from sublevel set filtrations. For simplicity, we focus on connected, embedded graphs in $\mathbb{R}^2$ whose PHTs are composed of 0-dimensional persistence diagrams.

Each unit vector $\omega \in \mathbb{S}^1$ induces a height function h_ω on the vertices of a simplicial complex K, resulting in a lower-star filtration and a persistence diagram $\mathrm{Dgm}(h_\omega^K)$. The PHT is the function $\mathrm{PHT}(K) : \omega \mapsto \mathrm{Dgm}(h_\omega^K)$. The integrated bottleneck distance between two PHTs is

$$d_B(\mathrm{PHT}(K_1), \mathrm{PHT}(K_2)) = \int_0^{2\pi} d_B\big(\mathrm{Dgm}(h_\omega^{K_1}), \mathrm{Dgm}(h_\omega^{K_2})\big)\, d\omega$$

which is stable under small vertex perturbations.

To enable tractable runtime analysis, we restrict our attention to *star-shaped* complexes, where each persistence vineyard exhibits *trivial geometric monodromy* [4]. In this case, each off-diagonal persistence point traces a continuous curve (vine) over $\mathbb{S}^1$.

We construct a complete bipartite graph G whose vertices correspond to vines and their diagonal projections from the PHTs of K_1 and K_2. The edge weights are determined by ℓ^∞ distances between points on the vines at a given direction ω. This reduces the integrated bottleneck distance computation to maintaining a minimum-cost matching in G across all directions ω. Figure 6 gives an example of the pipeline.

Given at most $2n$ vines from each complex, G has $O(n^2)$ edges, leading to $O(n^6 \log n)$ runtime for the kinetic hourglass using heaps. See Appendix C for full definitions of the filtration, stability, and geometric interpretation of the PHT.

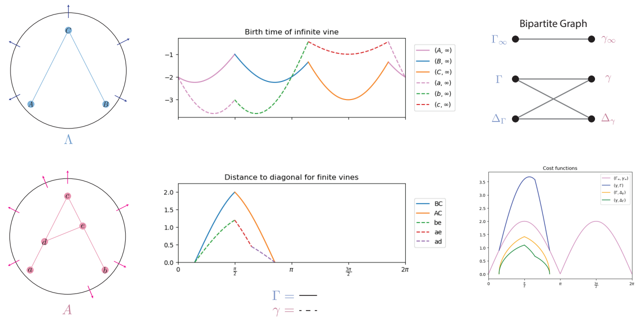


Figure 6: Example of edge weight functions in the bipartite graph induced by two PHTs.

5 Conclusions

In this paper, we introduced the *kinetic hourglass*, a novel kinetic data structure designed for maintaining the bottleneck distance for graphs with continuously changing edge weights. The structure incorporates two kinetic priority queues, which can be either the deterministic kinetic heap or the randomized kinetic hanger. Both versions are straightforward to implement.

In the future, we hope to improve the runtime for this data structure. In particular, the augmenting path search requires $O(n)$ time, falling short of the efficiency goals in the kinetic data structure (KDS) framework. Moreover, when comparing PHTs with n vertices, the kinetic hourglass holds n^2 elements, which can be computationally expensive. This method can immediately be extended to study the extended persistent homology transform (XPHT) to compare objects that have different underlying topologies [27], however there is much to be understood for the structure of the vines in the PHT and how this particular structure can deal with monodromy. Further, the nature of this data structure means that it is not immediately extendable to the Wasserstein distance case, however, we would like to build a modified version that will work for that case. Finally, since the kinetic hourglass data structure also has the potential to compare more general vineyards that extend beyond the PHT, it will be interesting in the future to find further applications.

Acknowledgment EXW acknowledges support from the Austrian Science Fund (FWF) grant number P 33765-N.

References

[1] P. K. Agarwal, M. De Berg, J. Gao, L. J. Guibas, and S. Har-Peled. Staying in the middle: Exact and approximate medians in R^1 and R^2 for moving points. In *CCCG*, pages 43–46, 2005.

[2] P. K. Agarwal, J. Gao, and L. J. Guibas. Kinetic medians and kd-trees. In R. Möhring and R. Raman, editors, *Algorithms — ESA 2002*, pages 5–17, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[3] P. K. Agarwal and M. Sharir. Davenport–schinzel sequences and their geometric applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 1–47. North-Holland, Amsterdam, 2000.

[4] S. Arya, B. Giunti, A. Hickok, L. Kanari, S. McGuire, and K. Turner. Decomposing the persistent homology transform of star-shaped objects, 2024.

[5] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.

[6] J. Basch, L. J. Guibas, and G. D. Ramkumar. Sweeping lines and line segments with a heap. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, SCG '97, page 469–471, New York, NY, USA, 1997. Association for Computing Machinery.

[7] S. Cabello, S.-W. Cheng, O. Cheong, and C. Knauer. Geometric Matching and Bottleneck Problems. In W. Mulzer and J. M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[8] L. P. Chew and K. Kedem. Improvements on geometric pattern matching problems. In O. Nurmi and E. Ukkonen, editors, *Algorithm Theory — SWAT '92*, pages 318–325, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[9] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, Jan 2007.

[10] D. Cohen-Steiner, H. Edelsbrunner, and D. Morozov. Vines and vineyards by updating persistence in linear time. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, SCG '06, page 119–126, New York, NY, USA, 2006. Association for Computing Machinery.

[11] J. Curry, S. Mukherjee, and K. Turner. How many directions determine a shape and other sufficiency results for two topological transforms. *Transactions of the American Mathematical Society, Series B*, 2018.

[12] G. D. da Fonseca, C. M. de Figueiredo, and P. C. Carvalho. Kinetic hanger. *Information Processing Letters*, 89(3):151–157, 2004.

[13] T. K. Dey and Y. Wang. *Computational Topology for Data Analysis.* Cambridge University Press, 2017.

[14] T. K. Dey and C. Xin. Computing Bottleneck Distance for 2-D Interval Decomposable Modules. In B. Speckmann and C. D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[15] H. Edelsbrunner and J. Harer. *Computational Topology - an Introduction.* American Mathematical Society, 2010.

[16] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, Sep 2001.

[17] R. Ghrist, R. Levanger, and H. Mai. Persistent homology and Euler integral transforms. *Journal of Applied and Computational Topology*, 2(1):55–60, Oct 2018.

[18] L. J. Guibas. Kinetic data structures: a state of the art report. In *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics: The Algorithmic Perspective: The Algorithmic Perspective*, WAFR '98, page 191–209, USA, 1998. A. K. Peters, Ltd.

[19] L. J. Guibas and M. Roeloffzen. Modeling motion. In C. D. Toth, J. O'Rourke, and J. E. Goodman, editors, *Handbook of Discrete and Computational Geometry (3rd ed.)*, chapter 57, pages 1117 – 1134. Chapman and Hall/CRC, 2017.

[20] A. Hickok. Computing persistence diagram bundles, 2022.

[21] A. Hickok. Persistence diagram bundles: A multidimensional generalization of vineyards, Oct. 2022.

[22] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, Dec 1973.

[23] M. Kerber, D. Morozov, and A. Nigmetov. Geometry helps to compare persistence diagrams. *ACM J. Exp. Algorithmics*, 22, Sep 2017.

[24] W. Kim and F. Mémoli. Spatiotemporal persistent homology for dynamic metric spaces. *Discrete & Computational Geometry*, 66(3):831–875, Oct 2021.

[25] E. Munch, E. X. Wang, and C. Wenk. The kinetic hourglass data structure for computing the bottleneck distance of dynamic data, 2025.

[26] K. Turner, S. Mukherjee, and D. M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, 12 2014.

[27] K. Turner, V. Robins, and J. Morgan. The extended persistent homology transform of manifolds with boundary. *Journal of Applied and Computational Topology*, May 2024.

[28] L. Xian, H. Adams, C. M. Topaz, and L. Ziegelmeier. Capturing dynamics of time-varying data via topology. *Foundations of Data Science*, 4(1):1–36, 2022.

Appendix

A Proof of Lemma 9

Proof. Let $M' = M^{t^-} \setminus \{e\}$. Then M' is a matching of size $n-1$ in G' (hence it is not perfect), and the unmatched vertices are u and v.

If there exists an augmenting path P from u to v, augment M' by replacing $M' \cap P$ with $P \setminus M'$ to make a new matching M''. This increases $|M'|$ by 1 and thus M'' is a perfect matching. Note that if $r \in P$, then M'' is also a perfect matching in t^- with strictly cheaper cost than M^{t^-}; but this contradicts the assumption of r giving the bottleneck distance. Thus we can assume that $r \in M''$, and all edges in the lower heap have cost at most that of r. This means that r is still the bottleneck edge of the matching, i.e. $\mathrm{c}^{t^+}(r) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M''\}$.

Assume instead that there is no augmenting path in G' from u to v. Then M' is a maximum matching for G' by Theorem 4, however it is not perfect. Note that because $G' = \hat{G}^{t^-} \setminus e$ and e and r have swapped places, we have that $G' = G_{\mathrm{c}^{t^+}(r)}$. Therefore, there is no perfect matching for $G_{\mathrm{c}^{t^+}(r)}$. However, M^{t^-} is a perfect matching at time t^- and $\hat{G}^{t^-} = G_{\mathrm{c}^{t^+}(e)}$, it still is a perfect matching at t^+ for $G_{\mathrm{c}^{t^+}(e)}$. Moreover, $\mathrm{c}^{t^+}(e) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M^{t^+}\}$. Therefore, $e = \hat{e}^{t^+} = r^{t^+}$ becomes the root, and r becomes a child of the root e in $\mathcal{H}_L$. □

B Proof of Lemma 10

Proof. Let $M' = M^{t^-} \setminus \{r\}$, and again M' is a matching of size $n-1$ in G' (hence it is not perfect), and the unmatched vertices are u and v.

Similar to the M-Event, if there exists an augmenting path P from u to v, augment M' by replacing $M' \cap P$ with $P \setminus M'$ to make a new matching M''. This increases $|M'|$ by 1; thus, M'' is a perfect matching. Further, because $G' = \hat{G}^{t^-} \cup \{e\} \setminus \{r\}$ and e and r have swapped places, we have that $G' = G_{\mathrm{c}^{t^+}(e)}$. Moreover, $\mathrm{c}^{t^+}(e) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M''\}$. Therefore $e = \hat{e}^{t^+} = r^{t^+}$, and r gets moved down to $\mathcal{H}_U$ and becomes a child of e.

Assume instead that there is no augmenting path in G' from u to v. Then M' is a maximum matching for G' by Theorem 4, however it is not perfect. Therefore, there is no perfect matching for $G_{\mathrm{c}^{t^+}(e)}$. However, M^{t^-} is a perfect matching at time t^- and $\hat{G}^{t^-} = G_{\mathrm{c}^{t^+}(r)} \setminus \{e\}$, it still is a perfect matching at t^+ for $G_{\mathrm{c}^{t^+}(r)}$. This is thus an internal event; we move e to $\mathcal{H}_L$ as a child of r, and r remains the bottleneck edge, i.e. $\mathrm{c}^{t^+}(r) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M''\}$. □

C Persistent Homology Transform

C.1 Lower-Star Filtration and PHT Definition

Let K be a finite simplicial complex embedded in $\mathbb{R}^2$. For any direction $\omega \in \mathbb{S}^1$, the height function $h_\omega : |K| \to \mathbb{R}$ is defined by $h_\omega(x) = \langle x, \omega \rangle$. The lower-star filtration induced on the abstract complex K is given by:

$$h_\omega(\sigma) = \max\{h_\omega(v) \mid v \in \sigma\},$$

and the sublevelset at height a is the full subcomplex induced by vertices $\{v \in K \mid h_\omega(v) \leq a\}$. Filtering K by h_ω produces a persistence diagram $\mathrm{Dgm}_k(h^K_\omega)$. We focus on $k = 0$, where the input can be assumed to be a connected graph, so each diagram contains a single point at infinity.

The persistent homology transform (PHT) is defined as:

$$\mathrm{PHT}(K) : \mathbb{S}^1 \to \mathcal{D}, \quad \omega \mapsto \mathrm{Dgm}(h^K_\omega),$$

where $\mathcal{D}$ is the space of persistence diagrams. The PHT is injective [11, 26], making it a faithful shape representation.

C.2 Stability of the Bottleneck Distance

Assume two geometric realizations $f_1, f_2 : K \to \mathbb{R}^2$ of the same abstract complex K. For a fixed ω, define $h_{i,\omega} = \langle f_i(\cdot), \omega \rangle$. Then:

$$\|h_{1,\omega} - h_{2,\omega}\|_\infty \leq \max_{v \in V} \|f_1(v) - f_2(v)\|_\infty.$$

By the stability theorem for persistence diagrams [9], we have:

$$d_B(\mathrm{Dgm}(h^{K_1}_\omega), \mathrm{Dgm}(h^{K_2}_\omega)) \leq \max_{v \in V} \|f_1(v) - f_2(v)\|_\infty,$$

and integrating over ω yields the global bound on the PHT distance:

$$d_B(\mathrm{PHT}(K_1), \mathrm{PHT}(K_2)) \leq 2\pi \max_{v \in V} \|f_1(v) - f_2(v)\|_\infty.$$

C.3 Point Tracking and Geometry of Diagrams

In a lower-star filtration, a k-dimensional homology class born at τ and dying at σ corresponds to a birth-death pair (b, d) in the diagram, where $b = h_\omega(v_b)$ and $d = h_\omega(v_d)$ for unique vertices v_b and v_d. Thus, each off-diagonal point can be traced to a pair of vertices.

As ω varies, each such point traces a curve:

$$x(\omega) = (\langle v_b, \omega \rangle, \langle v_d, \omega \rangle),$$

which parametrizes an ellipse. The projection to the diagonal is also a degenerate ellipse:

$$x'(\omega) = (\langle (v_b + v_d)/2, \omega \rangle, \langle (v_b + v_d)/2, \omega \rangle).$$

C.4 Trivial Geometric Monodromy and Star-Shaped Domains

Following [4], a simplicial complex K is *star-shaped* if there exists $c \in |K|$ such that for all $x \in |K|$, the segment $[c, x] \subseteq |K|$. For such K in general position, $\mathrm{PHT}(K)$ has trivial geometric monodromy: there exist continuous functions $\{\gamma_i : \mathbb{S}^1 \to \overline{\mathbb{R}}^2\}$ (vines) such that the off-diagonal points of $\mathrm{Dgm}(h^K_\omega)$ are exactly $\{\gamma_i(\omega) \mid \gamma_i(\omega) \notin \Delta\}$.

Theorem 11 ([4]) *Let K be a star-shaped complex with vertices in general position. Then* $\mathrm{PHT}(K)$ *has trivial geometric monodromy.*

Each vine either never intersects the diagonal or intersects it exactly once entering and once exiting. For connected K, one vine is of the form $(\tilde{\gamma}(\omega), \infty)$. A vertex v is extremal if it gives birth to a class for some ω, and the number of vines N is at most the number of such extremal vertices:

$$N \leq \left|\mathrm{ext}^K(V)\right|.$$

C.5 Matching Construction and Cost Function

Given PHTs for K_1 and K_2 with n_1 and n_2 vines respectively, we build a complete bipartite graph G with $2n_1 + 2n_2$ vertices—each vine and its diagonal projection. The edge weight between two vines γ_u and γ_v at direction ω is:

$$c(u,v) = \begin{cases} \|\gamma_u(\omega) - \gamma_v(\omega)\|_\infty & \text{if one or both are off-diagonal} \\ \|\gamma_u(\omega) - \Delta\|_\infty & \text{if } \gamma_v(\omega) \in \Delta \\ \|\gamma_v(\omega) - \Delta\|_\infty & \text{if } \gamma_u(\omega) \in \Delta \\ 0 & \text{if both are at } \Delta. \end{cases}$$

At each ω, this graph realizes the bipartite matching problem described in Sec. 2.1. With $n = \max(n_1, n_2)$, we have $O(n^2)$ edges, so the kinetic hourglass maintains $O(n^2)$ elements and runs in $O(n^6 \log n)$ time using heaps.

Open Problems from CCCG 2025

Hugo Akitaya[*] Shahin Kamali[†] Seyed-Mohammad Seyed-Javadi[‡] Frederick Stock[§]

The following is a list of the problems presented during the open problem session at the 37th Annual Canadian Conference on Computational Geometry (CCCG), held at York University in Toronto from August 13 to 15, 2025. This year's open problem session took place on the second day of the conference and featured lively participation from attendees. As in past years, the session served as a catalyst for spontaneous collaboration and discussion throughout the remainder of the meeting.

Open problem sessions have long been a tradition at CCCG, offering a venue for researchers to share intriguing questions, conjectures, and partially explored ideas. The 2025 session continued this tradition, with problems spanning classical themes such as Minkowski-type existence results and modern geometric inequalities.

To the best of our knowledge, the problems below accurately reflect the questions posed during the session. The problems are listed in the order in which they were presented at the conference. Where appropriate, references and attributions to presenters have been included.

1 Logarithmic Minkowski Problem

presented by Joseph O'Rourke

Smith College (jorourke@smith.edu)

O'Rourke posed a problem that was recently presented in the AMS Bulletin [Huang et al. (2025)]. Minkowski-type problems ask when a geometric object can be reconstructed from prescribed geometric data—such as the normals of its faces and measures associated with them (areas, lengths, or other quantities). These questions trace back to Hermann Minkowski's foundational results in convex geometry [Minkowski (1897)] and have inspired many modern extensions. Minkowski showed that a convex polyhedron is uniquely determined (up to translation) by the normals and areas of its faces. Constructive methods for explicitly finding such a polyhedron were later developed by Little [Little (1985)] and Sellaroli [Sellaroli (2017)].

[*]University of Massachusetts Lowell, USA, hugo_akitaya@uml.edu

[†]York University, Toronto, Canada, kamalis@yorku.ca

[‡]York University, Toronto, Canada, mohammad.sj2@gmail.com

[§]University of Massachusetts Lowell, USA, frederick_stock@student.uml.edu

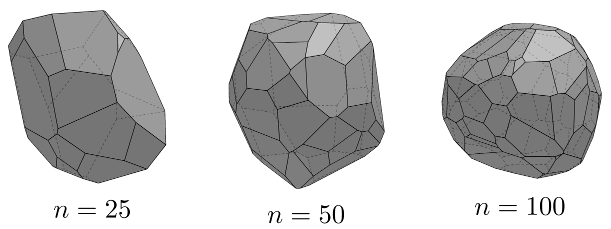


Figure 1: Examples of reconstructed polyhedra with $n = 25, 50, 100$. Figure is taken from [Sellaroli (2017)].

Theorem 1 (Minkowski's Theorem for Polyhedra) *[Minkowski (1897), Little (1985), Sellaroli (2017)] Given unit face normals $u_1, \ldots, u_n$ and corresponding face areas $A_1, \ldots, A_n$, there exists a unique convex polyhedron (up to translation) whose faces have those normals and areas.*

An illustration is provided in Figure 1.

The following theorem presents the two-dimensional version of Minkowski's theorem.

Theorem 2 (Minkowski's Theorem for Polygons) *[Huang et al. (2025)] Let $\ell_1, \ldots, \ell_m$ be positive numbers and $u_1, \ldots, u_m$ be unit vectors in $\mathbb{R}^2$. Then there exists a convex polygon P in $\mathbb{R}^2$ with side lengths $\ell_1, \ldots, \ell_m$ and corresponding outer unit normals $u_1, \ldots, u_m$ if and only if $\ell_1 u_1 + \ldots + \ell_m u_m = 0$ and the unit vectors $u_1, \ldots, u_m$ do not lie in a unit half-circle. Additionally, P is unique up to translation.*

The *logarithmic Minkowski problem for polygons*, introduced by [Huang et al. (2025)], is a variant of the classical Minkowski problem in which the prescribed data are triangle areas rather than edge lengths.

Question 1. (The logarithmic Minkowski problem for polygons) [Huang et al. (2025)] Given unit face normals $u_1, \ldots, u_n$ and triangle areas $A_1, \ldots, A_n$, construct a convex m-gon P in $\mathbb{R}^2$ containing the origin $\mathbf{0}$ such that, for each side s_i of P, u_i is its unit vector and the area of the triangle corresponding to the convex hull of $\{\mathbf{0}\} \cup s_i$ is A_i.

Figure 2 provides an illustration.

Even in this planar, discrete setting, the problem remains unsolved in full generality and is known to be substantially more difficult than the classical Minkowski problem.

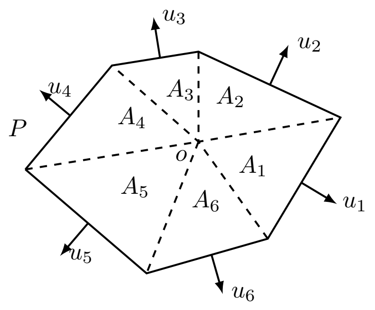


Figure 2: An illustration of the logarithmic Minkowski problem for polygons. Figure is taken from [Huang et al. (2025)].

Symmetric measures can lead to obstructions in the logarithmic Minkowski problem. For instance, if the prescribed unit normals satisfy $u_1 = -u_3$ and $u_2 = -u_4$, and the corresponding areas satisfy $A_1 = A_3$ and $A_2 = A_4$, then any solution must be a parallelogram centered at the origin. This forces $A_1 = A_2$; hence, if $A_1 \neq A_2$, no solution exists (See Figure 3). The challenge is to determine, given the u_i and A_i, when there is and when there is not a solution.

References

[Huang et al. (2025)] Yong Huang, Deane Yang, and Gaoyong Zhang. Minkowski problems for geometric measures. *Bulletin of the American Mathematical Society*, 62(3):359–425, 2025.

[Little (1985)] James J. Little. Extended gaussian images, mixed volumes, shape reconstruction. In *Proc. the First Annual Symposium on Computational Geometry (SOCG)*, pages 15–23. ACM, 1985.

[Minkowski (1897)] Hermann Minkowski. Allgemeine lehrsätze über die convexen polyeder. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, pages 198–219, 1897.

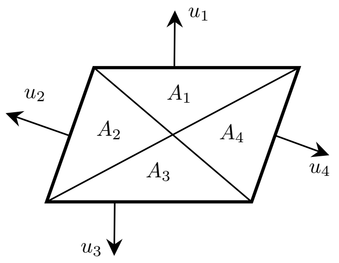


Figure 3: An obstruction in the logarithmic Minkowski problem

[Sellaroli (2017)] Giuseppe Sellaroli. An algorithm to reconstruct convex polyhedra from their face normals and areas. *CoRR*, abs/1712.00825, 2017.

2 Sibley's Convexity Measures

presented by Joseph O'Rourke

Smith College (jorourke@smith.edu)

O'Rourke posed a question recently presented in Mathematics Magazine by Thomas Sibley [Sibley (2025)]. While previous Minkowski problems focus on reconstructing convex bodies from prescribed geometric data, such as normals and areas, Sibley's conjectures are about quantitative relationships among geometric measures.

Sibley proposed two conjectures relating geometric measures that capture how far a polygon deviates from convexity and how much of it remains "visible" from within. For a simple polygon P, let kernel(P) denote the set of points from which every point of P is visible—equivalently, the intersection of the interior half-planes defined by its edges. The area of kernel(P) quantifies the region of complete visibility within P.

For a polygon P, define the following normalized quantities:

$$G(P) = \frac{\text{area}(\text{kernel}(P))}{\text{area}(P)}, \quad A(P) = \frac{\text{area}(P)}{\text{area}(\text{conv}(P))},$$

$$L(P) = \frac{\text{perimeter}(\text{conv}(P))}{\text{perimeter}(P)}.$$

Here, conv(P) denotes the convex hull of P. Intuitively, $G(P)$ measures the fraction of the polygon that is fully visible, while $A(P)$ and $L(P)$ quantify how much of the polygon's area and perimeter are "lost" due to non-convexity.

Sibley made the following conjectures, which compare the visible portion of a polygon to its geometric deviation from convexity:

Question 2 (Thomas Sibley's Conjectures). [Sibley (2025)] Do the following inequalities hold for all simple polygons P?

A: $G(P) \leq A(P)$

L: $G(P) \leq L(P)$.

Note, this question is trivial for any non-star-shaped polygon P, as the area of P's kernel will be zero.

References

[Sibley (2025)] Thomas Sibley. Comparing convexity measures. *Mathematics Magazine*, 98(3):162–174, 2025.

3 Separating Convex Sets

presented by Shakhar Smorodinsky

Ben-Gurion University (shakhar@math.bgu.ac.il)

Note: A very recent result [Chen et al. (2025)] shows that this problem has now been resolved. We include it here for historical context.

Separating families of points in the plane is a fundamental concept in computational geometry (and other domains such as machine learning). Let X be a finite set. A family $\mathcal{F}$ of subsets of X is called a *separating system* if, for every pair of distinct elements $a, b \in X$, there exists a set $F \in \mathcal{F}$ that contains exactly one of a and b [Biniaz et al. (2025)]. Equivalently, every pair of elements in X is distinguished by at least one member of $\mathcal{F}$. The smallest possible size of such a family is known as the *separation number* of X.

For instance, when X is a set of n points in general position in the plane and $\mathcal{F}$ consists of halfplanes defined by lines, it is known that $\lceil n/2 \rceil$ lines are always sufficient and sometimes necessary [Boland and Urrutia(1995)]. When $\mathcal{F}$ consists of disks instead of lines, and no four points are cocyclic, the tight bound becomes $2\lceil n/6 \rceil + 1$ [Gledel and Parreau (2019)].

The problem considered here extends this framework to two sets of convex sets, $A = \{A_1, \ldots, A_n\}$ and $B = \{B_1, \ldots, B_n\}$, where $\mathcal{F}$ is formed by lines and the separation condition applies only to pairs (A_i, B_j).

Question 3 *Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ and $\mathcal{B} = \{B_1, \ldots, B_n\}$ be two collections of convex sets in the plane such that for every $i, j \in [n]$, the sets A_i and B_j are disjoint. Define $f(n)$ to be the smallest integer such that, for any such pair of collections $\mathcal{A}$ and $\mathcal{B}$, there exists a set L of at most $f(n)$ lines with the property that for every $i, j \in [n]$, some line $\ell \in L$ separates A_i and B_j.*
Determine (or bound) the value of $f(n)$.

It is known that if, in addition, the family $A \cup B$ consists of pairwise disjoint sets, then $f(n) = O(n)$. This problem arises in the context of the recent work in [Alon and Smorodinsky (2026)], which relates to an extension of the classical Radon's and Tverberg's theorems in discrete geometry.

References

[Alon and Smorodinsky (2026)] Noga Alon and Shakhar Smorodinsky. Extended VC-dimension, and Radon and Tverberg type theorems for unions of convex sets. In *Proc. Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (to appear), 2026.

[Biniaz et al. (2025)] Ahmad Biniaz, Prosenjit Bose, Jean-Lou De Carufel, Anil Maheshwari, Babak Miraftab, Saeed Odak, Michiel Smid, Shakhar Smorodinsky, and Yelena Yuditsky. On separating path and tree systems in graphs. *Discret. Math. Theor. Comput. Sci.*, 27(2), 2025.

[Boland and Urrutia(1995)] Ralph P. Boland and Jorge Urrutia. Separating collections of points in Euclidean spaces. *Inf. Process. Lett.*, 53(4):177–183, 1995.

[Chen et al. (2025)] Wenchong Chen, Gennian Ge, Yang Shu, Zhouningxin Wang, and Zixiang Xu. A Tverberg-type problem of Kalai: Two negative answers to questions of alon and smorodinsky, and the power of disjointness, November 2025.

[Gledel and Parreau (2019)] Valentin Gledel and Aline Parreau. Identification of points using disks. *Discret. Math.*, 342(1):256–269, 2019.

4 Geometric Graph Covering

presented by Takumi Shiota

University of Hyogo (takumi_shiota@gsis.u-hyogo.ac.jp)

Covering problems in geometric unit-grid graphs arise frequently in combinatorial geometry and discrete tilings. Let G denote a unit grid graph whose vertices are a subset of the integer lattice points in a rectangular region of $\mathbb{Z}^2$ and whose edges are the unit horizontal and vertical grid segments. In this problem, we ask whether the entire edge set of G can be expressed as the union of the boundary edges of a small number of 2×2 tiles.

Question 4 *Let G be a unit grid graph. Given an integer k, determine whether there exist k (possibly overlapping) induced 2×2* blocks*—each block being the cycle C_8 induced by the vertices on the boundary of a 2×2 square—such that the union of the edges of these blocks is* exactly *the full edge set $E(G)$.*

Figure 4 illustrates an example.

This problem can be solved in polynomial time if 2×2 blocks are unlimited in number, because it is sufficient to check whether they can be placed starting from the top-left. However, when the constraint of exactly k blocks is imposed, it is not clear whether this problem can be solved in polynomial time or if it is NP-complete.

Covering all edges as a problem setting has not been extensively studied in existing research. A related problem is the Border Drawing Problem (BDP) in Excel [Iwama et al. (2009)], which has been shown to be NP-hard.

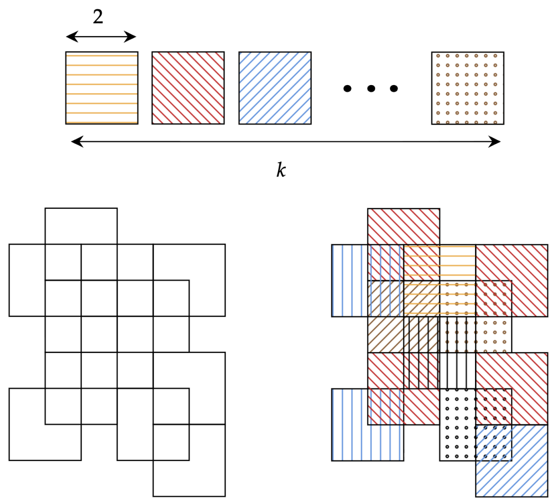


Figure 4: (Top) A set of $k = 12$ blocks of size 2×2, (Bottom left) A unit grid graph G, (Bottom right) A covering of G using the k blocks.

References

[Iwama et al. (2009)] Kazuo Iwama, Eiji Miyano, and Hirotaka Ono. Drawing borders efficiently. *Theory of Computing Systems*, 44:230–244, 2009. doi: 10.1007/s00224-008-9117-y.

5 Shapes of Vertex Sets of Planar Graphs

presented by David Epstein

University of California, Irvine (eppstein@uci.edu)

A recurring theme in graph drawing is understanding how restricted a point set can be while still supporting crossing-free straight-line drawings of various graph classes. Existing results include:

- If all vertices are required to lie in convex position, then the graphs that admit such drawings are precisely the outerplanar graphs. [folklore]
- Restricting the vertex set to lie on a constant number of parallel lines does not accommodate all planar graphs. In particular, the graphs that can be drawn with vertices on k parallel lines have pathwidth $\leq k-1$. Lemma 1 in [Dujmovic et al. (2008)].
- Even allowing a constant number of lines in arbitrary orientations does not suffice to represent all planar graphs [Chaplick et al. (2020)], that is, for every fixed k, there exists a planar graph that cannot be drawn with its vertices on k lines (not assumed to be parallel). More strongly, drawing an n-vertex planar graph may require $\Omega(k^{1/3})$ lines. This was shown in [Eppstein (2021)]. Its references provide earlier but weaker bounds on the required number of lines and on the NP-hardness of finding a drawing on two crossing lines.

In short, several natural geometric constraints on vertex placement are too restrictive to support all planar graphs. See [Bannister et al. (2019), Kryven et al. (2019), Eppstein (2021), Chaplick et al. (2023)] for some related work on limitations arising from constrained vertex locations.

The following question asks whether this limitation persists when the vertex set is allowed to lie on a fixed number of more flexible geometric supports.

Question 5 *Does there exist a constant k such that every planar graph admits a straight-line plane drawing in which all vertices lie on the union of k pairwise disjoint convex curves?*

This is closely related to open problem 16.14 from [Eppstein (2018)] which asks whether, for each n, there is a family of $O(1)$ convex curves that can support all n-vertex planar graphs. This question is also the motivation for Eppstein's upcoming paper [Eppstein (2025)].

References

[Bannister et al. (2019)] Michael J. Bannister, William E. Devanny, Vida Dujmovic, David Eppstein, and David R. Wood. Track layouts, layered path decompositions, and leveled planarity. *Algorithmica*, 81(4):1561–1583, 2019.

[Chaplick et al. (2020)] Steven Chaplick, Krzysztof Fleszar, Fabian Lipp, Alexander Ravsky, Oleg Verbitsky, and Alexander Wolff. Drawing graphs on few lines and few planes. *J. Comput. Geom.*, 11 (1):433–475, 2020.

[Chaplick et al. (2023)] Steven Chaplick, Krzysztof Fleszar, Fabian Lipp, Alexander Ravsky, Oleg Verbitsky, and Alexander Wolff. The complexity of drawing graphs on few lines and few planes. *J. Graph Algorithms Appl.*, 27(6):459–488, 2023.

[Dujmovic et al. (2008)] Vida Dujmovic, Michael R. Fellows, Matthew Kitching, Giuseppe Liotta, Catherine McCartin, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Sue Whitesides, and David R. Wood. On the parameterized complexity of layered graph drawing. *Algorithmica*, 52(2):267–292, 2008.

[Eppstein (2018)] David Eppstein. *Forbidden Configurations in Discrete Geometry.* Cambridge University Press, Cambridge, 2018.

[Eppstein (2021)] David Eppstein. Cubic planar graphs that cannot be drawn on few lines. *Journal of Computational Geometry*, 12(1):178–197, 2021. doi: 10.20382/v12i1a8.

[Eppstein (2025)] David Eppstein. Stabbing faces by a convex curve. In *Proc. the 33rd Int. Symp. Graph Drawing and Network Visualization*, volume 357, pages 29:1–29:8, 2025.

[Kryven et al. (2019)] Myroslav Kryven, Alexander Ravsky, and Alexander Wolff. Drawing graphs on few circles and few spheres. *J. Graph Algorithms Appl.*, 23(2):371–391, 2019.

6 Spanning Trees of Maps

presented by Arash Rafiey

Indiana State University (Arash.Rafiey@indstate.edu)

Consider a rectangular region subdivided by a finite collection of line segments into a set of connected subregions; we refer to this subdivision as a *map*. Each region contains a finite (possibly empty) set of designated points, called *special points*. Let m denote such a map. We define a graph G_m whose vertex set consists of all special points in all regions. There is an edge between two special points whenever the regions containing them share a boundary point (i.e., the regions are adjacent). Each edge uv is assigned a weight equal to the Euclidean distance between the corresponding special points.

A *representative spanning tree* of G_m is a tree that selects one special point from each non-empty region and includes only those selected vertices as its vertex set. Equivalently, the tree induces a connected selection of one representative per region, with edges corresponding to region adjacencies. See Figure 6 for an illustration.

Question 6 *Given a map m and its associated graph G_m, find a representative spanning tree of minimum total weight. What is the computational complexity of this problem?*

This problem is related to the Group Steiner Tree problem, where one is given a collection of groups of vertices and seeks a minimum-weight subtree that contains at least one vertex from each group. The Group Steiner Tree problem is NP-hard, even when the underlying graph is a tree [Garg et al.(2000)], and it is hard to approximate within polylogarithmic factors [Halperin and Krauthgamer(2003)]. In contrast, the problem considered here requires selecting exactly one vertex from each region, and the objective is to compute a minimum-weight *spanning* tree on the selected representatives, rather than a Steiner tree.

Another related line of work considers minimum-weight spanning trees under *neighborhood constraints*,

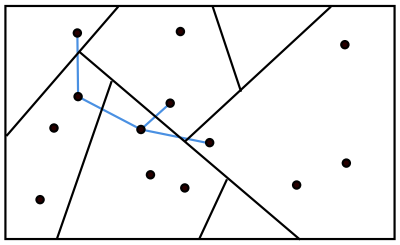


Figure 5: A map and its associated minimum-weight representative spanning tree

where a collection of geometric neighborhoods (e.g., disjoint disks) is given, and one representative point must be chosen from each neighborhood before computing the tree (see, e.g., [Dorrigiv et al. (2015)]). The setting in Question 6 differs in that the representative points are predetermined rather than chosen arbitrarily, and the underlying structure of the map is fixed by its line-segment subdivision.

When the regions are unit squares, the problem is known to have a PTAS [Mirzanezhad and Rafiey (2025)].

References

[Dorrigiv et al. (2015)] Reza Dorrigiv, Robert Fraser, Meng He, Shahin Kamali, Akitoshi Kawamura, Alejandro López-Ortiz, and Diego Seco. On minimum- and maximum-weight minimum spanning trees with neighborhoods. *Theory Comput. Syst.*, 56(1):220–250, 2015.

[Garg et al.(2000)] Naveen Garg, Goran Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000.

[Halperin and Krauthgamer(2003)] Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proc. the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 585–594, 2003.

[Mirzanezhad and Rafiey (2025)] Majid Mirzanezhad and Arash Rafiey. Approximate and Exact Geometric Generalized Minimum Spanning Trees In *Proc. of The 37th Canadian Conference on Computational Geometry (CCCG)*, 2025

7 Bicoloured Point-Separation Rounding

presented by Jack Spalding-Jamieson

University of Waterloo (jacksj@uwaterloo.ca)

Separating red and blue point sets in the plane using geometric obstacles is an emerging theme in computational geometry. Recent work separating two points with weighted obstacles [Spalding-Jamieson and Naredla (2025)] and on approximation algorithms for the unweighted variant [Lynch and Spalding-Jamieson (2025)]. The problem below asks for a "sparsification" of an already separating family of weighted lines—that is, for a small subset that preserves a required minimum separation thickness between red and blue points.

Question 7 *You are given a set of n red points R and a set of n blue points B, and a set of n line segments C. Each segment s in C has an associated weight $0 \leq w_s \leq 1$, with a guarantee that for any path from a red point in R to a blue point in B (a "bichromatic path") will cross segments with total weight at least 1.*

Let $W := \sum_{s \in C} w_s$. Can you output a subset $S \subseteq C$ that blocks (is crossed by) all bichromatic paths, so that $|S| \in O(W \, polylog(n))$? What about $|S| \in O(W)$?

Figure 6 provides an illustration.

A positive (and algorithmic) answer would form a rounding step for an approximation algorithm solving a special case of the "generalized point-separation problem" [Kumar et al. (2022)]. This is possible with standard rounding methods in some limited cases, for example, when the maximum degree of the geometric intersection graph of C is constant, in which case $|S| = O(W)$ suffices.

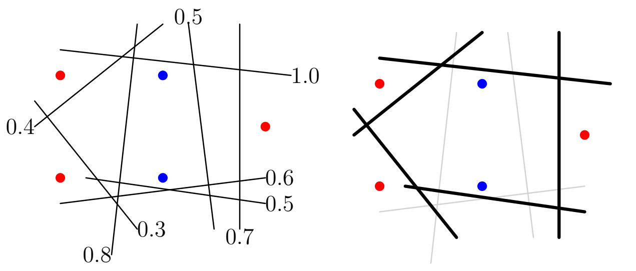


Figure 6: An input for the bicoloured point-separation rounding problem and one possible selection S

References

[Lynch and Spalding-Jamieson (2025)] Jayson Lynch and Jack Spalding-Jamieson. Subquadratic approximation algorithms for separating two points with objects in the plane. *arXiv:2507.22293*, 2025.

[Spalding-Jamieson and Naredla (2025)] Jack Spalding-Jamieson and Anurag Murty Naredla. Separating two points with obstacles in the plane: Improved upper and lower bounds. *Journal of Computational Geometry*, 2025. To appear.

[Kumar et al. (2022)] Neeraj Kumar, Daniel Lokshtanov, Saket Saurabh, Subhash Suri, and Jie Xue. Point Separation and Obstacle Removal by Finding and Hitting Odd Cycles *Proc. the 38th International Symposium on Computational Geometry (SoCG 2022)*, 2022.